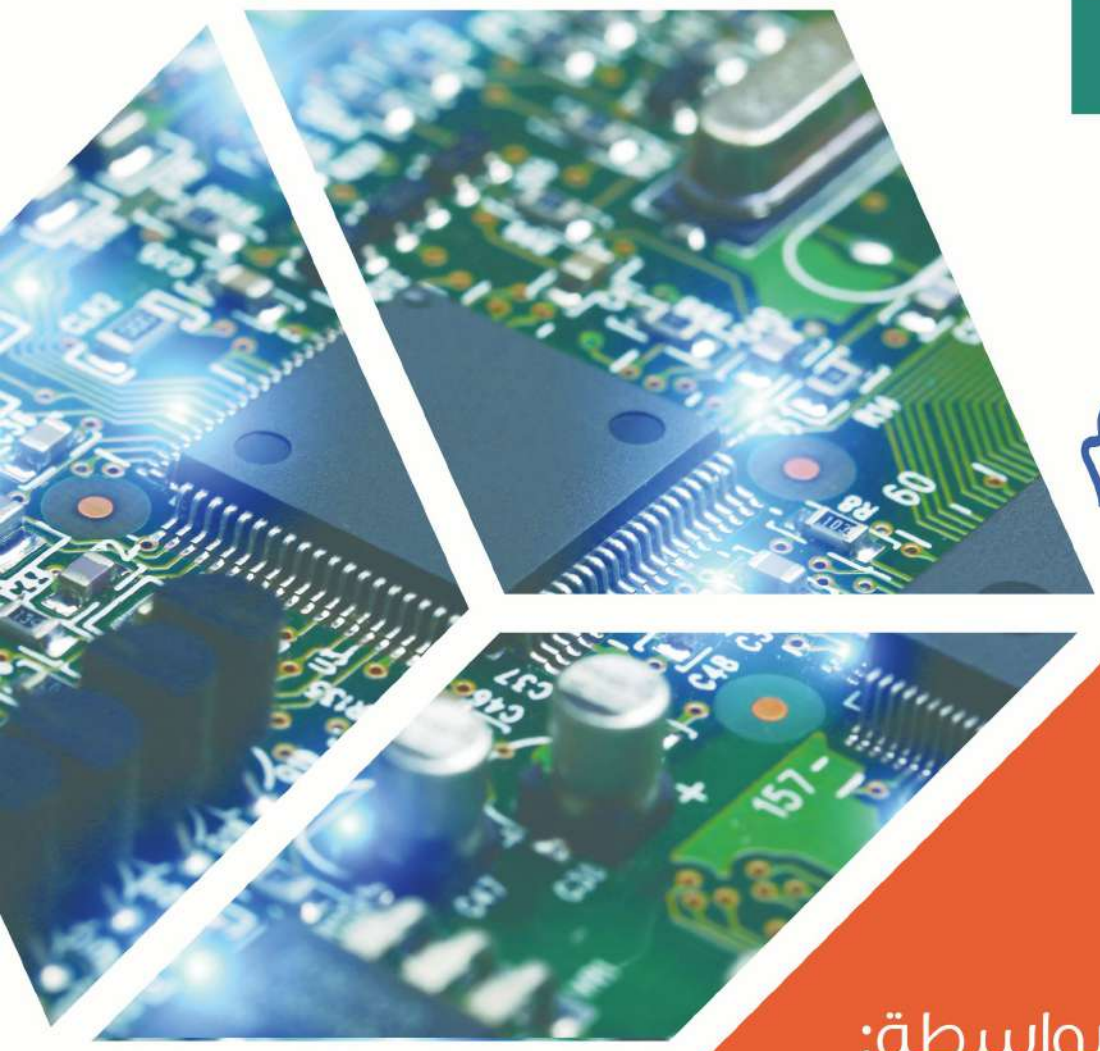
ElCoM
ELECTRICAL COMPUTER MECHATRONICS

www.elcom-hu.com



دوسية إلكم

تم اعدادها بواسطة:

رؤى فايز الرمامنة

دوسية شرح مادة:

أنظمة التشغيل

شرح مفصل للمادة
مضافاً إليها أسئلة سنوات

إرادة
ثقة
تغيير

ElCoM ElCoM HU ElCoM

Memo No. الأحد
Date 13 / 7 / 2025

Ch1

الـ OS موجود بكل الأجهزة إلي بنستخدمها من هواتف، حواسيب وما إلى ذلك
كل جهاز لازم يحتوي على الـ OS ، الـ OS بختلف من جهاز إلى آخر حسب
الوظائف إلي بقوم فيها ، مثلاً لو كان تلفونك سامسونج الـ OS اله أندرويد
لو بنحكي عن حواسيب فرضاً بعضها windows .

* الـ OS عبارة عن : مجموعة من الـ codes إلي بدونها الجهاز عديم الفائدة .
يعني لو اشتريت جهاز وما عليه OS انت بتكون اشتريت أدوات عديمة المنفعة
وما رح تشتغل لانه إلي بشغل الجهاز هو الـ OS .

* رح نتعرف على ① وظائف الـ OS (Functions) ② تركيب الـ OS ومكوناته (Sturcher)

- الـ OS ينقسم إلى 2-Componant : ① Shell (الواجهة) ← grafical user interface
② Kernal ← ما بشوفه ولكن بقوم بكل الوظائف
Sturcher of OS

- الاشي إلي بنسميه application هو بالنسبة للـ OS عبارة عن process
- عنا اشي cpu utilazation : يعني كل برنامج كم بستهلك من طاقة الـ cpu
- memory utilization : عارفين انه الـ memory هي shared بين كل الـ application

* بالتالي رح نشوف الـ OS بصفته الراعي الرسمي لكل الـ application اشي علاقته بالـ recources

* مفهوم الـ threads : كل application (process) تتكون من مجموعة من الـ threads على سبيل المثال الـ word . threads: smaller workers inside the proceses(app)

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

* بالنسبة لل cpu schagaling : أهم مكون بال hardware الموجود عندك هو ال cpu وأهم مكون بال application هو ال OS وهدول الثنين والعين التعاون بحيث انه ال OS هو إلي بعمل cpu schagaling ، يعني هو إلي بتحكم بتنفيذ كل application ، كل ما تغير application رح تتغير وظائف ال cpu لحتى تنفذ أوامر هذا ال application .

- ال cpu بنفذ جميع البرامج ولكن كيف رح يعرف انه اجى دور ال application الأول أو الثاني أو الثالث هاي المهمة منوطة بال OS عن طريق ال cpu schagaling (ترتيب)

- لو عندي أكثر من cpu زي 4-core cpu رح يصير ال ← مضروب بـ 4

* process syncronization : لما تشتري الجهاز المتوقع انه ال H.W recources أقل بكثير من عدد ال applications إلي رح تشغلهم .

- المشكلة لما يكون أكثر من application بتنافسوا على ال memory بالتالي لما app ياخد جزء من ال mem لازم نمنع apple ثاني ياخد نفس المتوقع فرح يصير complete for ressources

* prosece syncronization : يعني انك ترتب ال acesses يعني لو عندي mem location من 10 إلى 20 حجزها ال application الأول لازم تضمن انه ال other application can not access this part

- بالتالي لانه عدد ال H.recorsess أقل من عدد ال app بالتالي في منافسة بين ال app ولازم نرتب هاي المنافسة بحيث انه ال access لل resource يكون معروف من شخص واحد فقط .

* ال prosses syncronization وال did locks مشاكل بتنتج من schgaling وإلي بحل هاي المشاكل هو ال OS

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* رح نحكي عن ثاني HW من ناحية الأهمية بعد ال cpu وهو ال memory ورح نحكي عن ال main memory وال Virtual memory

RAM

وبدونها جهازك ما بشتغل لانه أي app بدك تشغله لازم يتحمل عليها بعدين بصيرلة execution مشكلتها انه مساحتها محدودة (سريعة ومحدودة المساحة)

* في حال انك بتعتمد على ال main اجراءاتك نسبيًا رح تكون سريعة

أما بالنسبة لل hard disk فاضي ولكن بطيء عشان هيك رح نستعير جزء من ال hard disk ونعتبره تخيليًا انه جزء من ال main memory

* بدي هاي الاستعارة ودايمًا رح تحتاجها في حال مشغل application واحد ما رح نتوقع انه ال OS رح يعمل lanch لل Virtual memory لانه ال main mem بحد ذاتها مكفية ولكن في حال مشغل أكثر من برنامج وصار في عبء على ال main بالتالي رح نضطر نستعير جزء من ال hard disk

* لو صار في اعتماد على ال hard disk ، انت وحظك اذا معلومتك بال main mem يكون سريع أما اذا بال Virtual رح تصير slow لذلك وقت ال access رح يصير اله معادلة بدلالة ال hit/miss

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* زي فكرة ال cach هي Quick memory ال main mem بتعملها Backup

* إي رح ناخده هون انه ال cach = main mem

وال Back up = hard disk

* the oprating system (OS) in terms of the hardware: recource allocater

* بوزع ال recources على ال applications وبحل مشاكل التنافس من syncronization أو dead lock

الممسوحة ضوئيا بـ CamScanner

Memo No. الاثنين
Date 14 / 7 / 2025

Mo Tu We Th Fr Sa Su

introduction to oprating system

- رح نحكي عن مكوناته ووظائفه ورح نحكي عن (how os manges the proceses (applications)

* كيف الـ os بحمل الـ app على الـ mem وكيف بضمن تنفيذ الأوامر على الـ app

- تذكر الـ word مثلاً وكيف الـ أكثر من أمر في أعلى الصفحة بالتالي أي app بكون معقد

من ناحية الـ structer يقسم in terms of programing إلى threads ← small exechun unit within the same process (workers / subroutines)

* ورح نحكي عن الـ cpu scheduling التابع لمبدأ (subprogram inside) the Big program

الـ process manegment وكيف الـ cpu فعلياً بنعمل allocation الـ

الـ word ، فرضاً لو بدي أعمل عملية حسابية على الـ word أو exal بالنهاية إلي رح

يعطيك الناتج هو الـ cpu مش الـ tool نفسها ، الـ cpu أو الـ A/U هي إلي

رح تعطيك الناتج .

* لما تكون شغل أكثر من برنامج مع بعض الـ Focase رح يكون على برنامج واحد at a time

وهنا تحت الـ cpu scheduling

- في مشاكل بتصير نظراً لكثرة الـ app وقلة الـ H/W recources رح نسلط الضوء على نوعين

من المشاكل : ① syncronization : لما 2 tool (app) يتنافسوا على نفس الـ recource

(كيف بدنا ننظم الـ access و كم كل app رح ياخذ وكيف ما نتعدى مساحة الأول

على الثاني هاي هي عملية الـ (proceses syncronization) وهي أحد أهم وظائف

الـ os فبضمن انه الـ recource يعطى one application

- الـ dead locks مشكلة بتصير بين الـ applications ولكن ما بنحس فيها لانه بتنحل

on the fly عن طريق الـ os

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• ليش ال OS لازم يعمل maneye لل memory ؟ لانه حجمها صغير ومشتركة بين جميع ال applications وبالتالي ما بنفع ال app اي بتشغلة ياخذ كل المساحة ويترك الباقين without allocating space

• بالتالي رح نشوف أحد أهم وظائف ال OS (how he manges the main and virtual memory)

- ال OS هو مفهوم معقد لذلك لا يوجد تعريف واحد.

• أحل هاي التعريفات بوصفلك ال OS حسب موقعه :

هو عبارة عن برنامج ويميزه عن غيره انه وجوده الزامي وموقعه وسطي بين ال user (intermediary) وال H.W.

• ال computer system فيه المكونات التالية :

user | SW | H.W

لـ انت كمستخدم you can never access the H.W بدون المرور بال OS

H.W ↓ بنزل عليه ال app عشان تستخدمه

OS موجود بال SW ولكنه ال Big Boss لهذا ال SW

- بالتالي ال OS هو جزء من ال SW ولكن هذا ال OS مسؤول عن ال SW وال H.W

- موقع ال OS وسطي لانه لو حطيناه بعد ال H.W رح يصير في خلل بالإجراء لانه انت كـ user ما بتعرف الطابعة مثلاً كيف بتشغل وأصلاً you shouldnot know وإي بعرف هو ال OS لذلك موقعه لازم يكون وسطي عشان يتواصل مع ال H.W وينفذ الأمر وهذا الاجراء بنسميه conveniace (easy of use) أحد أهم صفات ال OS انه يكون سهل الاستخدام

الممسوحة ضوئيا بـ CamScanner

Memo No. _____

Mo Tu We Th Fr Sa Su

Date / /

بالتالي الـ OS هو عبارة عن واجهة بين المستخدم إي بده يعمل app (اجراء معين) و الـ H.W

إي، رح يتم عليه هذا الاجراء •

المتوقع من الـ OS نتيجة موتعة الوصفين:

① execut user program: زي لما تكون كاتب كود فهو برح بنفذه

وبطلعلك النواتج وأكيد بحاجة يستخدم H.W

• ليش ما المستخدم ينفذ البرنامج لليش لازم الـ OS ينفذه؟ لانو الـ user ما بعرف كيف

② Make the computer system convenient to use.

يا

يعني دون تنفذ الـ task بدون ما تعرف التفاصيل

③ use the computer H.W in an efficient manner.

- الـ OS موجود لخدمتك (convenient) لذلك هو موجود بالنص

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• Computer < SW / H.W → لما يصير → computer system < SW / H.W / user
↓
doing actions

- الـ user هو إلي بده يستخدم الـ H.W عن طريق البرامج إلي بنزلها by the sponser chip (OS)

- وجود الـ OS هو إلي بضمنلك الـ Control / Correctness / Conveniance
كل التواصل بين الـ user والـ H.W يصبح ممكن اذا في OS

* أقل اشي لازم يكون موجود حتى الجهاز يشتغل هو الـ H.W والـ OS شغال

- الشكل بـ سلايد 6 هو طريقة مفصلة لعرض مكان وجود الـ OS

① multipul users (ممكن أكثر من حساب أو ممكن يكون seriver)
لو هذا الجهاز كومبيوتر عادي بتكون عامل أكثر من حساب وأي واحد بدك ياه بتروح بتعمل تسجيل دخول لذلك شغلهم بكون serial
أما لو كان servier زي servier الجامعة بصير الـ users شغلهم parall
بالتالي تبصير مهمة الـ OS أعقد لانه بده يحمي الـ H.W من أكثر من user

- الـ user بده يستخدم الـ H.W عن طريق شاشة مبسطة وهي الـ app
وكل هي العملية بتصير برعاية الـ OS

user = application servier = HW

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• التعريف الثاني يندرج تحت : (what os can do)
- Functinal way -

- مصنف أنواع الآلات وايش رح يعمل الـ os بكل حالة

① user : النمط التقليدي ، لابتوب ، distope : مستخدم واحد ولهذا
ما بتخاف انه الـ recourse ما تقدر توزعها عشان هيك بكون convenance : ease to use
(بدون التفاصيل بتقدر تعمل الـ task)

resource : any HW component (cpu, i/o, memory)

- لما يكون مستخدم وحيد مش رح تكون كثير مهتم وقلق من انه الـ HW لا يستخدم بكفاءة
إلي بهم بهاي الحالة ؟ os viwe point : ① سهل الاستخدام
② أداء جيد وقياسه عن طريق
الـ response time (وقت الاستجابة) -
من اعطاء الأمر لحتى بدء تنفيذه

② main frame / minicomputer : هذا كأنه serivers وعليه كذا
مستخدم وهون ما بهمني سهولة الاستخدام بقدر أشياء أخرى زي
الـ resourc utilization ، بالتالي هون الـ response time مش المعيار لانه
لو كان عندك علي مصاته بكون بخدم بغيرك . هذا هو المقياس

resource utilization : percantge of busy time

يعني اذا شغلت الكومبيوتر ساعة والـ HW بضل مشغول 50 دقيقة من هاي
الساعة بالتالي $\frac{50}{60} = \frac{\text{وقت HW}}{\text{وقت الكلي}}$ وافضل نسبة هي (1) أو 100%

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لو كانت 50% يعني نصف الوقت مهدور، ال HW ما بتستفيد منه، بس لما نحكي main fram يعني في concrant user يعني لو في وقت مهدور بكون على حساب ال users ومش قاعدين بياخدوا service، هذا المعيار ما كان كثير مهم في التصنيف الأول لانه في مستخدم واحد فقط أما في التصنيف الثاني هذا المعيار مهم لانه في أكثر من مستخدم ~~...~~ والوقت المهدور على حساب أحد المستخدمين إلي المفروض ال OS بخدمه

(3) workstations :- بتشبه ال main fram ولكن الفرق انه بال main fram ال concernant users على نفس الجهاز أما بال workstations في concernent user على different devices ومشتركين باشي واحد

- يعني بال main fram كلهم بيستخدموا نفس ال memory ونفس ال I/O ونفس ال resources فلازم أخدمهم لانه otherwise رح يعانوا ١٠٠٪ اشتراك
- بالمقابل بال workstation كل واحد عنده ال cpu الخاص فيه وال I/O تبعه وقد يكون مشتركين في ال DataBase يعني هون في single point للاشتراك
- لذلك في حال تحولنا من main fram إلى workstation ال resource utilization كمعيار لأداء ال OS بضل موجود فقط على الاشي المشترك وما بعدد معيار على الاشياء المفصولة (الغير مشتركة).

Handheld : أجهزة صغيرة الحجم زي التلفون واهمهم فيها obtimize Battery life بال disktop أو بال servier انت شابك على كهرباء نمش هامك البطارية بالتالي وظيفة ال OS تختلف باختلاف الجهاز إلي شغال عليه

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لذلك بتلاقي نفس OS ولكن باصدارات مختلفة بعضه بكون servierBased وبعضه userBased ، لانه المتوقع من ال OS يتغير بتغير ال machien

① OS is a resource allocator : (موزع ال HW)

وصف ال OS من خلال وظيفته ، زي لو تطبع كذا صفحة إلي بوزع مهمة الطباعة بين الصفحات هو ال OS ،

② OS is a control program: (متحكم بالبرامج)

برضه وصف من خلال الوظيفة ، هون هو control لل SW (البرامج)

- الفرق بين الوصفين انه الوصف الأول حاول يوصف ال OS بعلاقته مع ال HW ووظيفته انه يوزعهم (زي انه ال cpu تعطي وقت لل app بطريقة Fair and effectiv

* لو عندك (5) برامج و cpu وحدة إلي بوزع وقت ال cpu على هاي البرامج كلها هو ال OS

Why OS is resource allocator? Because the user can't allocat resources By himself

انت كمستخدم ماتعرف توزع الوقت بين برامجك (وقت ال cpu)

* الوصف الثاني : ال OS بحول الانتباه إلى ال program إلي عليه maximize يعني صح انت بتقدر تفتح أكثر من برنامج ولكن بتشتغل على واحد at a time وال OS لازم يعمل switch معك بس تغير ال program

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ال OS هو إلي بتحكم بكل ال program الخاصة بال user وهو إلي بوزع ال resourses على هاي البرامج

- هدول النقطتين هم 90% من إلي بصير على الجهاز

(استخدام برامج يترتب عليه تشغيل HW) ال OS بعمل كل اشي والباقي واجهة بتطلب من خلالها ال task ولكن المنفذ الحقيقي هو ال OS .

- صار عنا ٣ تعريفات لل OS :

① control program for other application that the user use

② resource allocater for HW

③ OS is a middle layer Between user and applications (HW)

• نظرًا لانه ال OS معقد لذلك لا يوجد تعريف واحد له

ال OS مكون من شغلتين :- ① Shell (الواجهة) ② Kernal (core / الأساس)

التعريف بالعادة لل OS بكون لل Kernal لانه ال Shell هي الواجهة يعني لما تدخل على ال OS ويتحمل عندك انت فعليًا بتشوف الواجهة وما بتشوف الوظيفة كيف بتصير

Other definition for OS:

One program that runs all the time, because if he become down the resource allocater become down, control program down بالتالي الجهاز ببطل بشتغل

الممسوحة ضوئيا بـ CamScanner

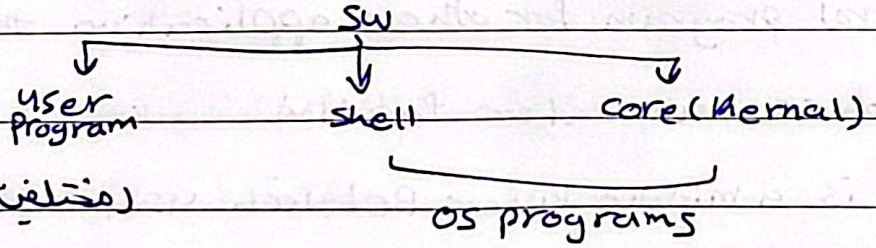
Memo No.
Date / /

- الـ Kernal هو الـ core تبع الـ OS يعني إلي بعمل control للـ program هو جزء من الـ Kernal وإلي بعمل resource allocation هو جزء من الـ Kernal

- احنا هون ما بعمنا الواجهة (shell) إلي بعمنا فعلياً هو الـ Kernal

- كل شيء آخر باستثناء الـ OS (Kernal) اما system program (shell) أو applecation program الفرق بينهم لما نسمع كلمة system يعني متعلق بالـ OS ، أما الـ ↗ هي البرامج إلي الـ user بنزلها بعد ما يشتري الجهاز

SW
- user program
- shell
- core (Kernal)



(مختلفين من جهاز إلى آخر)

OS programs

* بعد ما تشغل الجهاز سو إلي بخلي الـ OS يتحمل وبعدين يصير عليه عاتق كل اشغل؟

- الـ OS قبل ما يشتغل في program هو إلي بحمله واسمه bootstrap program (برنامج ما في مجال تتلاعب فيه أو تحذفه لذلك هو موجود على مكان none erasable وبالتالي ليس الـ harddisk لانه الك accesse عليه وليس الـ memory لانه لو طفيتي بروح اعتوى إلي فيها بالتالي لازم يكون موجود على الـ ROM والـ EPRom هي نوع من أنواع الـ Rom (writeable / erasable)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Bootstrap program: simple program loude the OS بغض النظر عن نوعة his location in ROM in address space named By (Firmware)

+ اسم الموقع إلى البرنامج محمل عليه

• لذلك التسلسل الزمني من لما تكبس ON عشان تشغل الجهاز :-

① الجهاز بشغل برنامج (bootstrap) من ال ROM من الموقع firmware

② هذا البرنامج بعمل تحميل لل OS

قبل هاي الخطوة بده تتأكد اذا ال HW بتحمل لانه بالاساس رح تاخد OS

من ال hard disk بتحطه بال memory

↓ non-Volutile — لو طفت الكهرباء المعلومات بتضل

↓ Volutile — هون المعلومات بتضيع

- ال Firm ware وظيفته الاساسية يبحث داخل ال rom على موقع ال OS داخل ال hard disk ويحطلك ياه بال memory وهيك بشتغل الجهاز

- لو ما في hard disk هل عمكن تتم عملية التحميل ؟ بالتالي ما عندك OS قابل للتحميل ولو ما في memory ما رح يزبط نحمل ال OS برضه

- لذلك قبل التحميل رح نعمل check على ال HW الموجودين (initializes all aspects of system)

① hard disk / خلاشة
② memory (RAM)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Bootstrap program make tow actions befor loud os :-

① بشيك على ال HW الموجود اذا كافي للتشغيل أولا : أهم اشي يكون موجود

① permenant storg (Secondry) (hard disk)

② temperary storg (memory)

if all nesesary HW is exist it will go to next step (loud oprating system). From the hard disk (any permenant storg) to the memory

الملخص:

ال bootstrap بعمل ال OS بعدما يشيك انه ال HW أموره تمام ولما يشتغل ال OS بصير مسؤولية يعمل كل اشي

- الرسمة سلايد رقم 11

عامل تلخيص لل HW الموجود بس مغلياً الاشي المفتاحي إلي بدك ياه مش موجود وحمد ال OS

الموجود بالرسمة : ① cpu وممكن يكون أكثر من واحد

② Bus : بشبك different componant مع بعض

③ one mem chip أو أكثر

④ divices وكل واحد منهم ال adapter / controlar

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• cpu / mem بشبكهم على طول على ال Bus

• ليش الأجهزة تحتاج إلى controlar / adapter

① لأنه هاي الأجهزة اسمها preferance (أجهزة طرفية يمكن تغيرها)

زي الماوس بالتالي بدنا adapter حتى نعمل :

① castimizing or formating information لطريقة التعامل معاه

لانه كل نوع جهاز بختلف بطريقة تراسل البيانات ، زي الماوس

لما تعمل clik هاي تترجم إلى Binary

(أول سبب لوجودة ، حاجة الأجهزة الطرفية للتعريف لانه طريقة التواصل بالبيانات مختلفة)

② the controlar make sure that the Bus is not Busy : بعمل sense وبشوف اذا حدا بستعمل

ال Bus بعمل Block (waiting) (Back off) للجهاز إلي بده يرسل وهذا بسبب وجود sing Bus

③ intrupt : اشارة من الاجهزة الطرفية لل cpu انه

بدي منك خدمة (الاجهزة الطرفية ما بتعمل أي اجراء لحتى

تكون مخبرة ال cpu عنه) بتواصلوا مع ال cpu حتى يعرفوا

انه * فاضي حتى ياخد هاي ال request أولا

• ما بنفع يصير 2-intrupt في نفس الوقت لانه ما بنفع جهازين يبعثوا

مع بعض

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• يتم مقاطعة (intrupt) ال cpu عن طريق الأجهزة الطرفية.

• طيب ليش فرضت انه بدي أقاطعة ؟ لانه ال cpu دايماً Busy ~~Busy~~ بـ tasks معينة.

- بما انه 2-intrupt ما بنفع يصيروا بنفس الوقت عشان ميلش ال controlar موجودين بتضمن one intrupt at a time

- ليش ال cpu وال mem ما بدهم adapter/ controlar ؟

① cpu هو أصلاً بتحكم بال Bus ما بده وسيط

② ال mem ما بدهما لانه عنا اشي اسمه Direct memory access (DMA)

وين ال os بالموضوع ؟ هو ال control إلي بساعد ال cpu عشان بتحكموا بال Bus يعني قبل ما ال cpu يعمل action ال os بضمن انه تتم بسلاسه

• فممكن إلي بشيك على ال Bus اذا مشغول أو لا هو ال os وهو ممكن يعطي أولوية لجهاز ويأخر جهاز تاني

- ممنوع مقاطعتين في نفس الوقت اذا عندي cpu واحد أما إذا عندي أكتر من واحد بنفع

الممسوحة ضوئيا بـ CamScanner

- أول تعريف لل OS كان بناءً على موقعة بين ال HW وال User app

- ثاني تعريف متوجهة نظر ال HW والحاجات resource allocation (توزيع ال resource على ال user)
بحيث اذا موجود عندي أكثر من app كلهم رح ياخذوا جزء من ال cpu time وجزء من ال memory و هذا لانه عدد ال resources أقل من عدد ال app
لذلك لابد من توزيعهم بطريقة عادلة بين ال app

- تعريف ثالث انه control program انه بتحكم بالبرامج اي بنظم استخدم وبعمل managing لهدول ال app وبضمن انهم executed in accuret and effeciwt way

- متى ال disk بدها معلومات لل mem ؟ لما تشغل البرنامج بكون محمل على ال harddisk أصلاً وبمجرد ما بدو يعمله execution رح تودع نسخة على ال RAM

- وظيفة ال controller اذا كان ال Bus مشغول منى بس انه يقول للجهاز استنى ولكن انه يعمل buffering لهاي المعلومات (ال disk مثلاً) بحيث انه ما يضيع لذلك ال diskcontroller رح يخزن هاي المعلومات لحتى ال Bus يصير فاضي)

- لو فرضنا انه ال Bus يستقبل البيانات حجمها Byte وانت متأكد انه ال disk او ال mouse بطلعوا البيانات Byte by Byte ؟ لا، لا، لا،

Memo No.
Date / /

• ال mem + cpu + i/o المفروض يشتغلوا مع بعض ولكن احنا نرتب الوصول للـ Bus ، يعني اذا ابدك تخلي ال cpu يستنى الاجهزة لتخلص ما بعده رح يشتغل ورح يصير ال performance سيئ

- لذلك من الأفضل انه ال cpu وال i/o يشتغلوا concurnetly (متزامن) عشان نكسب وقت

- device controler is in chary means:

① Formating the data

② Buffering the information

③ wating the Bus being avalable

- كل device controlar عنده local buffer والعمدة منه اذا ال Bus مش متاح يخزن فيه الداتا عشان ما تضيع (بشكل مؤقت)

- لمتى رح أضل أخزن بهذا ال buffer؟ لحد ما ال Bus يصير متاح وتطلع هاي البيانات من ال buffer باتجاه ال mem أو أي اتجاه أخر.

* هاي ال Buffer مش دايمًا مليانه وه دايمًا فاضيه تعتمد على حالة ال Bus

- إلي بتحكم بكل ال actions هو ال cpu وبالتالي أي واحد من ال i/o اذا بده يعمل action لازم يطلب من ال cpu إذن (permisson) على ذلك

- أي بيانات بتتحرك بال Bus مسؤول عنها ال cpu سواء جاي من ال i/o أو من ال mem وهذا بحد ذاته مشكلة.

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• بما انه ال cpu عنده عمله الخاص ، يعني البرامج كلها إلي بدخلها هو ال cpu فانت لما تطلب منه يستنى ال i/o لحد ما يخلصوا أو ينتبه عليهم معنى ذلك انك بتقطعة عن شغله (interrupt)

- افترض انه ال cpu دايمًا شغال وعنده مسؤولية أخرى وهي اسيطره على ال i/o
- لذلك ال cpu كل شوي رح يوقف خدمة" للأجهزة الأخرى بمفهوم اسمة ال interrupt

مثال : لو انت كاتب برنامج c++ وعملت executing ، لنفرض والبرنامج شغال ال disk بده ينقل بيانات ، ال cpu لازم يوقف ويشوف هاي العملية عمكن تتم أو لا خلال البرنامج وهو شغال رح كبست على start وطلعت قائمة (إلي أظهرها هو cpu) وهو أصلاً كان عنده task أصلية شغال عليها

(لذلك احنا بنحكي انه ال computer كله interrupt driven) يعني خلي

ال cpu شغال ولما احتاجة بقطعه عن شغله مؤقتًا عشان يعمل الخدمة الفرعية ويرجع لاحقًا.

* وبما انه ال interrupt اساس عمل ال computer ككل *
رح نوضح أنواعة وتعريفة.

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

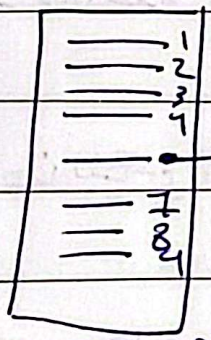
Interrupt :- Signal genarated By I/O gose to perform cpu to some task .

ليش مانخلي performing ال task على ال i/o نفسه ؟ لانه ال cpu مسؤول عن كل اشي حتى لو ما كان في مصلحته

* assumption :

① cpu is Busy By some taskes , So when i/o need service , cpu should Be interrupted

مثال : على فرض عندك code وهو (A. cpp) بما انه رح يكون executable (A. exe)



user click on key bord at this point

هاي مقاطعة لانه ال cpu مشغول بتنفيذ الكود

ولكن لازم يشارك بهاي العملية بالتالي عند هاي ال point

اشي لازم يصير ؟

① stop the current task (A.exe)

② mange how user clike should be handled

③ returns to (A.exe) from where he stoped

- هذا الإجراء بشبه اشي بالاسمبلي اسمه call أو ال subroutine

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

بما انه في مقاطعة لل current task و بروح ينفذ اشي تاني ورح يرد يرجع معناته في saving لبعض المعلومات :

(saving information that help to return to the stop point)

① instruction pointer : IP

② code segment

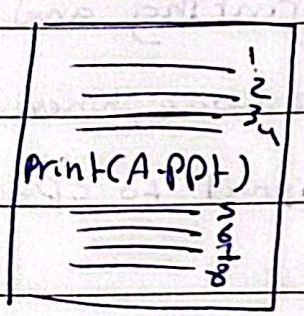
③ statuse of regesters (push in the stack)

When the interrupt happen on the computer ?

① at the begining : start i/o action

② at the end of i/o action

- في المثال السابق لما ضغط كبسة عملية الـ interrupt صارت في بداية الـ i/o action الـ cpu توقف عن العمل



- طيب مثال عن متى في النهاية بتوقف مجموعة من الـ inst بدهم يتنفذوا واجت جملة الطباعة إي بده ينفذها الطابعة برعاية الـ cpu يعني في تدخل لل i/o ، بعدما تخلص عملية الطباعة بدنا نرجع نكمل تنفيذ الـ inst وهون عن 2 sinarios :-

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

من inst ← (1-4) رح ينفذهم ال cpu بدون أي interrupt لما نوصل جملة الطباعة رح يصير عنا 2 sinarios :-

① cpu prepars data, printer prints while cpu is waiting once finsh, cpu ~~res~~ resume with 5,6,7,...

(Blocking, syncronous opration) serial ↓

يعني ما بشتغل ال cpu والطابعة مع بعض وهذا الحكي مش ideal

- لاحظ هون ما في interrupt لانه ما في signal اجت من ال i/o وقاطعت عمل ال cpu ، ال cpu واقف بسبب inst داخل ال code

(No interrupt)

② Non Blocking / Asyncronanous)

ال cpu بسلم الداتا للطابعة وما بضل يستنى بروح يكمل ال code للآخر ولما الطابعة تخلص بتعطيه اشعار انه خلصت في عندك اشي تاني ؟!

- لذلك هون اشعار ال cpu انه الطابعة خلصت هو ال interrupt

cpu prepares data, printer starts printing and cpu resume executing the remain of the code, when printer finshes (A.ppt) it sends interrupt signal to cpu to notify complicitin.

نهاية ال i/o reqused

* ليش لازم اشعار انتهاء؟ عشان ال cpu انا عنده أمر طباعة أخر يقدر يوخده وينفذه لانه عارف انه الطابعة خلصت

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• cpu opration is interrupt driven طبيعة عمل ال cpu مشتقة من ال interrupt

• هل ال memory شغالة بنفس الطريقة؟ هل يوجد interrupt عليها؟

لا، احنا بنعمل interrupt لل cpu لانه المسؤول عن كل اشي

أما ال mem من نفس الاشي لانها مش مسؤولة عن العمليات الفرعية غير المتعلقة فيها، أما ال cpu مسؤول عن كل اشي

- ال interrupt تنفذ بالتعاون مع ال cpu وال OS

* interrupt : HW generated signal or SW generated interrupt

i/o devices — البرنامج نفسه يطلب خدمة من ال cpu

مثال: لو كنت شغال على لعبة والتلفون شحنه نزل ال interrupt من اي نوع؟

(في بداية ال i/o action مجرد ما نزل شحن التلفون أعطاك اشعار)

* عادي يصير في interrupt في البداية أو الوسط أو النهاية ولكن المفهوم بصير أعقد وبندخل باشي اسمه

① maskable interrupt could be delayed

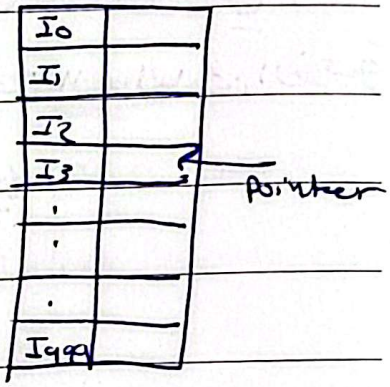
② Proiority interrupt (أمشي الجديد ولا أضل بلشان بالقديم)

عشان يصير simple لـ أفترض بالبداية أو النهاية فقط

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

• لو بدك تلخص حالات ال interrupt القائمة رح تطول وكل interrupt
اله وقت مختلف، كيف ال OS مع ال cpu بدهم يعرفوا ؟ رح يصير
في list بالتالي (بكل ال interrupt المحتمل وقوعة)
على اعتبار عندي 1000 نوع من ال interrupt
أي مقاطعة بتصير لازم تحدد نوعها
مثلاً لو صار interrupt signal من الطابعة بتروح بتدور
مين ال interrupt تبع الطابعة ولنفرض انه I_3
رح يكون عندك بالجدول pointer



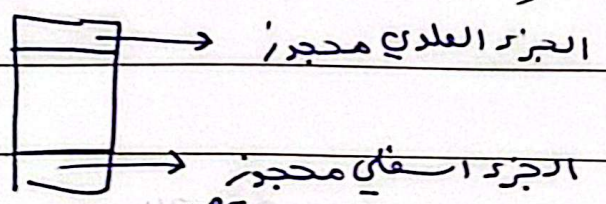
(pointer to interrupt handling subrotine)
(الاجراءات إلي لازم تصير لانه الطابعة طلبت خدمة)
بالمقابل لو كان keybord clik وكان فرضاً I_0
رح يكون اله pointer على مكان تاني

* هذا الجدول بكون مخزن كجزء من ال OS - Kernal

interrupt table (vecture) : list of all interupt and the pointer
you choose depend on the interrupt

معناته الجدول تقلك اذا صار مقاطعة روح مكان كذا، ما بقلك اشي
ال action إلي تعمله هو بس بوديك على ال subroutine وبكونوا مخزنين
بمكان معين بال memory

* بنعرف انه ال memory بالعادة ما بتكون كلها متاحة للاستخدام



الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

cpu: HW genarated interrupt driven

• الفرض الدائم انه ال cpu دايمًا مشغول ولو بدك منه اشي رح تقاطعة وإلي بقاطعة اسمه interrupt

• وأيضًا ال OS دايمًا مشغول ولو بدك تقاطعة إلي بقاطعة اسمة trap

سلايد 16

Once I/O starts you have cpu works

بدنا نشوف وضع ال cpu لما يكون ال i/o شغال :

حالة 1 : اذا كان Blocking

ال cpu مارح يعمل اشي رح يضل واقف يستنى لحتى ال i/o يخلص وبعرف انه خلص عن طريق اشعار بانتهاء العمل (interrupt) وهيك بخرج من حالة الانتظار

حالة ثانية : ال cpu يدخل في waiting loop

وهون معناها مش انه بستنى لا، معناها انه ال cpu يضل شخصيًا يشيك على عمل ال i/o (براقب ال i/o) خلص أولا بالتالي هون ما في داعي لل interrupt لانه ال cpu براقب ال i/o وبعرف متى بتخلص

* هذا الاشي مش منيح لانه ال cpu بده يراقب ال i/o device ال other i/o device مارح تقدر تتواصل مع ال cpu

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* الحكي السابق اسمه polling

(مراقبة)
Polling : cpu is Blocking and monitoring i/o devices until it finshes , the performance is not good

لانه مارح يصير عندك انجاز لاي اشي لانه ال cpu واقف وبستنى

(No interrupt , wasting time)

* non Blocking senario: after i/o starts

فش مانع انه ال cpu يكمل شغلة لحت ال i/o يخلص هون بصير ال interrupt

الزامي والمقاطعة هون الها شكلين :

① system call : ال os بخبر ال cpu انه ال i/o خلص

واذا في اشي ثاني بعملة ال i/o خبره

② device status table: (interrupt vector)

لما يخلص ال i/o بروح لهذا ال table وبشوف كيف

بده يعمل interrupt لل cpu وبعطيه الرقم و ال subroutine

إي لازم ينفذه

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* فكرة الـ DMA (Direct mem address)

لو بدك تشغل فلم حجمه 1G مش معقول انه ينتقل بالـ Bus دفعة واحدة
على فرض انه حجم الـ Bus = 32bit بالتالي رح يروح مقسم إلى قطع كل وحدة 32bit
أول 32bit لما بدها تروح من الـ disk إلى الـ memory لازم الـ cpu يسمح ويعمل
العملية بالتالي هي أول interrupt ولما ترجع تبعث الـ 32bit الثانية رح تعمل
كمان interrupt وهكذا لحتى نخلص كل الأجزاء بالتالي رح يصير لـ
interrupt بمقدار $\frac{1G}{32bit}$ بالتالي الـ cpu مش قادر يشتغل من كثر المقاطعة

أما لو استخدمنا الـ DMA الـ cpu رح يتدخل مرة واحدة في بداية النقل
لانه الـ harddisk بعلم الـ cpu انه بده ينقل لـ mem داتا حجمها 1G
مقطعة إلى أجزاء كل واحد 32bit هون الـ cpu بحكيله بلش ابعث ولما تخلص
احكيلي وبالتالي بدل ما تعمل n-interrupt رح تعمل 1-interrupt وبتكون بالأخر
إلي بالأول مش محسوب لانه نفذ خدمة

(i/o خليه يتواصل مع الـ mem مباشرة ولما يخلص يخبر الـ cpu)

الهدف من الـ DMA :

تحسين performance الـ cpu عن طريق عدم تعطيله كثيراً من المرات
في حالة نقل كمية كبيرة من الداتا

الممسوحة ضوئيا بـ CamScanner

interrupt vector : هو table لانه أسباب حدوث المقاطعة عديدة ، عشان يعرف
ال cpu كيف بده يعالج ال interrupt لازم يعرف نوعه لذلك هذا
ال table يحتوي على جميع أنواع ال interrupt إي ممكن الجهاز يعملها
وبناء على ال interrupt إي بصير ال i/o بزودك بالرقم وال cpu
ببحث داخل ال table وبعرف شو الاجراءات إي لازم يعملها to handle
interrupt x
ولو اجا interrupt y رح يكون موقع معالجته مختلف عن x .

OS / cpu → آلية عملة interrupt drevien
↓ manegar for SW
↓ manegar for HW
(شغالين لحد ما تقاطعهم بعمل آخر)

* لو شبكت جهاز جديد لازم تعرفه عشان يضيف entry على ال table
بنوع ال interrupt إي رح يصير .

OS بعمل عمليات التعريف من تلقاء نفسه حتى يصير اعوضوع Conveniant
(easy to use)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Polling : there is No interrupt , Becouse cpu is Blocking , wating loop so he is already monitor (watch) i/o devices ما رجع وكمل ال task الأصلية
(cpu Busy watching i/o until it finshes)

* على عكس ال interrupt system ، ال cpu بكمل ال task الأصلية وال i/o لما يخلص بعطي اشعار.

* حكينا لما ينقطع عن محله عشان ينفذ اشي ثاني رح يعمل save لل reg ولوين وصل بالتنفيذ وهاي بدها empty location in the stack وبالتالي overhead في حال كان في كتير مقاطعة وكان أحد الحلول استخدام ال DMA

* حكينا عنا نوع ثاني من المقاطعة وهو interrupt generated sw بقاطع عمل ال cpu وال os على حد سواء واسمه trap رايحة لل cpu

* أما ال exception هو برضه interrupt ناتج من sw ولكن سببه وجود خطأ

interrupt → HW
interrupt → sw → trap → عملية صح
sw → exception → عملية خطأ

لم يتمكن من تنفيذ الطلب زي قسمة error = $\frac{1}{0}$

كلهم بدهم خدمة عن ال cpu

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

HW interrupt بوقف عمل ال cpu بشكل مباشر أما ال sw بطلب من ال os
وال os هو بوقف عمل ال cpu

* تذكر جملة ال print إلي بنص الكود حماي trap أما الاشعار نهاية
التنفيذ بتكون interrupt

* mostly the hardware interrupt comes after sw generated
interrupt.

* OS works in tow modes! (Dual mode opration)

① usermode : وانت تارك الجهاز ومش بتعمل فيه اشي يعني كانه
ال os بستنى تطلب منه اشي

② Kernalmode : لما تطلب ال task ولحتى يظهر الأثر تبعها بشتغل
temperary بهذا الحدود

• ليش عنا أكثر من mode ليش ما يكون one mode just ؟ لانه بنرجع لنفس
آلية الحماية HW must Be protected ، فلو بده يكون شغال بال usermode
دائمًا بصير ممنوع ال user يستخدم ال HW (بتطلب بس ماني تنفيذ

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

ولو بده يضل شغال بال Kernalmode رح يضل شغال بس ما بتقدر تطلب

* usermode : عشان تطلب * Kernal mode : عشان الاستجابة

* most of the time user mode have along time *
Kernal mode is vary fast so he did the task
and return you to user mode to give you
high performance.

سلايد 29

لما تكون بالمستطيل إي فوق وتطلب أي خدمة بدها HW مباشرة بصير في
system call (التحول من ال usermode إلى ال Kernal mode) يعني استطيل
إي تحت وبصير عنا trap

- لما تكون شغال ب userprocess يعني app يعني إي شغال عليها sw
لو عملت أي اشي على هذا التطبيق لازم ال OS يدخل بال Kernal mode عشان
يعمل هاي ال service (هون بصير اشي بشبه ال interrupt procsing
والمفروض وقت الاستجابة يكون قصير (كل ما يقصر معناته ال OS سريع)

* أي systemcall يترتب عليه trap لانه رح تقاطع ال CPU/OS
ولما يخلصوا برجع لل usermode

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* لنفرض عملت request معين وكنت بال usermode ال os ممكن يباشر فيه.

- لنفرض انه عملت request وال os شغال بال Kernal mode هل بقدر يباشر فيه؟

لا، لانه هو أصلاً شغال

- كيف ال os بده يعرف انه انتقل لل Kernal mode؟ بنعمله Flag بتشبه ال carry flag أو ال overflow flag فبكون عنا اشي اسمه mode bit وبما انه singl bit فقيمته ممكن تكون 0, 1

أما لو كان ال mode bit = 0	شغال بال othermode	ممكن يكون بأي عن ال 2-modes Kernal / user ما بعني بالزبط
هذا indecution انك بال usermode وأي request بطلب لازم يعمله لانه في مجال	يعني لو فرضنا انه ال 1 لل Kernal mode أي request بصير وال mode bit = 1 should be bufferd لانه بقدرش أعلمها	

* أي instruction بدها ال HW لازم ال os يحول لل Kernal mode عشان يعملها لذلك بصير هاي ال inst اسمها (Privileged) أو Forpeden بتصير بال usermode (زي مثال ال print) بما انه بدك HW مباشرة حول لل Kernal mode (cout / cin / write / read) كلها بدها HW كل هاي ال instruction privileged لانها تتطلب التحول عن ال user mode إلى ال Kernal mode

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

مهم

Define the privileged instruction :- any inst needs HW and oprating system in kernal mode to execute the instruction.

Q: is (int x) privileged ? Yes , لانه ترجمتها لل compiler احجزلي 4 Bytes من ال mem (HW) اذاً ال OS هو اي يعمل ال action مش برنامج ال c++

* لو تنقل ملف من مكان لمكان برضه privileged

* بما انه كل ال inst العا دخل بال HW لذلك صار ال OS هو interrupt driven

أو trap driven هو بخليك تتخيل انه انت بتنفذ ولكن بالواقع ال OS بقوم بكل اشي عشان يجي ال HW لانه انت كمستخدم (you can't manye the HW)

* OS is not ideal (دايماً شغال)

* لما يصير في switching لل Kernal mode هل بنفع ال OS يغيب ؟

اذا غاب ، باقي ال request رح تضل تستنى ورح يصير عندك اشي اسمه (process hogging resources) (يعني اول request حجزت خدمات ال OS لمدة طويلة) احتكار

* أحد الحلول عن طريق استخدام timer ، لما ال OS حول لل Kernal mode وعلق الحلول لو ضل الوضع زي ما هو ما رح أقدر أخدم ال other proceses لذلك مع كل switch لل Kernal mode ال OS بعمل initilize لل timer

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

وهذا ال timer ذكي (smart) اذا ال task شكلها قصير نتيجة

system call زي ال write / read بعطيها وقت قصير اذا شكله

طويل بعطيه timer أطول ال timer مع مرور الوقت رح

يتناقص بعد ما يوصل (0) هذا انطباع انه ال OS كان لازم يخلص اذا ما خلص

معناته في مشكلة بالتالي اطلع من ال Kernal mode وارجع لل user mode

لانه ما بنفع تضل غايب لانه في request بتستنى

(أحد الطرق إلي بستخدمها ال OS في حال الدخول في Kernal mode وعدم العودة)

(Timer=0 ال Kernal عجز عن انهاء المهمة)

* ال OS بحتاج يخزن داتا بتخصه زي ال interrupt vector

(الجدول إلي فيه كل interrupt ووين تروح (pointer) لذلك ال Kernal أو OS

بياخذ مساحة تخزين من الجهاز ، لازم هاي البيانات يتم تخزينها واسترجعها

بشكل سريع لذلك ما بعتمد على array تقليدية يخزن فيها لذلك بستخدم

- linked list
  - singl
  - double
  - circular

  كلهم فيهم pointers لذلك مش شرط التخزين

- Bitmap : مبارة عن Flags يكون ورى بعض
- Binary search tree
  - root
    - أصغر
    - أكبر

  لذلك البحث بكون سريع
- hash map: بتحط الرقم بـ Function بوديك على index direct O(1)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

أنواع الـ cpu :

① ممكن يكون عندك Single processor

② اذا زدت عدد المعالجات سواء من نفس النوع أو من أنواع أخرى بصير اسمه multiprocesser يعني لا cpu صار منها كذا نسخه

والاستفادة : ① increse throughput :

(كم بتخلص taskes في وحدة زمن معينة)

② economy of scale : بتوفر مصاري لانه

بدل ما تشتري جهازين عليهم كل واحد معالج

بتشتري جهاز عليه معالجين

③ incresed Reliability :

اذا عندك أكثر من cpu حتى لو واحد خرب لن تشعر بذلك

الا اذا تم فحصه واسبب انه في Backup (other cpu)

شغالين وهذا اسمه (graceful degradation) يعني لو واحد خرب

النزول بالخدمة بكون الهبوط بالخدمة محدود

لذلك ما بتشعر فيه

Type of Processers : نوع cpu

① asymmetric : كل واحد شكل زي الكرتبو general مع بعضه

② symmetric : كلهم من نفس الشركة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• شو الفرق بين ال multichip وال multi core ؟

multichip ↓ أكثر من شريحة كل وحدة عليها (core) cpu

multi core ↓ شريحة وحدة وعليها أكثر من (core) cpu

• مين أحسن ؟ ما في حدا أحسن من حدا لانه كل وحدة العمامين ات

① multichip : كل واحد عنده reg خاصة فيه ولو واحد منهم خرب ما بأكثر على الثاني

الشيء فيه انه لو عدلت على أحد ال caches في أحد ال cpu باقي ال cpu رح يضلوا يشوفوا القيمة القديمة

(لذلك رح نحكي عن ال cache coherency) عشان نحل هاي المشكلة

(أي تعديل على ال cache ينعكس على ال mem وال other caches عشان ال cpu الثانين ما يشوفوا القيمة القديمة)

② multi core : ممكن تكون ال reg مشتركة أو مفصولة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Virtualization:

افرض انه جهازك إلي شغال على windows اضطريت تستخدم نظام os أخر
زي ال unix هسا هذا النظام ما بلزمك كاملاً فرضاً احتجته مرة
عشان تعمل عليه تجربة في لاب مثلاً الخيار الأول وغير اعوفق انك تنزله
ويصير عندك 2 os ولما تصير تشغل الجهاز يقلك أي واحد بدك ورح
تخسر من ال hard disk مساحة كبيرة

الحل

نعمل Virtualization : يعني خلي ال os الاساسي وينزل الاضافي
إلي ما بدك ياه كتير as a virtuale على الجهاز وبتقدر تنزل
أكثر من واحد ، ال os الاصلي اسمه host والباقين اسمهم
os gest (ما بشتغلوا مع بعض أكيد) ، بالنهاية إلي بعمل
resource allocation هو ال host يعني ال windows فرضاً
يعني لو حاولت تحجز مساحة تخزين باستخدام ال virtuale الواقع انه
ال host هو إلي بحجز

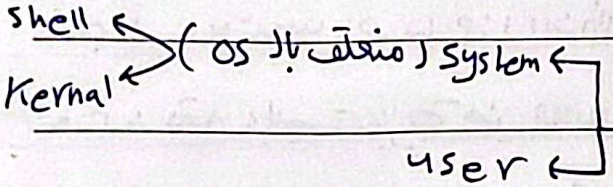
* any tow differant things in computer need interface
named (Virtuale machine manger / hayper vaiser)

بتمر ال request بين ال gest وال host ، اغيرة انه أي برنامج بتكتبه ممكن
يشتغل اما على ال host أو على ال gest وانت بكون عندك الاصلي ومنزل الثاني
لذلك انت اعتفيد من رح تضطر تشتري جهازين

الممسوحة ضوئيا بـ CamScanner

• الهدف من شراء أي جهاز حاسوب انه يشغل عليه مجموعة من التطبيقات على اختلاف أنواعها

Processes : هي التطبيقات إلي بنشغلها ال OS بغض النظر ايش هي



- حكينا سابقاً عن أنواع ال applications
  - system (متعلق بال OS) ← shell ، Kernal
  - user

oprasting system app : أي تطبيق مسؤول عن الجهاز الخاص فيك بنزل ضمنياً لما تعمل

install لل OS ، Shell: الواجهة grafical user interface ، Kernal: core (doing every thing)

trap/system call: frequent switch between shell (usermode) to Kernal mode
- shell ← استقبال أوامر
- Kernal ← تنفيذ ما الأوامر

* بهذا الشابتر رح نحكي عن ال user application وبهذا الشابتر أي app شغال رح نسميه (Process) وإلي نزل هذا ال app هو ال user وإلي بحافظ على أداء هذا ال app هو ال OS

- Process : برنامج شغال وبما انه شغال معناته موقعه بال HW يكون على ال memory

* البرامج إلي مش شغالة بتكون على ال (harddisk)

* مش كل برنامج نازل عندك يكون شغال

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* البرامج إلي مش شغالة رح يضل اسمها application

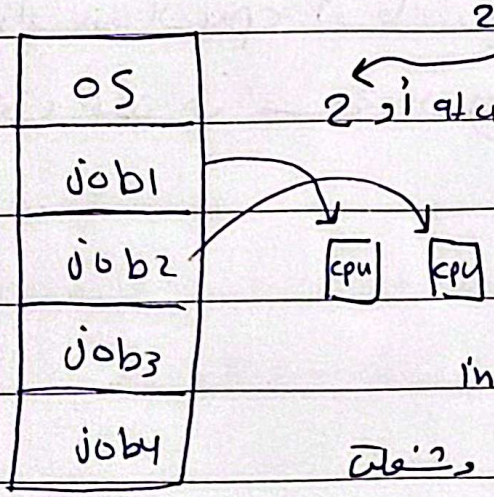
* مجرد ما تشغل البرنامج بصير اسمه process (program in execution) in memory (RAM).

(ليش على ال RAM ؟) لانه البرنامج لما تشغله بصير اله process وبنتقل من ال hard disk إلى ال memory (RAM) ومجرد ما تخلص منه وتسكره أو حتى الكهربا تقطع بتختفي المعلومات من ال RAM

Q: In modern competurs How many process is ON?
more than one process, some of them owened by user ande some of them owend by operating system.

operating system structer :- (سلايد 26 بشابتر 1)
(process = job = task)

* على افتراض عندك single cpu أو 2cpu
المفروض بتقدر تشغل تطبيق واحد at a time أو 2
طيب ليش هون حاملة load for 4 job ؟



ال process قد تتعرض إلى interrupt وتتوقف عن العمل. لو حطيت job 1,2 وشغلت الطاقة القصوى لل 2cpu إلي عندك، رح يضل job 3,4 قاعدين بستنوا

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

- أخذنا من مفهوم الـ interrupt انه أي job ممكن يجيبها i/o signal وتوقف بالتالي لو صار مقاطعة لـ job1 رح تبطل موجودة على الـ cpu وبما انه في other process موجودة بالـ memory ايش التصرف بعدما عطلنا عمل job1 ؟ اجيب job3 واعطيه لـ cpu (مش شرط job3)

* الآن لو استمرينا على هذا الحال الـ cpu رح يضل شغال وبنكون حققنا هدف اساسي وهو cpu utilization (مفهوم مضمونه حاول ما تخلي الـ cpu يوقف لانه اذا وقف معناته جهازك مش قاعد ينفذ taskes

* كل ما زاد عدد الـ jobs كل ما كان أفضل لانها تضمن وجود شيء يبقي الـ cpu شغال (في حال تعطل أي job ، رح نلاقي بديل)

* اذا زاد عدد الـ cpu أو قل يضل من الأفضل زيادة عدد الـ jobs (العائق هون هو مساحة الـ memory)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* multiprogramming (multiprocesses) (multitasking) (Batch system):

multipul jobs بتروح لـ cpu بطلب منها نفسها ، عشان توضح افترض عندك
محمله في الـ mem
(only one cpu) الـ Batch sys كيف بشتغل ؟

* بتحط الـ job إلي بدك ياها على الـ cpu لحتى هي تخلص من الـ cpu وتبطل
بدها ياه (إما خلصت تنفيذ أو ضاربها I/O signal interrupt) وبالتالي هي من تلقاء
نفسها رح تبطل بحاجة الـ cpu والتصرف البديهي بهاي الحالة انك
تختار من الـ job اعتاحين وتحطه بالـ cpu مكانها

(Batch system) : multipulprocess بتعطي وحدة لـ cpu عشان تتحول لـ
executing phase لما هي يبطل بدها الـ cpu بتبحث عن بديل
أخر من اعوجدين وهذا الأمر يندرج تحت عملية الـ job scheduling

job scheduling : هين يكون دوره بدل إلي ترك الـ cpu ، الاختيار مش شرط يكون متسلسل
(عملية عشوائية)

* Timesharing :- نفس الحكي السابق ولكن الفرق انه الـ scheduler
ممكن يفصل الـ process عن الـ cpu حتى لو انها حابة تكمل

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

- مثلاً job1 انحطت على ال cpu وكان ممكن تكمل لدقيقتين على سبيل المثال
بعد 30 ثانية ال cpu scheduler رح يشيل هاي ال process ويختار وحده
ثانية (من مبدأ العدالة fairness) انه إي بنحط على ال cpu ما رح
ياخد ال cpu للأزل، انه هو بعد فترة اجبارياً رح ينشال من ال cpu
وينعمل allocation لواحد ثاني

- ممكن تتخيل اشي شبه ال Blocking, ال non Blocking

① الحالة الأولى
multiprograming
(Batch system)
متى ما ال process
انحطت على ال cpu
هي بتتركه لوحدها نتيجة:
① interrupt
② termenation

② Time sharing
متى ما ال process
انحطت على ال cpu
بصير اختيار ال OS
انه يخليها أو يشيلها

* مفهوم ال response time (كم بده لحتى يبلش بالوظيفة) يصبح غير مهم
في الحالة الأولى لانه ما عنده صلاحية يشيل ال job
أما بالحالة الثانية لانه يشيل ال job و بحط غيرها فهو بحاول ما يزيد
ال response time على ال jobs الأخرى ويبلش فيهم بسرعة وما يأخرهم كثير
لذلك يصبح مفهوم ال response time مهم في الحالة الثانية

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

الـ computers في هاي الأيام من أي style ؟ Timesharing

• لو بدي أحول اهتمام الـ OS لـ process ثانية كل إلي عليك انك تختار other application عن طريق الـ graphical user interface الـ OS بشيل القديمة من الـ CPU وبحمل وحدة جديدة ، (القديمة بتظل بالـ mem لانه مدام صفحة التطبيق مفتوحة ممكن بأي لحظة ترجعلها) بالتالي بخليها موجودة بالـ memory ولكن بحولها للاحتياط

- multiprogramming فية الكثير من الـ processes ولكن لانه كل وحدة منهم بتترك الـ CPU لحالها بالتالي بتخلي الـ OS أقل تعقيداً

- Timesharing مهمة الـ OS أكثر تعقيداً لانه ممكن يفصل process بينما هي حابة تكمل execution من مبدأ (العدالة بين الـ processes)

* Batch system : وحدة بتنحط على الـ CPU ولما تترك الـ CPU بيجي غيرها

* Timesharing : multipule user applecation your time is shared bettwen them

• Job = process = task → program in execution

- لو البرنامج مش شغال (موجود على الـ hard disk) ما بصير اسمه برجع اسمه program أو app

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بال linux بالعادة بنستخدم (task)

* لو بشكل عام ممكن استخدم job / process

• أي برنامج عشان تشغله بال memory شو الجزئيات إلي رح تتحمل؟

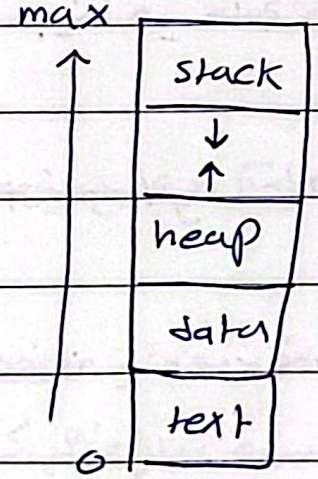
① البرنامج نفسه بروح على ال Code segment , text section

② البيانات إلي بشتغل عليها البرنامج ← data segment

③ اذا البرنامج عندك بعمل push , pob ← stack segment (temperary story)

④ program counter بعدّلك وين البرنامج وصل بال execution (instrection pointer) IP

* ال Job مقسمة داخلياً إلى multipul segment



(0 - max) ال task الوحدة ايش اجزاءها داخل ال memory

• text section : بتحمل داخله برنامج c++

• Data : المتغيرات سواء vector, array

• heap : delete / new

• stack : push / pob

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• بما انه الـ OS مابعرف من البداية كم يحجز بدقة فهو رح يحجز requirement معينة قابلة للزيادة

• (o-max) اعساحة إلي حجزها الـ OS وباعتقادة انها كافية لـ process

• بالعادة بحجز أكثر من اعساحة إلي شوفها مناسبة

• اذا الداتا زادت عن الحد إلي توقعه رح ياخد من الجزء الفاضي ↓ ↑ ويضيف على الداتا

- الجزء الفاضي ممكن ينضاف لأي حد ابده مساحة زيادة.

- instruction pointer/ program counter هو عبارة عن regster ولكنه مش موضحة بالرسمة سلايد 6

مش شغال
↓
Q: Is it possible that one program make multipule processes? Yes
↑ شغال

* انت بتنزل powerpoint كتطبيق وماني حد أقصى من ملفات الـ powerpoint إلي شغلها

* برنامج واحد تشغل منه many instances كل instant تعادل one process

سؤال: هل بحجز لهم نفس اعكان بالـ memory؟ 8

الممسوحة ضوئيا بـ CamScanner

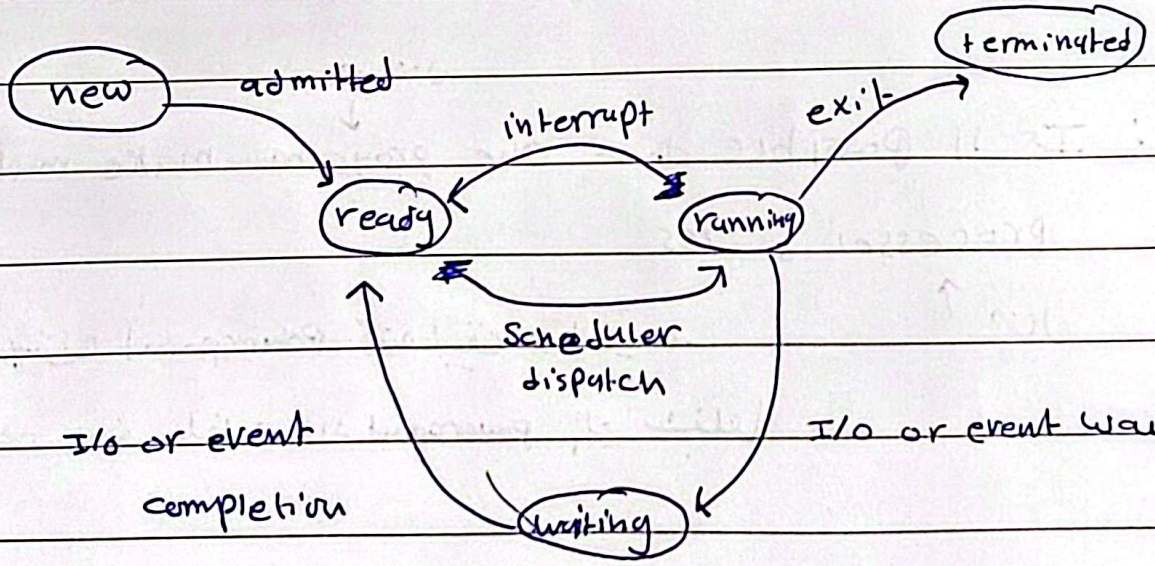
Memo No.
Date / /

① Servier process comes from multiphle users

② single machine process comes from single user

• ال OS برعى مجموعة من ال process كلهم بال mem وعشان نخلي ال cpu utilization علي بنزود عدد ال jobs في حال وحدة منهم تعطلت أو خلصت تيجي وحدة ثانية مكانها

• Lifetime for any process passes through a finit : سلايد 8
لا تنتهي
state machine.



الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

OS

① new : ضغط على البرنامج عشان تشغله ولسا ما ظهر قدامك ← allocating resources

② ready : لما يشتغل البرنامج تماماً

it is a process ← cpu can execute

(بحجز)

وبحدد

- program counter (PC)
- set of regsters
- space in memory

new : it is not process yet

admitted (يعني دخلت على ال system وأصبح job جديد في ال mem)

ready : هي جاهزة للتنفيذ ولكن هل ال cpu بلش فيها ؟ مش شرط لانه بعتمد على :

① كم processe عندك ② هل ال cpu مشغول أولا

③ هل عندك cpu بكفو أولا

∴ إذا ال ready state تعني انه ال job محجوز لها resources وهي موجودة بال memory بانتظار أوامر ال cpu انها تتنفذ

③ running state : بتتحول ال job لهاي الحالة لما يجي دورها بالتنفيذ

* هذا الأمر بناء على مين ؟ ال user هو بختار أي تطبيق يشغل

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• مع كل سهم بالرسمة بحكيلك ايش ال action إلي صار.

① بمرحلة ال new حجزنا ال resources وأصبح جاهز وتحول لمرحلة ال ready

② من ال ready إلى ال running إلي بدخلها هماي الحالة هو ال scheduler dispatch

↓

جزء من ال Kernal

dispatch = transfer

إلي بختار وحدة من ال process عشان يعطيها لل cpu ، ليش هو دايما بختار؟ ليش مايعمل run الهم كلهم؟ لانه عندي process أكثر من ال cpu على الأكيد

لو عدد ال cpu نفس عدد ال process فش داعي يختار كله رح يتحول من ready إلى running

③ مجرد ما ال process صارت بال running state معناته هي under execution • في عدة أمور ممكن تحصل بهماي المرحلة

Ⓐ interrupt : هي حالة تكمل بس اجالتي وقاطعها بالتالي لازم تنشال ويجي غيرها وعشان يجي غيرها ال process إلي كانت running بتتحول waiting state وبختار من ال ready أي وحدة من ال process تنحط على ال cpu

Ⓑ exit : في حال كانت قصيرة وخلصت تنفيذ رح تتحول من ال running إلى terminated عن طريق system call اسمه exit

- exit اجراءاتها عكس ال new يعني رح ترجع ال resources

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

ال Process إلي بتروح لل waiting مارح تضل هناك لانه ال event

إلي بسببه تحولت لل waiting رح يخلص وبالتالي بترجع candidate في ال

ready state انه يرجع ال os يختارها تتنفذ

لو بدك ترسم ال cpu على الشكل وبين مجالس تحطه ؟ على ال running

Q: How many process in (new state) ? one

لانه احنا بنشغل برنامج برنامج وبنحجز لوحدة وحدة ، ما بقدر يحجز لأكثر

من وحدة at the same time

Q: How many process in (ready state) ? many (N)

Q: How many process in (running state) ? the same as cpu number

اذا عندك cpu وحدة معناته بتشغل برنامج واحد اذا عندك 4 بتشغل 4 برامج وهكذا

Q: How many process in (terminated state) ? many

Q: How many process in (waiting state) ? many

وعددهم بتحكم فيه عوامل عدة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

∴ ال life time تبع أي process هذه المراحل الخمسة إلي بالشكل السابق

Q: How many traps exist in the figure?

① admitted: بدك تحجز resources وبتحاول تتواصل مع ال HW ومصفوع انت تعمل هذا ال action ال os بعمله عنك بالتالي هون 2 يصير في switch من ال usermode إلى ال Kernalmode

② ال process إلي بتكون بال ready شو بده يحولها للـ running؟ ال os وبرضه صار في switch من ال user mode إلى ال Kernalmode

③ من running إلى terminated برضه في HW بالموضوع لاني بدي أرجع ال resources إلي حجزتهم

بالتالي كلهم action على process من مكان إلى مكان لذلك كلهم بدهم trap / system call وبتحول ال os إلى Kernal mode عشان ينفذ

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Process Control Block (PCB) : Collection of information

OS must know about the process

(ما هي المعلومات إلي ال OS لازم يعرفها عن ال process حتى يعملها track)

① Process state : مهم يعرف الحالة لانه ال OS لما بده يختار وحدة لل execution

رح يختار من ال ready مش رح يختار من new, running, waite

أو terminated وهكذا

② Process number : (uniquely identify the process) ما رح

يكون عندك 2 process اللهم نفس الرقم (بميزهم عن بعض من خلاله)

③ Program counter : وين وصلت ال process بالتنفيذ عشان في حال صار interrupt

وبعدين رجعت تعرف لوين وصلت وتكمل شغل

بلزمني خاصة لما تتحول من running إلى waiting أو من waiting لـ ready

④ registers : فيهم النواتج لازم يتخزنوا عشان تحصل على ناتج بالآخر

⑤ memory limits : زي من (0-max) بتخزنها عشان لما بدنا نحجز لـ other process

يكون عارف انه هاي المساحة مأخوذة سابقًا وكمان عشان لما يخلص تنفيذ يرجع هاي

المساحة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

⑥ list of open files : لما تيجي مرات تحذن ملف مفتوح بكيلك ما بقدر
سكر اعلف بالأول ، كيف بعرف انه ال app فاتح ملف معين
عن طريق هاي المعلومة ، بعرف كل process شو ال i/o devices
أو resources إلي حاجزيتها

∴ ليش كل هاي المعلومات ال OS بضل مخزنها و بتذكرها ؟ عشان يقدر يعمل
correct opration on process

بال OS كم Block رح تلاقي فيه ؟ زي Block سلايد ٩ ؟
— نفس عدد ال programs إلي شغالين ، لانه كل برنامج شغال ال OS بحتاج
لهاي القائمة حتى يقدر يرعاه

• حسب ال OS كيف بخزن هاي الداتا ؟ ① Singl linked list
② Dubly linked list
③ circular linked list
④ hash table
⑤ Bit map
⋮

* ال OS يحتاج إلى داتا فرعية تساعده على أداء مهماته

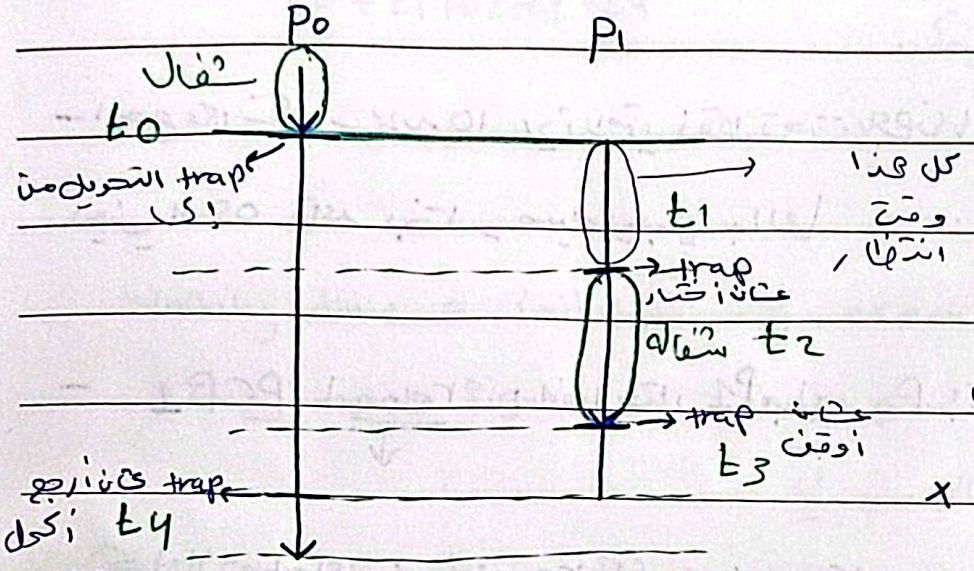
الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

CPU switch from process to process (Contex switch)

سلايد 10

الرسمة بتوضح اشي إلي لازم يصير في حال انتقال ال process من ال running إلى ال waiting



Process X no longer can execute.

لازم نشيلها ونحطها بال wait

ونختار process y بديلاً عن x

① لازم نخزن معلومات عن x أولها:

• وبين لازم نرجع بعد انتهاء التوقف

② Process y إلي بدي أحملها مش شرط ما تكون اشتغلت قبل هيك لهيك

لازم أشيك على ال (PCB)

من ال PC بعرف اذا بتنفذ لأول مرة أولا

PC: قيمته نفس قيمة أول inst

معناته هاي أول مرة بتنفذ

P0 وقفت لسبب ما حكالنا ياه ولكن الأكيد انه مش بسبب انها خلصت لانه تحت بترجع تشتغل كمان مرة

التوقف اما ① i/o أو ② schedual decision

الاجراءات إلي رح تصير:

① save PCB0

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لو بدك تعمل zoom على ال PCB0 رح تلاقي:

① ال state كانت running لازم تتبدل وتصير waiting

② update program counter

- ارجع على شكل سلايد 10 رح تلاقي نقط تحت بعض بالرسمة وهي تمثل cpu scheduling

يعني ال OS قاعد بختار مين يجيب بدالهما

- reload PCB1 : انه اختار P1 بدل P0 إي توقفت

حول ال state من ready إلى running

* لما تتحمل ال process Block إي وقع عليها الاختيار ال cpu يستأنف عمله

ونفس ال action السابقة تتكرر بشكل دائم

• P1 تنفذت لمدة محددة وبما انه مش راجع أشو فهما تحت معناته قد

تكون خلصت أو توقفت في انتظار ال i/o

• مصير P1 :

① terminated if done

② waiting

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

احسب وقت الـ cpu utilization ؟ $= \frac{\text{فترات العمل}}{\text{الفترة الكلية}}$

يا ريت يكون الوقت بالـ contex switch أقصى ما يمكن $= \frac{t_0 + t_2 + t_4}{t_0 + t_1 + t_2 + t_3 + t_4}$

شوف الشكل سلايد 10 :

waiting time = idle , executing time

بتحس انه الـ OS إلي شغال على هذا الـ figure مش منيح

لانه وقت الاختيار عالي نسبيًا

Q : on the same figure how many trap exist ?

هذا في حال ما صار عندي , 4-trap

أي trap داخلي زي write , read , print

كون الـ (4 trap) فقط to switch from usermode to kernal mode

* لما يكون البرنامج معقد بكون مقسم داخليًا إلي subprograms واسمها Threads .

لو بدك تستخدم 2-threads على الأقل لازم في 2-cpu حتى تشتغل على الأقل

* if you have many threads and single cpu ← مارح تستفيد

اشي

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Process control Block → (programmed) تابع لـ (linux)

والـ process control Block صار اسمها task structer

← شو الـ os بخزن داتا حتى يرعى الـ process

① Pid t - Pid = Process identifier - process name/number

② long state =

data type , OS doesn't have words (ready, running, ...)

so, he give them a numbers معناته الـ state

تبعه الـ process ما تبكون كلام تبكون رقم يدل على الحالة.

③ unsigned int time slice = كم الـ process مسموحلها

تحجر من الـ cpu فترة زمنية بعد هيك بتترك الـ cpu وبيجي غيرها من مبدأ

(العدالة) رقم

④ Struct task struct *Perant = أي process تبعمل

الها أولاد بالـ os ومسؤولين بتأثر وبين الأب موجود

عن subtasks

* من المنطق بضل الأب بدون الابناء بس العكس غير صحيح

- الـ os بعمل tracking لـ parent عشان to kill the pearnt if

the child die for some resone

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

⑤ struct list head children = multipul children

وبالتالي عندي data structure بدلني عليها كلمة ↑ يعني عندي linkedlist

* معرفة الاب تتطلب monitor ومعرفة الابناء تتطلب linkedlist

⑥ struct files - struct *files = list of open files

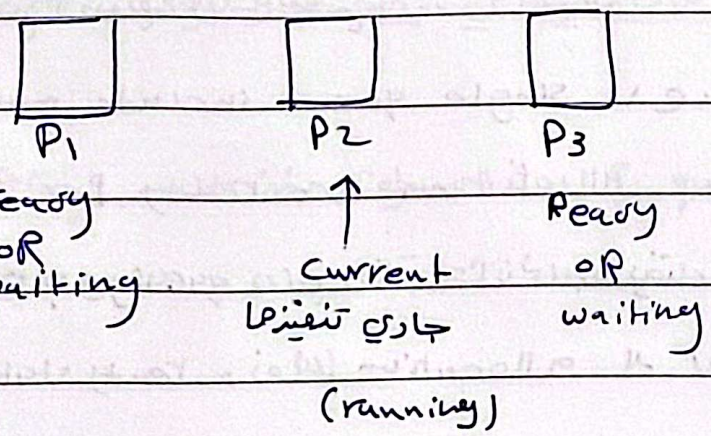
عدد الملفات المفتوحة ولما ال os يعتبر ال process بعمل free لل files

وما بعود يتردد انه اعلن لسا مفتوح

⑦ Struct mm - struct *mm = address space /mem limits

انه ال process حاجز من مكان كذا إلى كذا وهاي القيمة تخزن في structer

سلايد 12 : الشكل إلي تحت بفرجيك النظام ككل



الممسوحة ضوئيا بـ CamScanner

Memo No. الاثنين
Date 21 / 7 / 2025

من المهم انه تعرف بال Linux ال process control Block اسمها task_struct

↳ Collection of information → بتساعد ال OS يعمل tracking لل process

والشكل إلي تحت في سلايد 12 هو عبارة عن linked list بتساعد ال OS يعرف أي process شغالة باستخدام ال current pointer وبالتالي من الشكل بنلاحظ الوحيدة allocate in cpu هي ال process إلي بالنص والباقي اما بال waiting أو بال ready

ال OS لما يختار process مكان process هاي العملية اسمها scheduling عنا أكثر من queue عشان ترتب حالة ال process

① Job queue : set of all processes (new, terminated running, waiting) not usful for OS لانها تضم الجميع دون تفصيل حالتهم

② Ready queue : single queue include many process all of them are ready for execution

مهمة ال OS بعمل queuing بهاي الحالة عشان لما بده يختار ال يختار وحدة بينهم ما إلي بال ready state ونعملها allocation لل CPU (one of them will run soon on cpu)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

③ device Queues (i/o) : كل واحد من ال i/o لازم يسجل هو

قاعد بخدم أي process وعلى جميع

الحالات كل ال process هون بحالة ال waiting

• طيب ليش في أكثر من queue بهاي الحالة؟ شوف سلايد 14

ready queue : is a single queue include PCB7 , PCB2

جاهزين يصيروا run ومنتظرين يقع عليهم الاختيار من ال cpu (they are not witing anything)

(الاختيار عشوائي) يعني مش لازم 7 قبل 2

• تحت ال ready queue عندك 4-waiting queue

are waiting for some event

magnatic tape | disk | terminal (scnreen)

الاختيار بالترتيب

$PCB_3 \xrightarrow{\text{بعدين}} PCB_{14} \rightarrow PCB_6$

الترتيب مهم هون

* ال running هل يحتاج إلى queue؟ لا، لأنه ال cpu لما بده يبلش بـ process

ببلش بوحدة فقط بالتالي ماني داعي تعمل buffering لأكثر من process

* ليش بنستخدم queue؟ لأنها وسيلة لحفظ الدور FIFO

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• شكل سلايد 15 عشان يوضح مسببات الانتظار لأي process بتعمل run

• لا يوجد طريقة تتحول فيها ال process من ال waiting لـ running بشكل مباشر

* إلي كان مكتوب على الشكل السابق هو سبب واحد وهو ال I/O events

أما هون عنا ٤ أسباب :

① I/O request : أي process بتحتاج I/O بالتالي لازم يصير interrupt وتتحول ال process إلي احتاجت I/O إلى waiting وتبدلت process ثانية لحتى يخلص ال I/O وبس يخلص بترجع على ready

② حكينا عن ال Timesharing : لو عندك 3 process وبدك تكون عادل بينهم رح تعطيهم كلهم نفس ال (Time slice) حتى لو بعطيهم بده مدة أطول

ما بعمله ال cpu utilization هون بعمله ال response time

(لما يخلص الوقت المحدد بترجع على طول على ال ready كمان مرة)

③ forka child : ال process الأصلية إلي كانت شغالة على ال cpu (P1) عملت ابن الها اسمه (P2) ، الأصل ينتظر الفرع

* the child will execute first than pearent will contain

عشان هيك حاط دائرة بالرسمة

وأي دائرة تعني لسا في waiting لازم يصير قبل ما ال process تتحول لـ ready state مرة ثانية

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

بما انة الاب P1 عمل ابن P2 بالتالي الابن رح يتنفذ بالبداية وبعدين الاب ، فترة تنفيذ الابن تعتبر فترة انتظار عند الأب .

(4) Wait for an interrupt : ال process توقفت نتيجة ال i/o مجرد ما يخلص ال i/o بدك ترجعها

شو الفرق بين فرع (1) وفرع (4) : interrupt ، Blocking , non Blocking

فرع (4) بضل يعمل monitoring بظاهرة اسمها Polling براقب ال i/o لحتى يخلص

فرع (1) تبروح بتشوف اشي ثاني تعمله لحتى يخلص ال i/o

* إي بحرك ال process من مرحلة إلى مرحلة هو ال os

* شو معنى انه ال os شالهما من ال waiting وحطها بال ready ؟ وكيف بغير حالة ال process ؟ عن طريق تغير ال state الخاصة فيها الموجودة في ال process control Block .

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Schedulers: Subpart of Kernal, moves process from place to another place By using process control Block

(لازم يكون موجود)

① Short-term (cpu scheduler): بشيل من ال process الجاهزين وبعطي ال cpu من ال ready لل running، عليه عبء اشغل ما برتاح، لازم يكون سريع

② long term (job scheduler): بشيل ال process وبحطها بالنظام من البداية يعني بدخلها على ال life cycle. (تخيلي ومش دائماً موجود)

وأصلاً ال user بالواقع هو إلي بجيب ماكن جديدة

③ mide term: بشيل من ال process إلي بال mem وبوديها على ال hard disk في حال بطل في مساحة بال mem

بساعده ال kernal, OS

* الاشي اشترك بينهم كلهم انهم بحركوا process بس الفرق من وين لوين

بتحكم بال degree of multiprogramming: كم برنامج موجود بال memory مهما

user هو إلي بتحكم

Job1
task2
process 3

كانت حالته سواء running او waiting او ready

مثلاً هون في 3

بنسعى نكبر هذا الرقم

عشان ال cpu تضل شغالة

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

① كنا قسمنا ال process حسب حالتهم إلى : waiting ، ready ، running
أو terminated

② الآن رح نقسمهم حسب وضعهم بال execution إلى :

Ⓐ i/o-bound : معظم وقتهم بتحتاج
ال i/o فبتوقف
أغلبية الوقت doing i/o معناته
هيك يصفي كل الوقت انتظار
وكمان many context switch و cpu utiliz-
قليل مش لانه ال system شيء
بل لانه حظك اجاك كلهم من هذا
النوع

اجمع كل ال execution = cpu time
واجمع كل ال waiting
واجمع كلهم مع بعض
* ما بقدر أتحكم بالوقت ، إي بقدر
هو طبيعة البرنامج
* ال optimail بكون كل execution
وبالعادة ما بكون موجود

Ⓑ cpu-bound : لو كان أغلب ال process من هذا النوع ، بالاصل هذا
النوع أغلب الوقت شغال وبضل يتنفذ لذلك ال cpu-utilization
رح يكون عالي ، بحتاجوا ال cpu لوقت طويل وما بوقفوا

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• لو كانوا كلهم من نوع cpu-bound رح يكون ال utilization عالي ولكن مش عادل لانه أخر process انضمت رح تستنى كل إي قبل وبالتالي ال response time رح يكون سيء

• لذلك ال process يكونوا Mix بين النوعين أفضل

* Mide term : بحاول يحافظ على وضع ال memory وما يخليها معتلة تماماً وبالتالي ال system يصير يعمل posing

فبعمل swapout ، swap In

swapout ↓ بشيل ال process اعوجودة بال mem وجودها على ال harddisk (temperary) → حالتها waiting

swap In ↓ بتذكر انه طلعها لبره وبرجع يرجعها

بعمل swapout اذا هي وقفت وهو شايف انها ما رح ترجع قريباً وال mem معبية لحد كبير

ال mide term ممكن ما نحتاجة scheduler أصلاً لانه ال process عددهم قليل وما بوقفوا لفترات طويلة وال mem مساحتها جيدة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* the contex switch figure من process لـ process وبين بكون موجود بشكل سلايد 10 . P0 (إلي وقفت كانت تعمل execution) و P1 إلي وقع عليها الاختيار .

* بالـ contex switch دائما وحدة بتوقف وحدة بتيجي بدالها ، إلي بوقف هو إلي يطلع عن الـ cpu وإلي بيجي بدالة بكون مصدرها من الـ ready

* multitasking : بالتلفونات كم process موجود ؟ أكثر من وحدة وكلهم ready to execut ولكن مابقدر يشغل أكثر من وحدة بنفس الوقت لأنه قدراته مش زي الكاسوب ، بكون عندك وحدة بالـ forground والكثير بالـ background

↓ ready ↓ running

إلي عليها التركيز

Q : if there are many forground ? there is many cpus

الـ iOS بركز بالتشغيل على اشي واحد وبخلي الباقي يستنوا عشان يحافظ على البطارية

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Contex switch : old process stop for some resone and shortterme scheduler pike a new one

∴ اذاً الـ contexswitch هي مهمة الـ shortterm scheduler ، وقت الانتظار لاختيار new process لازم ما يكون طويل لانه بصير wast of time طيب كيف نعمل improve time ؟ من الشكل سلايد 10 بنقلل المناطق السوداء وبنزيد الأسهم

* وقت الاختيار يعتمد على الـ os وبالتحديد على الـ short term scheduler
اذا كان شغال على policy سيئة رح يكون وقت الاختيار طويل

* more complex os لأ يعني الـ short term scheduler شغله معقد عشان مبلغ بطول لحتى يختار بديل بالتالي (time longer)

- أول عامل بأثر على الـ waiting time بالـ contex switch :
① How complex the os is (بالتحديد عمل الـ scheduler)
② HW support (لما تحكي بدك تخزن الـ PCB القديمة وتحمل الجديدة)

- اذا كانت مجودة بالـ hard disk (HD) بالتالي رح تخزن القديمة بالـ HD وتسترجع الجديدة من الـ HD رح تكون العملية بطيئة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بنحسن الموضوع عن طريق انه نصير نخزن ال PCB إما بال memory
او بال cach

③ number of register set : بالاسمبلي كان عنا ال reg من
(Ax ... Dx) لو معندش غيرهم ساعتها ال Process إلي وقفت (P0) كانت
تستخدمهم اذاً لازم اخزنهم عشان أقدر استخدمهم مرة ثانية.

* طيب لو عملنا duplecat (نسخة اضافية) يعني صار عنا مجموعتين من (Ax ... Dx)

(Ax ... Dx) P0 (Ax ... Dx) P1 بالحالة هاي حتى لو صار contex switch
ما في داعي تعمل saving لانه رح ينعملهم
frezing وما رح تستخدمهم أبداً

* طيب لو عندي many reg set : ما في داعي أخزن اشي لانهم
مش shard - بالتالي بتصير عملية ال contex switch
أسرع

LS: Listing → بترجيلك كل الملفات
الموجودة

CD: chang directory cd ← اكتب ال directory
أو cd ... (أي بدك تحول عليها)
بترجعلك Backword

الممسوحة ضوئيا بـ CamScanner

Memo No. الثلاثاء
Date 22 / 7 / 2025

at any time لو حبيت تشوف انت بأي level صرت استخدم pwd
بفرجيك موقعك بالنظام ، لو عطاك (/) فقط معناته انت بال root

* ال contex switch موجود عشان ال cpu ما يتعطل عن العمل وبالتالي
ينخفض ال utilization ، لذلك جزئيه بال os بال Kernal اسمها Schadaler
تبحث عن بديل في حال توقفت ال process وهو عمل ال short term scheduler

- ال contex switching time كله overhead لانه وقت البحث عن بديل
وبالتالي احنا بنتمنى نقلل هذا الوقت قدر الامكان
يا إما عن طريق اختيار algorithem سهلة
أو عمل تحديثات على ال HW زي انك تخزن ال PCB جوا cache أو
تستخدم أكثر من reg set عشان ما نضطر تخزن أشياء جديدة

• How to mange the processes ?

① By use grafical user interface (زي لما تسكر تطبيق وان ابدك ترجعة بتعمل عليه dupal clik)
(in direct way , easy way)

* إي رح نشوفه هو How to use programing to control processes

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* مثلاً ال process creation : عندك الطريقة البديهية انك تعمل ضغطتين عليه وبشتغل ولكن احنا لازم نكتب كود عليه

In programing بدل ما تكبس على [X] عشان تسكر البرنامج بدك تعمل process termenation باستخدام ال programing عن طريق commands معينة

* اي رح نشوفه بالسلايدات اي جاي كيف ندير العمليات من انشاء حذف، ايقاف، استرجاع عن طريق system commands موجودين بمكتبة معينة وانت بتتحكم فيهم عن طريق ال code

* رح نحكي عن scinaros عمكن المبرمج يعملها خلال ال mangment عن طريق الكود ورح نستخدم ال pid (process idintifive) عشان نقدر نميز أي process هو قاعد بركاها المبرمج

• رح نحتاج نجوء ال perant process ورح يعمل منه children ورح يصير عنده many process زي Tree سلايد 22 ورح يصير يشغل البعض ويوقف البعض ويعمل kill للبعض وكل هذا بالبرمجة بدون ال grafical user interface الخاصة بال OS، عشان ترعى كل process بحاجة انك تميزهم واسم ال process هو ال pid

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• على فرض انك أنشأت perant و child عن طريق برنامج ،
شو وضع ال resources (HW) ؟

أول ما حكينا عن ال process حكينا انه ال process لما يعملها ال OS
في مجموعة من ال secondary resources تعطى الها تلقائياً زي مجموعة
من ال reg بتشتغل عليهم هاي ال process عشان تنظم نواتجها
وتعطى mem limits وبتوخذ كمان PC
بالتالي لما يكون عندك process عضاته تلقائياً عندك set of resources
محجوزة اله

- ايش وضع ال resources إلي تكون محجوزة للاب لما يصير عنده ابن؟

① perant and child share all resources ما بحجز مساحة جديدة
بالتالي بوفر مواقع من ال memory ، السيئ انه مارح يكونوا معزولين عن بعض فلما تبرمج
لازم تكون حذر ، ال reg إلي استخدمتهم مع الاب لا تستخدمهم مع الابن
(① ما في عزل ② ما في أمان ③ توفير مساحات)

② perant and child share No resources كل واحد اله
مساحة تخزين خاصة فيه ، معزولين تماماً عن بعض ولكن بتخسر too much
resources

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

③ Child and perant shard subset of resources:

يعني اذا كان محجوز للأب الكثير من المواقع على الذاكرة الابن يكون مشترك معاه فيهم ولكن يكون عنده differant reg set

* بالبرمجة بتقدر تختار أي واحد من الخيارات الثلاث بناسب البيئة عندك (فش خيار أحسن من خيار)

الـ execution opitions :

الأب

الابن

سواء الأب أو الابن هو بالنهاية برنامج وهدفك تنفذه

لو عندك single cpu مارح يزبط الاثنين يكملوا تنفيذ مع بعض لذلك رح نوقف الاب ونخلي الابن يكمل والعكس غير منطقي لانه اذا الاب كمل وانتهى هذا يعني انه كل الـ tasks اعطاه للابن ما عادت مهمة

(الفرعي بعض نواتجه للاب بعدين الاب يكمل تنفيذ)

• لو عندك 2cpus نخليهم يتنفذوا مع بعض في حالة ماني بينهم dependancie

(الخيارين هدول فشو واحد أحسن من الثاني هو يعتمد على المتوفر عندك وطريقة ترتيبك للكود)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• اذا ما في cpu's كافيه رح نوقف واحد ونعشي الثاني أو اذا في dependancie الاب بوقف والابن بكمل تنفيذ

• if the perant fork a child, the perant must wait while the child execute

* هاي الاشياء بتقدر تتحكم فيها من خلال الكود

A Tree of processes in linux سلايد 22

① لاحظ كل process عندها (id) وهذا بساعد الـ OS يحدد بوك action من اي وحدة بالزبط بساعده معلومات أخرى موجودة (PCB)

② كل وحدة من الدوائر موجود بالـ OS ولكن لـ (PCB) اسم آخر (task structure) ومن الـ PCB بتعرف مين الاب ، الابن

③ root of tree : هو الاب إي كل الـ processes اله رقمه pid = 1 .
There is at least single process have number (1)

* بالمناسبة بالغالب الرقم يدل على الـ creation time (وقت الانشاء) الابن رقمه أكبر من الاب

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ال address space (مساحة بال memory) يعني resource (code segment)

• child duplecat of the perant : قصده انه كود الابن نفس كود الاب يعني code segment

• child has single program loud into it • الابن عنده task فرعي مختلف عن الاب لذلك عنده كود مختلف

* بتقدر تعمل 2 options : ① اذا عملت ابن د process بتقدر تخليه يشتغل على كود الاب ويصيروا cooprating

② أو أعطيه secandary task ينفذلك ياه

هذا الحكي ممكن تعمله عن طريق fork command

fork : let a current process creat a child

- إي بدك تتخيله دايمًا لما تيجي تتبع أي كود اذا كان عندك كود C أو C++ في بدايته عندك simple process هاي إي كانت دايمًا تنفذ معنا البرنامج وتمشي step by step.

- كل ما تمر على كلمة fork بتعمل creat a child بالتالي لو بدأ الكود جوا ال main سلايد 24 بكلمة fork معناها let this process creat a child

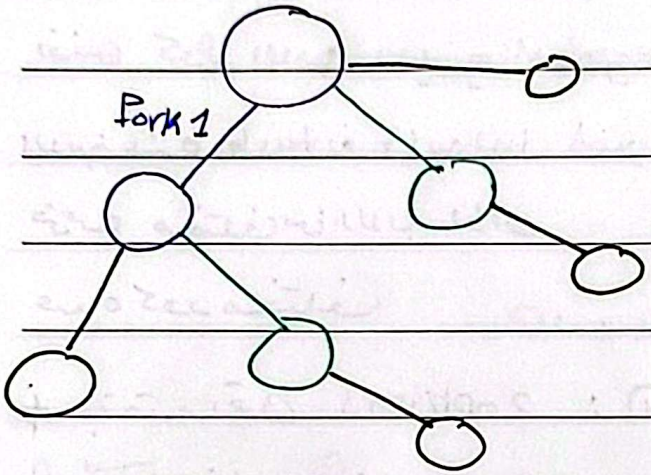
الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

- لما بلشنا البرنامج كان عنا single process مع أول fork صاروا ثنتين والثنين رح يتنفذوا وفرضاً وصلوا لكمان fork اخرى

وهسا كلهم الـ 4 رح يتنفذوا ويوصلوا لـ fork ثالثة

كلهم وكلهم يعملوا child



* لذلك بالعادة كلمة fork مرتبطة بـ condition عشان ما يصير هذا الاشي لانه بالغالب مش كل الـ process بدك تعملهم child

* سلايد 23 الـ fork إي باشكل هاي systemcall لانك لو كاتب برنامج بلغة الـ C وفيه fork مش البرنامج نفسه إي رح يحجز resources

* كلمة fork ايش رح يترتب عليها بالـ HW ؟

① حجز مكان بالـ memory

② PC ③ set of reg

بالتالي بحتاج إلى خدمة الـ OS

بالتالي كلمة fork يترتب عليها: مناداة الـ kernal عشان يعمل ابن عن طريق حجز الـ resources بعدما الـ systemcall يخلص بصير في switch من الـ kernal mode إلى الـ user mode

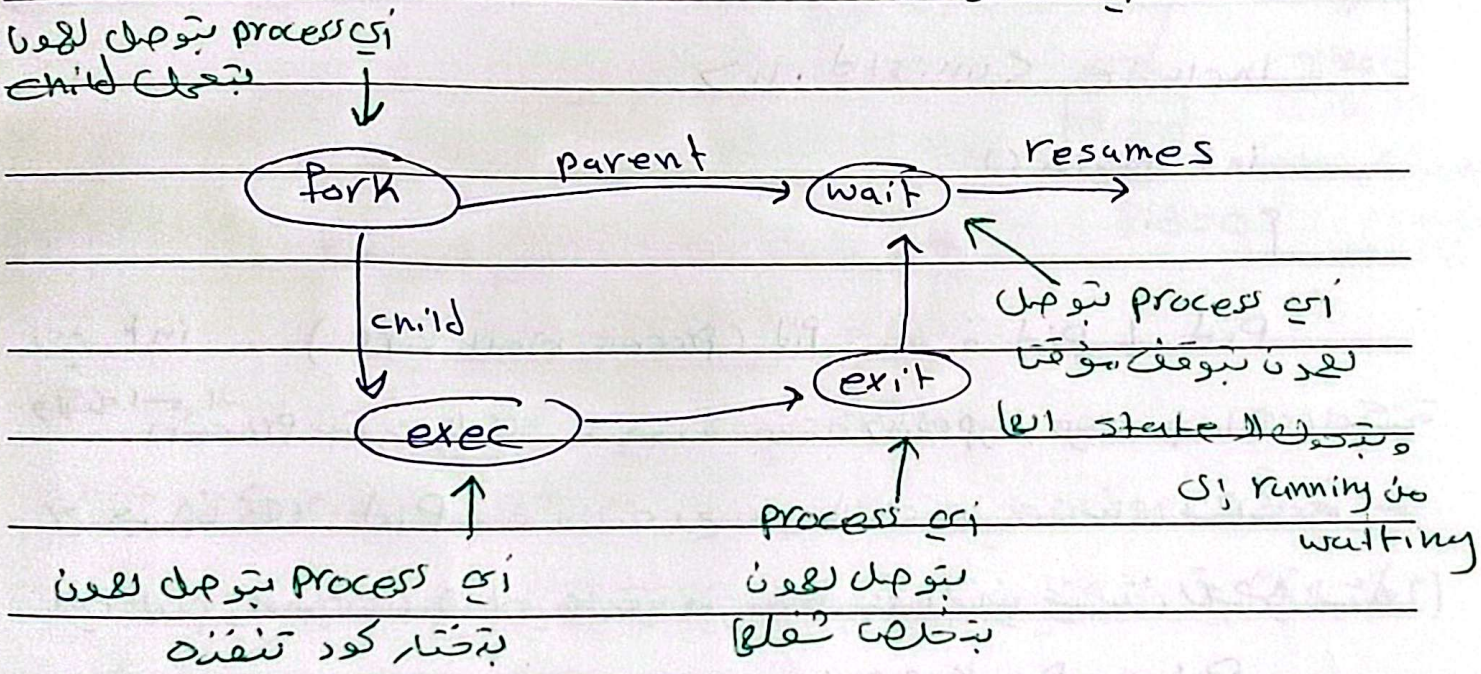
الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

بنفس شكل سلايد 23 : عندك exec() هون بتقدر تختار من خلالها
انه ال child يواصل مسيرة الاب أو تعطيه كود جديد يشتغل عليه

exit() : أي process بتوصل لنهايتها بالكود أخر سطر فيها بتعطي
notification بال exit command

wait() : ال process إي بتوصل هون بتوقف والسبب بتحطه
بين الأقواس



الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

ملاحظة 24 : example على code بتقدر تعمل فيه process & mangment

هذا هو أسهل كود ممكن تشوفه to creat a child وتشوف العلاقة بين

ال child وال perant

وايضا لما بدنا نبلش بكود زي هيك بدك تعمل single process عند نقطة

البداية هي تنشأ عند ال main ، ولكن تخيلها موجودة عند البداية

(أول سطر بالكود)

مكتبات

```
# include <sys/types.h>
# include <stdio.h>
# include <unistd.h>
int main()
{
```

زي int Pid (process identifire) ؛ Pid - t pid

ولكنه اسم ال process مش رقم عادي موجودة بمكتبة <sys/types.h> بدون هاي المكتبة

رح يعترض على Pid ، ولانه رح نعمل child استخدمناها عشان نميز

بين الأب والابن ، (ما في طريقة تميز واحد عن الثاني غير انك تعطيه ال id)

رح تسبب وجود another process ؛ Pid = fork()

بالكود

* رح نمر عند جزئية تتنفذ مرتين بسبب وجود أب وابن

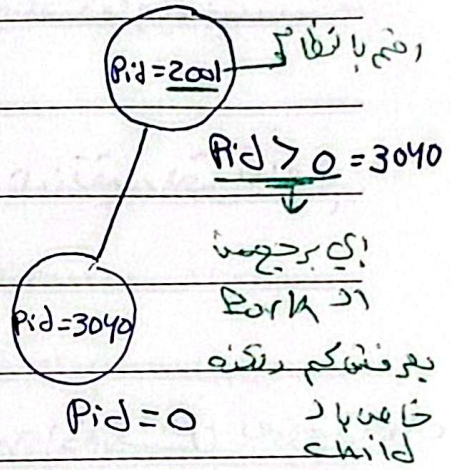
زي كأنك رح تعمل لفتين على البرنامج لانه صار عندك 2 process وذلك

يوضح سبب وجود 2 Pid

الممسوحة ضوئيا بـ CamScanner

الآن المكتبة التي تتيح التمييز بين الاب والابن وإذا ما صارش أب وابن نتيجة error عن طريق التالي:

① الاب دائماً وأبداً الـ pid > 0 . بالغالب بكون الرقم إلي اختاره الـ os للابن ، ركز ... أول ما بلش البرنامج الـ process إلي عملت الـ main كان الـ pid الها فرضاً 2001 مجرد ما عملت fork وصار في child رح يجيب الـ os يعطيها pid عشوائي فرضاً 3040 ، الآن الـ pid إلي بستلمها الاب إلي هي pid>0 بتكون خاصة بالابن والـ pid للابن بتكون صفر



* هون عنا اشيين اسمهم الـ pid

① المميز

② الاسم

- من خلال هذا الاشي بتقدر توجه الاب إلى مكان والابن إلى مكان

الآن لو بدك تحكي للاب اطبع Hi وللابن اطبع hello

وهذا من خلال if pid > 0 اعمل action (هون اطبع Hi)

if pid == 0 اعمل action (هون اطبع hello) هيك بتكون وجهت كل واحد لمسار في الـ execution

* ملاحظة مهمة: رقم الـ process بالنظام 3040 بس إلي برجع من الـ fork هو صفر

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* إذاً عن اثنين اسمهم Pid :

① اسم ال process الفعلي بالنظام (ما بساعدني بال programming) وبكون مخفي زي مثالنا السابق (2001 , 3040)

② أو ال Pid إلي بميز الاب عن الابن ومش لازم يكون اسمة Pid ممكن نخلية (X) فرضاً فيصير الاب X > 0 والابن X = 0 ولكن بالغالب بستخدموا ال Pid ولكأنها صارت identifire

* اذا في أكثر من fork لازم تلاقي اسم ثاني غير ال Pid عشان تقدر تميز بينهم أو ترقم (Pid 2)

* المعلومة إلي بترجع من fork قد تكون negative اذا فشل يعمل child

- اذا كان ال Pid رقم سالب معناته في error

```
if (Pid < 0)
    fprintf(error, "Fork Faild");
    return 1;
```

الاب والابن رح يمروا على ماي الجزئية وبما انه الاب Pid>0 لذلك مارح يكمل قراءة وبما انه الابن Pid=0 برضة مارح يكمل قراءة

هذا الكود بدخل عليه اذا صار في error وال Pid سالب بالتالي بعرضلك رسالة Fork Faild

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

else if (Pid == 0) ← رح يمر على هذا السطر ثنين

الاب Pid>0 والابن Pid=0

فواحد منهم رح ينطبق عليه اشرط

- واضح انه الاب لا ينطبق عليه اشرط

لذلك رح يعمل skip

execlp("/bin/ls", "ls", Null) ;

give another code to child

garpeg name (أعطيه أي اسم) أو حتى اتركه فاضي

لانه ال LS مابيجي بعده اشي

- الابن ينطبق عليه اشرط فرح يكمل قراءة . وظيفة ال child يطبع كل الملفات

* بما انه الابن دخل if بالتالي مارح يفوت else لذلك خلص رح يروح على ال return

else { wait (Null) ; → the perant stop untill the child finsh

بالتالي ال Perant رح يعلق هون ← بالتالي رح يضل الاب واقف هون لحتى الابن يخلص

وبتصير زي Blocking Steatment

كيف الاب بده يعرف انه الابن خلص ؟ لما الابن يخلص ويوصل عند return ال C-enviroment رح تعمل زي exit signal ، واذا كان الاب ينتظر الابن ووصلته هاي ال signal رح يعمل resume (يكمل)

يعني انت مش بحاجة تخبر الاب مجرد تسكيره القوس بالآخر بتعمل signal تخبر فيه الاب ساعتها الاب رح يكمل ويطبع

printf("child complete")

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• أي جزء من الكود بكون الاب والابن مروا عليه ولكن إي ينطبق عليه اشرط كحل وإي لا عمل skip

• لذلك فرضًا لو حطيت جملة غير مشروطة زي (cout << hi) رح تظهر مرتين لانه شافها الاب والابن

• عشان تحسن تنفيذ الكود لازم تعمل double execution مرة تعبر حالك child ومرة perant

• الأفضل دايمًا تبلش بال child بعد بين ال perant

* الكود السابق لو تحاول تنفذه بشتى الطرق رح تلاقي ال "child complete" ظهرت بالآخر وال list of files الموجودين بال bin لازم يظهر قبل لانك عامل Block للاب wait(null) ومجبر الترتيب ضمنيًا بالكود

- مكتبة <stdio.h> لجملة الطباعة printf
- مكتبة <unistd.h> لكلمة fork , wait , execute

زي ما قلنا سابقًا : الاب ينتظر والابن يكمل عن طريق استخدام ال wait أو الاب والابن ينفذوا مع بعض في حال لا يعتمدوا على بعض

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* في حال عدم وجود wait اح يعتمد مبن اسرع يعني ممكن الاب يسبق الابن ويطبع child complete وبعدين يطلع ال output تبع ال child فتكون العبارة إي بال cout statment غير دقيقة

- لهيك بهذا الكود إي خلا الابن يمشي ويكمل والاب ينتظر هوا ال wait غير هيك بصيروا يتسابقوا ومرات لو تعمل run للكود 10 مرات اح تلاقي 8 مرات الاب سبق و 2 الابن سبق (على سبيل المثال)
الاصل ، اعتوقع — حالة شاذة

* المزيد من التفاصيل للاشياء إي بتصير وراء الكواليس بخصوص (process termination)

- لما ال process توصل لنهاية البرنامج عندك خيارين اما بتكتب exit (اختيار 1) أو ما تكتبها : بالكود اسابة مش موجود ولكن لو حلب تحطها ممكن يكون موقعها بعد ال return أو بديلاً عنها

- في لغات البرمجة الحديثة exit غير مهمة لانه ال enviroment نفسها زي (visual studio) أول ما تشوف تسكيره القوس النهائي هي بتتكفل تعمل issue لـ heden exit

- بالتالي استعمال ال exit يكون فعال لو بدك تنهي البرنامج قبل نهايته الطبيعية

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* Run time environment when it reach to the final statment in process it make an issue for exit after exit, the resources are deleted

يعني كأنه المطلوب من ال os يعمل release لكل ال resources

(إمحي كل اشي بخص هاي ال resources)

- إذا ال process إلي خلصت كانت perant : رح نحذف ال resources الخاصة فيها

- إذا ال process إلي خلصت كانت child : check اذا كان في perant بستناه أو لا

② وبتعمل release (رح تبقي ال PCB لحد ما ال perant يخلص اذا كان بستناه)

اذا في perant بستناها رح تعطي signal اله انه بشر بعمله

في عنا system call أخرى اسمها abort والها نفس نتيجة ال exit

ولكنا نستخدمها في مواقع فشل اجراء معين بنعملها issue في ٣ حالات:

① child Try to exceeded allocated resources . الابن حاول ياخذ مساحة أكبر من المعطاه اله والنتيجة خلل (abort)

② tast assigned to child is no longer required . زي لما تفتح صفحة على انت وهو بحمل تضغط [X] بالتالي ال task تبعت ال child انت بطلت مهتم فيها بالتالي abort (مش exit لانك ما دخلت وخلصت شغل عليها)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

(3) The parent is exiting and the OS doesnot allow a child to continue if its parent terminates

(العبارة هاي غير دقيقة ولا تنطبق على كل البرامج)

مش دائماً لما الاب يروح لازم الابن يروح في بعض ال OS واعمم فيهم هو ال unix مابعمل هذا الاشي يعني ممكن تلاقي process الاب تبعها مخلص ورايح من النظام وهي لسا عندها task بتعمله

* ال OS إلي ما بتعمل إلى حكيناه قبل بصير منفي (cascading termination) يعني لنفرض انه ال OS من نوع لا يبقي child اذا الاب مات فهاي ال process اذا وصلت عند ال exit بروح بعمل cascading termination يبدأ من bottom وبصير بعمل kill لل process.

- لو انه unix عادي لو الاب مات الباقي يضل شغال لحتى يخلصوا

- root (ال process الاساسية مارح تموت لذلك ممكن تضم أبناء ال process إلي ماتوا على أنهم أحفادها)

وهاي الحركة بتخلي ال root يشيل على ال actions إلي بتعملها ال child

* ال unix لا يحتوي على cascading termination لانه يمكنك ايجاد child بدون perant

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* مش ضروري انه الاب يكون عنده wait ولكنه الأصح ليش؟ لانه لو
ما في wait رح يصير يعتمد على اسرعة

* الآن عنا Tow naming conventions على حالة الـ process ما بين ابن وأب
مصطلح الـ zombie , orphan لا تنطبق اذا كان عندك single process
أما اذا كان في Tree of process ينطبقوا

① Zombie : يطلق على الابن إي الاب اله خلص وراح وبالتالي أصبح
الأب الجديد اله هو الـ root (process 0) (No parent waiting)

② orphan : (if parent terminated without invoking wait)

* فش standard لـ Browser لانه بعضهم بعملولك كل اشي عن طريق single process
يعني اذا بتفتح 50 صفحة وبتروح على الـ task maneger رح تلاقيهم كلهم تابعين
لنفس الصفحة

* سلايد 28 حاط صورة chrome ، كل صفحة مفتوحة هي process/tab
منفصل عن الاخرين واذا أغلقت وحده ما رح يتأثر الباقي الا إذا أغلقت
الـ root

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Inter process Communication :
↓
Between

يعني هل يلزم يكون في تواصل بين الـ processes
سواء هول الـ processes أب وابن أو أب مع أب آخر
مش شرط يكون عندك كـ Tree ممكن يكون بين process A و process B
ويكون فش بينهم علاقة

* الـ processes بالنظام ممكن يكونوا independant أو cooperating

* ارجع على شكل صفحة 28 هل الصفحات المفتوحة بمحرك البحث
هم cooperating ولا independant : لو افترضت انه كل Tab
process قائمة بحد ذاتها ؟ cooperating ، لانه كلهم موجودين
داخل chrom ولكن تشاركهم مع بعض يكون بـ min level يعني
اشتراكهم يكون بس بالواجهة ، كل واحد عنده resources

independant : يعني في complete isolation بينهم

* انت كمبرمج أي option تختار حسب :
① اذا الكود بسيط و one process كفيلة انها تخلصه بسرعة بتعمل issue
لـ one process وبتخليها تقوم بالمهمة وبالتالي يكون عندك single independant process
② بالمقابل اذا كود معقد وحاب تعمل أب يمسك نص الكود وابن يمسك النص الثاني
هدول صاروا تحت مظلة برنامج واحد وبالتالي صاروا cooperating

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لذلك بالعادة اذا كانت الـ task معقدة بنعمل multipul process
أو multipul threads وبعملوا execution لجزئية من الكود

* اذا استخدمت مبدأ الـ cooperating رح تحقق المنافع التالية:

① information sharing بشتركوا بأغلب الـ resource وجزء
قليل independant فأكنك وفرت resources

② computation speed up ; أي task بتقسمها على ثنين
بتصير أسرع (اذا الـ HW يسمح ، اذا عندك 2cpu's)

③ Modularity : بتقسم الـ complex task على أكثر من process
فبصير الـ debugging تبع العملية أسهل
(تصحيح الاخطاء)

④ convenience : easy to use

* Interprocess communication (IPC) have 2 mechanism:
① shared memory
② message passing

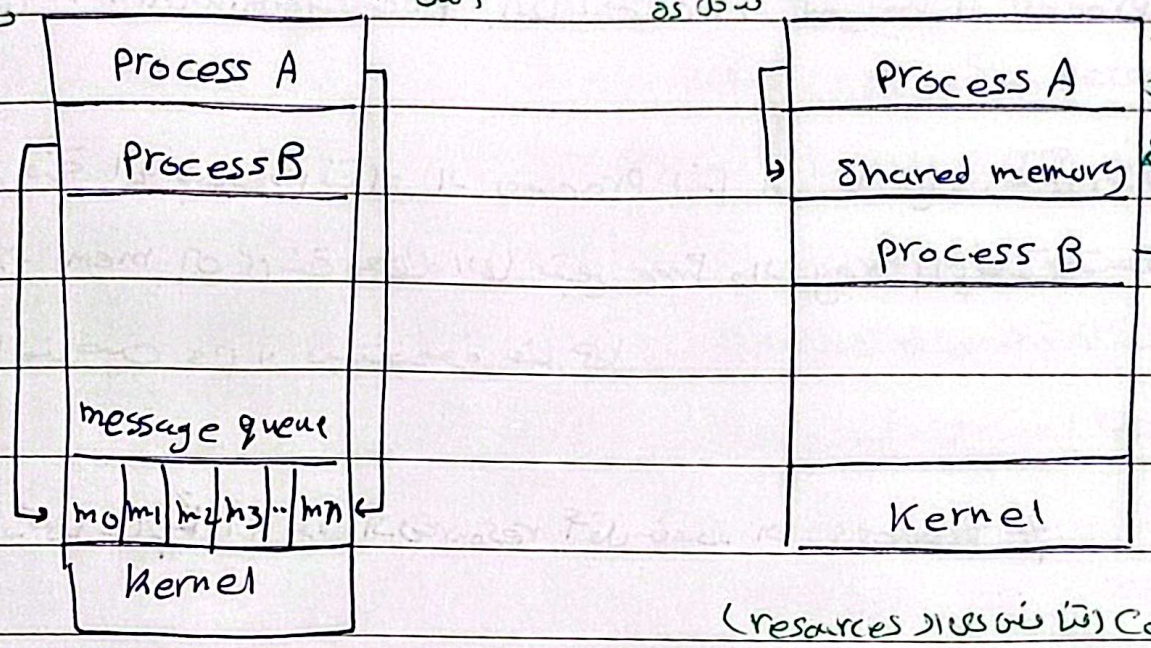
(تابع سلايد 30)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Ⓐ message passing (slow) — آمن / ما في تواصل مباشر ، بطيء لانه Kernel دائماً مشغول

Ⓑ shared memory (fast) — ممكن مشاكل / تواصل مباشر ، أسرع لانه ما في تدخل OS



اذا الكل بحلل كل الصلاحيات هون

Contention (تنافس على الـ resources) في حال كان في كثير لـ process

اذا PA حابة تتواصل مع PB بتعطيها للـ Kernel

الـ Kernel بحطها بالـ massey queue وبتروح PB بتجيب الرسالة من

* لو بدك تعمل تبادل بيانات وقدامك الخيارين إي فوق أي واحد بتختار؟

السؤال لو ضل عملي ما اله اجابة أما اذا قلنا بدنا سرعة بصير فرع B أفضل

لانه ما عليه قيود أو تدخل من الـ OS

* اذاً الفرق بينهم انه فرع Ⓐ الـ OS بتكفل بالارسال والاستقبال (slow)

بالمقابل فرع Ⓑ PA بتكتب على الـ shared memory و PB بياخدها عن هناك

لذلك لا يوجد تدخل للـ OS

* من ناحية الـ scalability (التوسع) يعني لو بدك كذا process يتواصلوا مين أفضل؟

فرع B ممكن يصير ضغط على المساحة المشتركة وخربطه بين الـ process

لذلك الأفضل استخدام فرع Ⓐ

الممسوحة ضوئيا بـ CamScanner

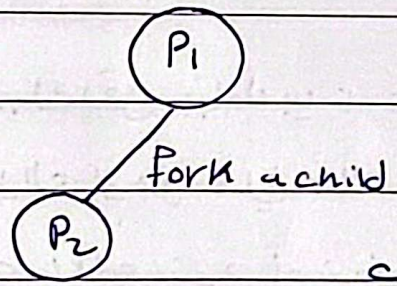
• زي ما قلنا ال process هو برنامج شغال لذلك ال OS لازم يهتم فيه

• حكينا عن ال process termination وانما أخر state بتمر فيها ال process

• حكينا برضة انه بمجرد انتهاء ال process لازم ال OS ياخد منها ال resourses وبالتالي ال mem إلي كانت معطاه لها بتصير free وال reg إلي كانت تستخدمهم برجعوا متاحين عشان يستخدموه غيرها

* زي ما ذكرنا سابقًا انه عدد ال resources أقل من عدد ال process #

- دح نوضح بشكل أكبر الفرق بين ال zombie وال orphan



• لما يصير في termination (انتهاء) للأب ممكن يتبع ذلك cascaded termination (يعمل kill للابناء)

• طيب لو انه عادي الابن يضل شغال بدون وجود أب

هاي حالة ال orphan (the exestance of a child while the perant is terminated)

وبما انه الاب مش موجود بصير يتبع للجد (sys process)

(حالة عادية ولا تمتلك نقاط ضعف)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• في حالة الابن خلص قبل الاب ال OS ما بحذف كل معلومات الابن لكنه يحتفظ ببعض المعلومات الموجودة بال PCB ، لانه أحياناً ال perant ممكن يحتاج معلومات عن ال child (خلص وكالسا اخر output) بتضل هاي المعلومات بال RAM

- المشكلة بتصير انه الاب ما يكون بستنى الابن بالتالي المعلومات إلي خليتهما في حال احتاجها الاب عالفاضي بالتالي :

Zombie : هي process انتهت أبقينا على بعض المعلومات الخاصة فيها في حال الاب احتاجها ولكن الاب ما احتاجها أصلاً

(if No perant is waiting, and the child is terminated But still some information saved in system about it)

(process خلصانه والها معلومات مخزنه هاي zombie)

* هاي نقطة ضعف لانه معلومات الابن المخزنه رح تحسسنا من المساحة لمتى ؟؟ على طول (لانه لو الاب بستنى كان بيوخذ المعلومات وخلص بنمسحها من الmem) لذلك لازم نلاقي حلول :

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

① Timer : أي process بتتحول لل zombie بنحطلها مؤقت اذا خلص
رح نستنتج انه الأب مابحتاج هاي المعلومات ونمسحها

② garbage collector : كل فترة ال OS يشيك على ال zombies الموجودين
بالنظام ، اذا كان ال perant اللهم مش موجودين خلصوا
تماماً ، معناته فش داعي اخلي هاي المعلومات بال RAM
وأخر مساحة

• Interprocess communication (IPC)

- معظم ال process الموجودين بالنظام بكونوا بشتغولوا بطريقة تعاونية
(cooperating) عشان نحقق الاهداف التالية :

① sharing information : مشاركة المعلومات بالتالي الاستفادة من ال resourcess

② speed up : تقسيم complex tasks على أكثر من process بسرع التنفيذ
(if there is cpu's) يدعموا هاي التقسيمة

③ Modularity : شغلك مقسم إلى subtasks كل وحدة معروف مين مسؤول عنها
اين بتعطيك واين متوقع انها تنجز

④ convenience : بالتالي يصبح النظام ككل سهل الاستخدام

- الكلام إلي جاي لو ال process كانوا cooperating وليسوا independant كيف ممكن تخليهم
يتبادلوا المعلومات

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* عنا طريقتين عشان 2 processes يتواصلوا :

(A) Message Passing

B بده يحكي مع A بروح يكتب
هاي المعلومة على جزئية تخص ال OS
(message queue) وال OS
الترتيب مهم
بوصلهم لـ A
ليش بحتاج queue ؟ (1) لانه ال OS
زي ال cpu مش فاضي وبالتالي عشان
ما تضيع ال massege
(2) لانه ال B ممكن كل شوي يبعث
رسالة لـ A وما ينتظر عملية ال delivery
وبالتالي عشان الرسائل ما تضيع عنا هاي
ال queue
- عمليات الارسال والتواصل تتم برعاية
ال OS لذلك A, B ما الهم دخل
كيف بتصير العملية لذلك :
synchronized / slow opration
صحيحة 100%
أكثر scalable
more secure

(B) Shard memory

لا يتم برعاية ال OS الاشي
الوحيد إلي بعمله ال OS
هو انه يحجز ال shared memory
(لانه ما بنفع إلي بحجز اي مكان
يكون process)
إلي رح يبعث الرسالة رح يكتبها
على ال shard area وإلي رح يقرأها
رح يقرأها من هناك برضه
(يتم برعاية ال process)
- fast لانه لا ينتظر ال OS
- بغلب بالبرمجة ولذلك هي
عرضه للخطأ
(قد تكون الصحة 100% ولكن غير مضمونة)
not scalable
less secure (أي حدا بشوف اشي بتكتب)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Scalability (قابلية التوسع) : كم بقدر أفعل users على النظام

نظام A ← scalabil أكثر لانه كل الـ tasks تتم برعاية الـ OS

بينما نظام B ← not scalabil لانه لو بدك تضيف process ثالثة بصير بدك

تزيد المساحة (shared area) وبالتالي يصبح العبء أكبر على المبرمج

to enforce more synchronization between processes

• تفاصيل أكثر عن كيف نعمل shared memory (SM) أو massage passing (MP)

داخل الـ OS

① Producer - Consumer Problem سلايد 32

(Sender) (reciver)

* اسم العملية لا يوحي عمل هو (SM) أو (MP) وحتى الكلام إلي رح نحكيه منطبق

على الحالتين

1) نحكي عن مساحات التخزين (الانتظار) في الحالتين :-

① unbounded Buffer : فش حد أعلى لاستيعاب الـ Buffer

طبعاً هاي الفرضية غير منطقية لانه أكيد في العمانعماية حتى

لو كانت على الـ harddisk ، هي لغايات الذكر ولكنه

لا يمكن تطبيقها على الحاسوب

- دايماً لما بدك تعمل Buffer سواء برعاية الـ OS أو mange By process في

إلعما حد أقصى بالمساحة

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

• طيب ليش بنحكي انه الـ Buffer الواحد ناقص ؟

① اذا فلت الـ producer لازم يوقف
(امرسل)

② اذا فضيت الـ consumer لازم يوقف
(استقبل)

— بالتالي الـ producer-consumer لازم يراعوا 2-events لحتى تكون
فاضية الـ consumer لازم ما يستقبل (لانه فش اشي يوخذه)
وحتى تكون مليانة الـ producer لازم ما يبعث

— رح يبرمجيا 3-codes
① setup Buffer
② code of producer
③ code of consumer

① لحتى نعمل setup Buffer لازم نحدد ٣ حالات :

Ⓐ اعادة اي بدهم يكتبوا ويقرأوا منها

Ⓑ 2-pointers عشان track reading and writing

* رح نعمل implementation لـ Buffer اي هو array حتى لو كانت
الـ input هي بترجع array

الممسوحة ضوئيا بـ CamScanner

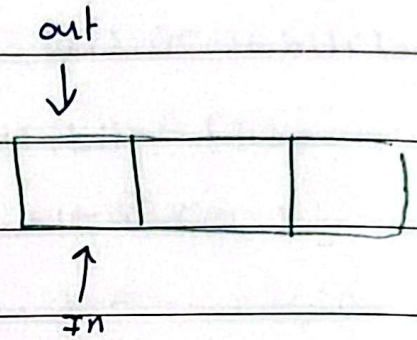
Memo No.
Date / /

item buffer [BUFFER size] ;

array

حجمها

افرضه ٣ مواقع



int in = 0 ; ← مؤشر تعلك وين تضيف

int out = 0 ;

مؤشر بعملك traking من وين بتوخذ فهذا بعمم ال consumer

لذلك ال producer معتم فيه

(In) بكون على أول موقع لما تكون فاضية

* بالتالي ال Buffer صارت جاهزة ، مكان تخزيني بغض النظر ايش هو (array ، linked list ، ...) بتكون من عدد من المواقع و 2-pointers

In : تعلك وين تضيف بالتالي لازم يأشر على مكان فاضي

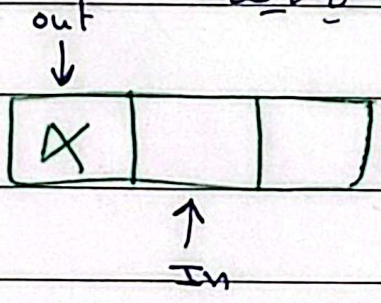
out : تعلك من وين تاخذ بالتالي بأشر على مكان مليان

* اذا كان In = out بكون ال Buffer فاضية

ال producer ما عنده عائق انه يشتغل بس ال consumer ما في اشي ياخده

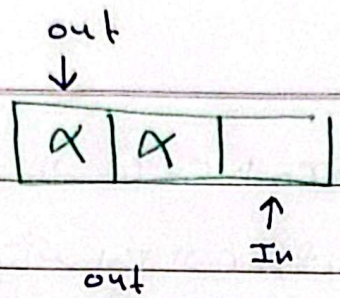
لو ضفت عنصر ايش رح يصير على ال pointers

In رح يتحرك لأول موقع فاضي

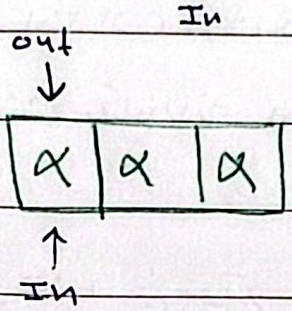


الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /



* لو ضفنا كمان عنصر



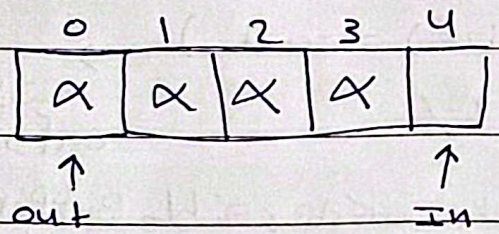
* لو ضفنا كمان عنصر

تحرك الـ In

للموقع الأول كمان مرة لانه circular implementation (آخر موقع يليه مباشرة أول موقع عشان نعمل reuse للمواقع)

* عملية الاضافة الاخيرة خاطئة لماذا؟ لانه صار In=out واحنا قلنا اذا الـ In=out المفروض يكون فاضي ولكن لاحظ فوق هو مليان ، ايش الحل طيب؟؟

• ضيف لحد (max-1) ، يعني اذا الـ Buffer فيها ٤ مواقع بتقدر تضيف لحد ٣ واذا فيها ٥ مواقع ضيف لحد ٤



وفي الحالة هاي ممنوع تضيف بدون ما تكون أخذت منها قبل

لانه اذا ضفت مباشرة رح تحصل على حالة out=In وهي خطأ بتعطيك miss leading على الـ condition إي قاعد بصير

* In=out فاضية

In=out-1 مليانة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

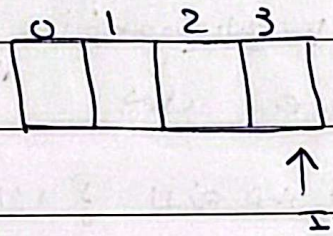
* الآن لو بدنا نكتب كود ال (producer) رح نكتب كود genaric

① ال producer لازم يتأكد اذا ال Buffer مليانه أو لا اذا مليانه

بعمل Block بستنى فش مكان لانه اذا فاضيه بضيف وبتحرك

ال In لأول موقع فاضي

* عشان نضمن انه In لما يوصل لاخر موقع يرجع لأول موقع رح نستعمل (mod)



رح نضيف item اسمه (next produced)

4%4 = 0
برجع لموقع صفر

* كود genaric يعني غير مخصص لأي داتا تايب (data type)

while ((in+1) % Buffer-size) == out) → اذا حركت ال In ولقيته صار عند ال out

/* do nothing */

يعني ال Buffer مليانه، هاي كانها Block طول ما هي مليانه ممنوع تضيف

→ while (شرط) { } هاي الجملة تعادل ؟

اذا تحقق الشرط رح يدخلك ب loop فاضيه لحتى

الشرط يبطل صحيح وتقدر تنزل للجزء التالي

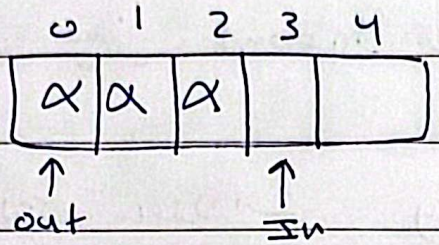
يعني (In+1) ≠ out

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

0	1	2	3	4
α	α	α		

↑ out ↑ In



$(In+1) \% 5$

$(3+1) \% 5$

$4 \% 5 = 4$

الشرط $4 \neq 0$

لم يتحقق لذلك رح نروح على الجزء إلي بضيف

* بما انه In بأشر على مكان فاضي لذلك رح نضيف بالأول بعدين نحرك اغوشر

buffer [in] = next_produced ؛

الموقع الفاضي — الاشي إلي بدي أضيفه

in = (in+1) % BuffER_size

بحرك اغوشر لأول موقع فاضي — عشان اضمن انه العملية circular

* هذا بالنسبة لكود ال producer ، بشيك بالأول اذا فاضي بعدين بضيف

* أما كود ال consumer رح نشيك اذا فاضيه رح نعمل Block واذا مليانه رح ناخد ال item وبعدين نحرك ال (out)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• الفاضي يعني in=out وهذا بالعادة أول ما أنشئ ال Buffer

بالتالي ما عندك اشي empty while (in==out)

لـ consumer تأخذه

بالتالي بتفوت بـ loop أو Blocking statment لحتى يبطل In=out

* طيب في حال In≠out رح تنتقل لي بعده

لأنه There is something in the buffer the consumer can take it.

next_consumed = buffer [out];

المكان إلي بدك تأخذ العنصر عليه

موقع العنصر إلي رح تأخذه

out = (out +1) % BUFFER_SIZE;

بأشر على أول موقع مليان

عشان نشتغل circular

• الآن بعد تتبع الكودات هل هم Shard memory ولا massage passing ؟

Shard memory واسمها بالكود هو Buffer وانت كمبرمج

بتشوف الفاضي واعليان منعها وكل ال actions انت عاملها

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

واذا ما ميزت بين ال empty وال full ال Buffer عندك بصير لها overwriting

أو ممكن (undefinde Behavior) اذا بتعمل consuming empty buffer

- This bounded Buffer is always faster than massege-passing, But the problem is a complexity on the Programmer, and it less secure because the Buffer is available for evergone.

* بتشبه push , pop كانها stack ال (shard memory)

• عمليات الادخال والاخراج على ال shard memory من مهام مين ؟؟

production ← ال producer process
Consumption ← ال consumer process

العبء على المبرمج إلي كتبهم

• We have major issues :

① When it full to Block the producer

② When it empty to Block consumer

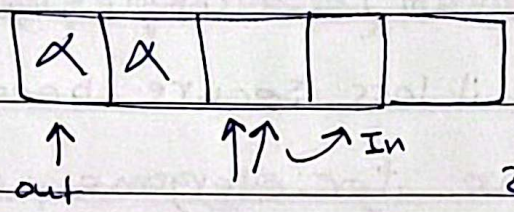
③ How to move the Pointers In/out

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• We have a synchronization problem when shared memory used by more than tow processes.

تخيل عندك 2 producer بدهم يحطوا بنفس الموقع وبنفس اللحظة



2 رح يكتبوا فوق بعض

وال Pointer رح يتحرك مرة وحدة

بالتالي ظهر أثر واحد بس مش كأنه 2 شغالين

* أول ما تشوف الكود اذا لقيت send , receive (system call)

أعرف انه massey passing عن طريق ال OS

send : معناها انه ال producer بطلب من ال OS يعمل ال send

إلي بطلب ال request = producer

إلي بنفذ الطلب = OS

البرنامج الثاني رح يعمل receive وبرضه عن طريق ال OS

• The message size is either fixed or variable

رح نفترض انه كل الرسائل

الها نفس الحجم للسهولة ولكن حتى لو كانت

ال Buffer دايما نوعها زي نوع ال item متغيرة العدد رح يكون

على ال OS

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* mechanism for processes to communicate and to synchronize their actions, done by operating system

هيك صار بحضن ال message passing ← التعريف لحد هون ينطبق أيضاً على ال shard memory

* message system - process communicate with each other without resorting to shared variables. هذا الحكي خطأ

• تقبلك انه ال message passing بحتاجش shared area :)

- طيب ال memory queue مش ببعث عليها وباخذ منها بالتالي هاي shared area بالتالي هاي الجملة لو احتبك بالامتحان بتحطها خطأ

* هسا قصده انه ال shared area بما انها برعاية ال os ومحجوبة عن ال processes انها بطلت shared (لانك ما بتقدر تستخدمها بشكل مباشر انت بتعطي رسالتك لل Kernel وهو بحطها هناك) وبالتالي ولكانه فش shard area (There is no shard area accessible from process)

- لما نحكيلك انه ما في shard area بكون على أرض الواقع في بس مش مسموحلك تستخدمها لانه ال os شغال عليها بالتالي هاي (message passing)

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

• ليش ال message passing بطيء ؟ لو A بده يبعث لـ B ؟

A → Kernel Kernel → B

برضه لما يفضي يعمل لما يفضي و يصير قادر

send يعمل receiving

* Criteria of message passing :

Physical	Logical
مواصفات المنطقة المشتركة (message queue) ممكن تكون موجودة بـ ١	كيف تعمل send و receive
① Shard area : موجودة بال RAM ومش متاحة لـ process متاحة لـ os	① Direct or indirect ② Synchronous or Asynchronous ③ Automatic or explicit buffering
② Hardware Bus : ناقل بين ال process ، اذا بتضمن انه ال os ينقل الداتا on the fly ما في داعي تعمل issue لـ shard memory	- اح نشرح كل وحدة منهم -
③ network : الشرط انه ال process A و B يكونوا على أجهزة مختلفة متصلة عن طريق الانترنت	

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

Message passing

① Direct Communication : إي بخلي عملية الارسال

والاستقبال direct وجود اسم أسرع ، متزامن (syncr...)

المستلم في ال system call

send (P, message) الرسالة طلب من ال OS يودي الرسالة لمستقبل

send (1234, m1) P (بالعادة اسم ال process بكون رقم)

receive (Q, message) طلب من ال OS انه يستلم الرسالة

القادمة من ال Q (المستلم)

② Indirect communication : أبطرء ،

لاحظ : ابعث لـ A واستلم من A

Send (A, message) اذاً A كأنها صارت نقطة استلام

receive (A, message) وتسليم زي كأنها mail box (صندوق بريد)

↳ Besy , slow , Shared

إذا ال OS هدفه يسرع على ال process بستخدم ال direct ، اذا هو شايف

انه ال process ما بتفرق معلم اسرعة بخليه indirect

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لو عندك 3-process و single mailbox إذا P_1 بعث

P_2 , P_3 ممكن أي واحد يستلم فعمون P_1 كان قصده يبعث لـ P_2 و P_3

فعمون المشكلة انه الـ real receiver unspecified عنا مجموعة حلول

بما انه في مرسل واحد وعدة مستقبلين :

① خلي الـ link على الأكثر بين اثنين

② الـ OS بخلي process وحدة تستلم

وبوقف الثانية

③ خلي الاسرع يستلم من المرسل

Process communication

- message passing
  - direct
    - محصور بين مرسل ومستقبل واحد
  - indirect
    - Scalable (يدعم عدد أكبر من المستخدمين)
- Shared memory

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بال message passing اذا ال processes بستنوا بعض هذا اسمه Blocking
يعني اذا ال sender لازم يستنى ال message توصل لل receiver
وقاعد بعملش اشي هذا Blocking ، ليش هذا الاشي بصير ؟
لانه لو ال buffer مليانه لو انت ما استنيت ح تضيع الرسالة

* ال sender, receiver بقلبوا على وضع ال Block اذا ال Buffer وضعه
يلزمه ذلك وال Blocking بخليهم synchronance يعني قبل ما يعملوا
action بتاكدوا من بعض

* انت ك sender متى ممكن ما تستنى ؟ اذا رسالتك راحت على ال buffer
بأمان وانت لو كملت ما بتضيع الداتا ساعتها بكون non-Blocking

- بالتالي ال OS لما A و B بتواصلوا بقلب مرات Blocking ومرات
non-Blocking حسب وضع ال Buffer

* فش واحد أحسن من التاني

* اذا اعرسلوا واستقبل في وضع ال Blocking بصير اسم الحالة
randezvous أو syncronance comunicating

الممسوحة ضوئيا بـ CamScanner

Memo No. الاثنين
Date 28 / 7 / 2025

gcc, cc command do compilation and execution for c or c++ files

① Fork system call

بعمل لل process إلي عندك ابن وبالتالي بتضاعف عدد ال process

② execlp system call

من خلالها بتقدر تخلي اهتمام ال child يروح لبرنامج أخر، لانه بطبيعة الحال لما تعمل child لازم تسلمه مهمة أو task By default، ال يكون شغال على كود الاب اذا استخدمت execlp تكون أعطيته مهمة فرعية يعملها

* عشان نقسم البرنامج ونخلي الاب يروح على جهة والابن يروح على جهة أخرى رح نستخدم ال Pid إلي راجع عن ال Fork

* بداية أي كود رح يكون في single process بالتالي تنفيذ الاسطر يكون لمرة واحدة أما بعد ال Fork رح يكون التنفيذ مرتين مرة للاب ومرة للابن

* getpid (system call) : بجيب ال id تبع ال process بالنظام (ما بكون بعرفه لانه يعطى بشكل عشوائي لما ال process تنشأ

* فأنت أول ما تشغل البرنامج عن طريق ال gcc تنشأ عندك process وتعطى resources والها PCB وتعطى رقم يمثل الاسم (عشوائي) بقدرش أحزركم ولكن بقدر أسترجعه عن طريق getpid

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

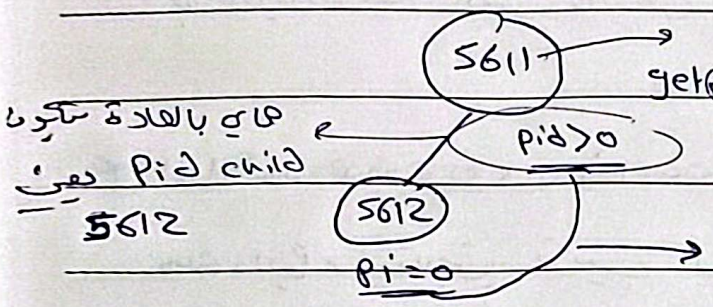
* قلنا بنميز الأب عن الابن عن طريق انه pid للابن = 0

وال pid perant > 0 هدول ما الهم دخل بال getpid

5611 → عشان أجيب هذا بستخدم getpid (system)

pid>0 → هاي بالعادة بتكون pid child يعني 5612

5612



pi=0 → pid بساعد فقط في توجيه الكود

لـ part segment أو child segment

الابن والاب بمروا على كل اشي تحت ال fork وإلي بنطبق عليه اشرط

لكمل وإلي ما بنطبق عليه بعمل skip وبروح لاشي تاني

* ال ls ما بتوخذ parameters أما ال cd مثلاً بحط بعدها

اسم اعكان إلي بدي أروح عليه أو (...) في حال بدي أرجع لورى

mkdir بحط بعده اسم اعلف إلي بدي أنشئه

- زي ما قلنا ال child إلي ال perant ما بستناه اسمه (زومبي)

واسمه هيك لانه قلنا انه ال child بخلص وهو ما بعرف انه الأب

ما بستناه لذلك رح يخزن جزء من ال PCB لانه باعتقادة

انه الاب رح يحتاجها ولكن في الواقع الاب خلص وما بحتاجها

رح نخسر مساحة تخزين اذا ما حلينا اعوضع.

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ممكن نحط بالكود زي wait ، لما ال child يخلص بعطي signal
لل wait وبشون انه مش لازم يعمل passing لل information
بالتالي ال OS بلغي ما تبقى من معلومات ال child وهيك بنحل الزومبي

* لما الكود لا يحتوي على (wait) الاب ينتظر الابن لحتى يخلص
مش دايماً رح تلاقي ترتيب ال output زي ما هو رح يصير ال output
يعتمد مين أسرع

- pwd يطلع ال active directory
- sudo apt بتعمل insallation لمكتبات مش موجودة عندك

* مواصفات ال massege passing : بقلك صح انه الطريقة
message passing الا انك بتحتاج تخزن هاي الرسائل لحد ما تروح من ال
sender إلى reciver وهذا التخزين اعوقت ممكن يصير على
shared memory (ولكنها مختلفة عن ال shared memory إلي حكينا عنها قبل)
هاي ال shared memory هي حكراً على ال OS وهو بيستخدمها مؤقتاً لايصال
الرسائل وهي not accessable from users وبعد ما يوصل الرسالة بلغي
استخدام ال shard memory

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* حكينا عن احتمالات انها تتخزن في ال HW Bus أو بال network
أو ممكن باستخدام ال pipe
هاي كلها طرق عشان ما نضيع الرسالة إلي رح تنتقل من مكان لمكان

* حكينا عن خواص اخرى لل massage passing هل هو direct أو indirect

direct	Indirect
• لما المرسل يعمل send بحدد ال process id إلي بده يبعثله • لما المستقبل يعمل recive بحط ال process id لل sender	• بحطوا الرسالة باشي زي ال mailbox عشان لاحقاً المستقبل ياخدها عن هناك

Syncronanuse	Asynchronanus
• المرسل بستنى توصله الرسالة وبالتالي بصير Blocking (زي كلمة wait)	• المرسل ببعث وبكمل شغله وبالتالي non Blocking

← ملحوظة دخل بال message passing
ولكن فقط لتوضيح الفكرة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* اذا كان ال sender وال receiver بوضع ال Blocking بصير اسم الموضوع randevo

• أخر اشي بال message passing ايش وضع ال Buffer عندها حل هي Automatic ولا explicit

يعني ال sender وال receiver ما بعملوا أي اشي عشان يتأكدوا اذا ال Buffer مليانة أو فاضية

أما هون ال Sender لما بده يبعت بده يتأكد حل ال Buffer مليانة ولا لا زي الكودات إلي شفناهم سابقاً يعني كأنهم shared memory

- اذا الكود انت قاعد بتكتبه لازم يكون التأكد explicit
- اذا انت راكن على ال OS يعني بتعمل (message passing) بتكون عملية التأكد من المساحات والتوقف اذا المساحات مليانة ر2 تكون Automatic وبرعاية ال OS

* العملي = ال complex في ال shared memory بتصير أسهل في ال message passing لانه ال OS تتكفل بأي complexity

الممسوحة ضوئيا بـ CamScanner

Memo No.

Date / /

كود سلايد (45)

while(true) دائماً ال sender بكرر نفس طبيعة الاجراء

لاحظ بالكود اعدل بعدها ماني (؟) لانه لو كان في ح يصير معناها

while with empty body ولكن هون while(true) وبعدها اقواس مفتوحة

يعني في اجراء ح يصير اذا اشرط كان صحيح

ال while إلي بعدها فاصلة منقوطة هدفها انه البرنامج يعلق

فيها لحد ما يتبعي اشرط

* أما هون while(true) ح تكون جملة دائماً صحيحة فبعمل sending

والمقصود انه اعرسل ح يكرر نفس طبيعة الاجراء انه دايماً ح يعمل

send بدون ما يعمله وضع ال buffer لانه هاي صارت من معمة ال os

* ال send إلي بالكود هل هي Blocking و nonBlocking ؟ بقدرش أعرف

* هاي ال send هي direct و indirect ؟ indirect لانه

اسم المستقبل مش موجود وبالتالي هاي الرسالة رايحة و mailbox

* ضرب ال consumer كل إلي عليه يستقبل بدون ما يتأكد اذا في

رسائل بال Buffer أو لا

* لنفرض انه ال Buffer فاضية هل ح تكون ال receive ← Blocking أو non Blocking

— مش مبين

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لما نحكي Buffering ممكن نكون بنحكي عن message passing / shared memory

① ما بنفع يكون حجمها unbounded (ملحوش حد أعلى) لانه بالنهاية الحجم مكان بال mem او بال harddisk واله limit

② ما بنفع تكون zero capacity لانه وقتها بتكون عديمة الفائدة وبالتالي لازم المرسل والمستقبل يكونوا Blocking لانه اذا ما وقفوا يستنوا ال message رح تروح (تضيع)

③ bounded capacity يعني عندك upper limit (variable size)

* رح نشوف كود نعرف فيه كيف How to program a shared memory باستخدام pozex stanterd فيه مجموعة من المكتبات إلي بتيحلك تعمل تجارب للأشياء النظرية إلي بتقرأها بال OS
زي ال fork , wait بالتالي جوه هذا ال Standerd مجموعة من ال systemcalls : shm-open , ftruncate
shm-map (mmap)

* هذا ال Standerd وظيفته الاساسية تجرب command unix على environment windows

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

كود سلايد 48 : بعمة منه ٣ شغلات أساسيات وهنه ابي بعملوا sharedmemory

① shm_fd = shm.open(name , O-CREAT ① O-RDWR , 0666) ;

shm → shared memory
name → اسمها ومش مهم
اسمها هو اشي اسمه discreptor
O-CREAT → create
O-RDWR → read, write
① → concatenation

هدول مع بعض بصير معنا هم :

اذا هاي الـ sharedmemory أصلاً موجودة أعملها open واذا مش موجودة أعملها creat (O-CREAT) والـ process ايش بقدروا يعملوا على هاي الـ sharedmemory ؟ Write/ Read (O-RDWR) بتقدر تبدلها بـ (O-RD) ولكن بتصير الـ mem لقراءة فقط بالتالي بصير مشكلة عند الـ producer لانه ببطل قادر يكتب عليها لذلك لازم الـ command يكون (O-RDWR)

owner
لباقي الـ process
octal ← 0666
110 110 110
RWE

read بتقدر تقرأ لانه قيمة الـ R-Flage = 1
write بتقدر تكتب لانه قيمة الـ W-Flage = 1
executie ما بتقدر تنفذ لانه قيمة E-Flage = 0

ليش مكررة ٣ مرات ؟ أول (6) بخص الـ owener of the shard buffer ، ثاني (6) نفس الصلاحيات ولكنها لـ other group بشبهوا الـ owner يعني اذا الـ owner هو user process اح يكون أي user process عندها نفس الصلاحيات =

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لو فرضاً بدلنا ال 666 بـ 640

owner
1 1 0
R W E

100
R W E
بقدر يعمل
بس reading
(شبه ال owner)

000
R W E
ما الهم صلاحيات
ما بقدروا يعملوا
اشي
(other process)

* ال Execute (E) دايماً (0)

* الآن بنهاية سطر الكود إلي بالصفحة السابقة إلي حصل فعلياً : البرنامج
إلي نفذ هاي الجملة طلب من ال OS يدور له على مكان بال mem لحتى
يكون shared mem وبالاسم إلي انت اخترته (لو ترجع بالكود سلايد 48 رح
تلاقي فوق const char *name = "OS" وهذا اسم ال shared حد OS)
والصلاحيات إلي عليه (Read, write) اذا ما كان موجود اعمله creat
اذا موجود اعمله open وصلاحيات ال process كلهم R, W فقط
- لحد الآن لا يوجد أي شيء مكتوب على ال shared memory
- الآن في جزئية ناقصة، لو طلبت من ال OS يعملك ال shared memory بعمق التفاصيل
تكون لسا ناقصة معلومة اشي هي ؟ size لذلك رح نشوف ال
command التالي .

② ftruncate (shm-fd, size)
shared memory discreptor

فبصير هذا السطر معناه انه طلبنا منك
سابقاً تحمل تعمل shard memory الآن خلي حجمها مساوي لـ size لو رجعت
بالكود فوق رح تلاقي (const int size = 4096 Byte)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بالتالي رح يروح يدور على 4096 Byte متتالية بال memory بيسميلك
(يا هم (OS) وهاي هي اعادة إلي بتقدر تقرأ وتكتب عليها

* ال shared memory discreptor الموجود بالسطر السابق رح يكون نفس قيمة

السطر الموجود فوق لو دورت بالكود رح تلاقي (int shm-fd)
(هو وصف لل shared memory) في ال operating system -
- لحد الان لسا ما كتبنا اشي على ال shared memory
- آخر خطوة لحتى تكون ال shared memory جاهزة

(3) Ptr = mmap (0, size, PROT_WRITE, MAP_SHARED, shm-fd, 0)

هو إلي بنسخ ال shared memory وبخليها جزء من ال producer وال consumer
(لو نسيت تكتب ال mmap تبطل قادر تكتب أو تقرأ من ال shared memory)

هون حالياً هاي تابعة لل producer فقط ولازم ال consumer يعمل زيها عشان يقدر يقرأ من ال shared memory

(لا عطائك صلاحيات) mmap القراءة أو الكتابة

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

• الآن عشان نكتب رح نستخدم SprintF(ptr, "%s", message0)

string

printf تشبه ولكنها ما بتطبع على الـ tairmenal (الشاشة)

هاي بتعطي buffer لـ أدموش

لو ترجع بالكود رح تلاقي (void *ptr) وهاي معناها genaric يعني yo can write any data type on the shared memory

* السطر هذا معناه أكتب message0 وإلي الها format انها string على الـ pointer (ptr)

* ايش يعني نكتب message على pointer ؟ يعني انه الـ pointer ptr بأشر للـ message0

* فلو حكيت cout*ptr رح يطبع الرسالة

• للدلالة على انه مساحة الـ sharedmen قلت لانه كتبنا عليها رح نستخدم التالي :

ptr += strlen (message0)

length of the message

يعني زود الـ pointer وحركة والهدف من هاي العملية (الـ pointer إلي كان بأشر على بداية الـ sharedmen وبعد عملية الكتابة تغير مكانه) : تحريك اعدوش لمكان فاضي عشان تقوم بالادخال عنده

* لو ما حركت اعدوش أي رسالة جديده رح تنكتب فوق القديمة أما لما تحرك اعدوش بصير وكأنه pointing to next free location

* هذا الحكي كله ضمن نظام 4096 Byte المحجوزين وبضل اعدوش بتحرك لحتى يخلصوا الـ 4096B

الممسوحة ضوئيا بـ CamScanner

Memo No. الثلاثاء
Date 29 / 7 / 2025

- كود سلايد 49 : لاحظ في عندك أول الكود 5 مكتبات لانه بعض الجمل بالكود بتحتاج هاي المكتبات.

- لما نحكي shared memory (SM) يعني مكان في ثنين بعملوا عليه accesse واحد يضع الداتا والثاني يقرأ الداتا

- كود سلايد 48 هو كود ال producer يعني رح يحط داتا داخل ال (SM)

* لما نشوف int shm-fd هذا عبارة عن discreptor وهو اسم ال (shm) داخل ال process نفسها

* اما "05" const char *name هذا اسم ال shm بين ال process

* فإذا كان process (A) و process (B) بدهم يتشاركوا ال shared memory بعمهم اشي اسمها

* إذا ال process A بدها تعمل عليها writing or Reading بعمها ال discreptor (هو ال id إلي بميز ال shared memory)

* إذاً ال shm هو ال discreptor وهو مميز ال shared memory داخل ال process بس هذا المميز ما هو نوعه (هون int) بالتالي اتم بعدد أي shared memory بدلا تتعامل معها

الممسوحة ضوئيا بـ CamScanner

Memo No. _____

Date / /

(producer code)

حكينا عن Shm-fd = shmopen(name, O-CREAT | O-RDRW, 666)

بالكود تبلاقية (OS)

اشي بعمل ال shared memory

owner of shared memory can read and write from this number (OS, رقم يمثل اعالك)

other process have the same type of the owner

create هون يعني انت مش متأكد انها موجودة ، فاذا مش موجودة اعملها

creat واذا موجودة اعملها open

واذا انت ضامن انه ال shared memory موجودة

بتقدر تكتفي بـ O-RDRW وبتعملها open

عن طريق systemcall

ما تبقى عن ال process بال system

* ftruncate(shm-fd, size)

هاي معناها انه ال discreptor هذا اللي بأشر على ال shared memory

أوصله عشان تحدد size تبع ال shared memory

- بما انه بدنا نروح على ال mem الكبيرة اللي هي ال RAM ونحجز منها

ال shared memory اللي اسمها ال OS وعليها shm-fd discreptor وبحجم

4096 كل هاي الاجراءات بعملها ال (OS)

فانت بتطلب منه هاي الخدمات وبتعمل switch لل Kernel mode عشان يعملك

يا هم

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* الآن هاي ال shared memory إي حجمها 4096 موجودة بمكان ما
في ال (memory) ال producer رح يكون في مكان أخر لذلك
عشان نعطيه صلاحية الكتابة على ال shared memory رح نستخدم
(mmap) يعني اربط ال shared memory مع ال process الحالية

Ptr = mmap (0, size, PROT_WRITE, MAP_SHARED, shm_fd, 0)

- 0 ← خلي ال OS ينقي أي مكان لل shared memory (يحطها وين ما بده) داخل RAM
- size ← 4096
- PROT_WRITE ← Process operation write (لأنه كود Producer) Read ممكن لو كان consumer أو التنين مع بعض
- MAP_SHARED ← خلي ال shared memory عامة للجميع ليست حكراً على process معينة
- shm_fd ← file discrepter
- 0 ← لما تحب تكتب على ال shared memory بدل من row رقم 0 offset 0

* AT this point we made three major actions : *

① define we have shared memory with specific size
② we have fixed the size of this shared memory
③ the shared memory become part of producer code

بعد هدول ال ٣ شغلات أصبحت ال shared memory جاهزة لـ producing information

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* ايش نوع الداتا إلي بدنا نخزنها على الـ shared memory ؟ يحدد هذا ايش القيمة إلي بترجع as a return value from map وهو pointer (ptr) وبما انه بالكود حاطين void يعني بتقدر تستخدم any data type لو حددت نوع معين لازم تلتزم فيه الـ shared memory ولكن هون خليناها ginaric باستخدام void .

* عشان تضيف كل إلي عليك تحط عند الـ pointer وبعدين تحركه

* sprint (ptr , "%s" , message0)

ptr ← وين بدك تضيف

"%s" ← نوع إلي بدك تضيفه (string)

message0 ← ايش بدك تضيف

(Hello) → موجودة بالكود سلايد 48

H
E
L
L
O
\0

\0 → terminating string
بتكون موجودة ضمنيا بعد آخر حرف

Ptr += strlen (message0) → تحريك المؤشر بحجم الرسالة إلي ضفتها

↓ برجعلك حجم الرسالة

ونفس الخطوات لإضافة message1

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Consumer code

* To make undefined Behavior not happen you should stick to the things that logically have in common bettween the tow such as:

① ال producer وال consumer من غير المنطقي يكون كل واحد شايف ال shared memory بحجم مختلف (تثبيت ال size)

② تثبيت الاسم ③ ال access write (666) RWE 110 حاول تخليهم أي بعض، بتقدر تغير ولكن لازم تغير بذكاء ال consumer بما انه بده يقرأ أهم اشي يكون ال Read Play = 1 يعني ممكن أستبدلهم ب 444

④ ال Pointers الموجودين بكون ال consumer مختلفين عن الكود أي قبل لانه برنامج قائم بحد ذاته

⑤ رح يكون ال Pointer للي بده يقرأ فأول مكان حتى لو كان مؤشر إلي يكتب بمكان تاني واذا حكيته اقرأ string رح يبحث عن محل معلوم واقف ويقرأ لحد ما يوصل لأول termination string

اذا ناسي تضيف هاي رح يقرأ لك كل ال memory ويصير (undefined Behavior)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

string

جملة القراءة printf("%s", (char *)ptr) ؛

قراءة من pointer location

هاي الجملة ما بتعمل

cout يعني ما بتطبع

اي ال buffer

على الشاشة

ولو بدك تشوف اشي قرأ لازم تتبع هاي الجملة و

printf (ptr)

* أي process بتخلص من ال Taske إي العمل لازم تعمل ال release resources

عن طريق command اسمه unlink ومعناه امسح ال shared memory

أي واحد من ال producer أو ال consumer بعمل unlike بمسح ال shared memory

كاملة

- sprintf (ما بتطبع نهائياً)

printf (حسب ال option إي جاي معاها ممكن توجه ال output على

الشاشة بتطبع أو على مكان ثاني)

- system commands : (system calls)

① send ② resive ③ fork ④ execute ⑤ waite ⑥ exit

⑦ mem-open ⑧ mem-map ⑨ trancat

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* آخر technique لـ interprocess communication إلي رح نحكي عنهم

قسامعظمهم عن الـ message passing وهي عبارة عن representation

لـ ~~Path~~ Path بين الـ sender والـ receiver :

① sockets : networking (not operating system)

معناها دمج 2-id مع بعض

Ip address — Port number

↳ عشان بعدما تتعرف على الـ computer تتعرف على الـ Application الفردي إلي بده يستلم الرسالة

↳ بحتاجة عشان أتعرف على الـ computer والرسالة إلي الك لوين رايحة

* بالتالي لما تربطهم اثنين مع بعض كأنك بتحكي وصل الرسالة لهذا الـ App الواقع على هذا الجهاز

* لذلك كل جهاز connected على الانترنت عنده على الأقل one Ip address مع many port numbers

161 . 25 . 19 . 8 : 1625

↓ جهاز صاحب الـ IP

↳ port number (App)

صار كأنه logical address : الجزء اليسار بحدد لك رقم الجهاز ككل

والجزء اليمين بحدد الرسالة على أي app تروح

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ال Sockets ليس طريقة لل message passing وليس Shared memory
ولكنه بشبه اشي حكينا عنه قبل شوي وهو ال discreptor

* ال Sockets هي Tool بتمكنك تعمل message passing لانه
واضح انه الجهازين اللي بدهم يتواصلوا مفصولين عن بعض تماماً

Remote / procedure calls

ال function أو ال procedure اللي بدك تنفذه لا يقع على جهازك

استدعاء function

* device X do remot procedure call on device Y to exeate function exist in device Y.

* هذا الاشي يتم وراء الكواليس وانت ما بتعرف عنه وتصير هذا الاشي اذا كان ال Subrotiure / function لا يقع على جهازك وانما تقع على servier مثلاً

* بالتالي بصير في تنفيذ للكود ولكن عن بعد وبترجع من الجهاز البعيد للجهاز اللي عمل issue لعملي ال communication

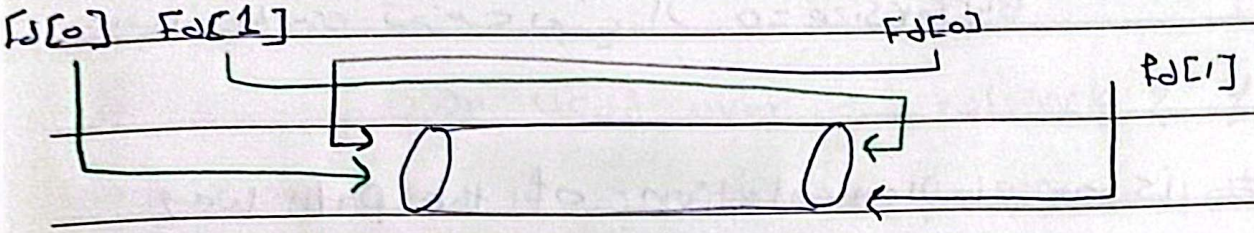
* Stubs = discreptors , end poits of communication
يعني التواصل وين يبدأ ووين ينتهي وعنا 2 stubs زي ما كان عنا 2 sockets

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Pipes

* ال pipe بكون مربوط بين 2-process



* موام 2-process يعني العمل تعاوني cooperative واحد يرسل والثاني يستقبل لذلك عن writing discreptor = Fd[0] reading discreptor = Fd[1]

* هذا ال pipe بستخدمه 2-process على الأقل A,B

* إلي بصير كأنك بتقسم ال pipe قسمين: بتحط بالنصف العلوي وبتأخذ من النصف السفلي

* لما نحط بالنصف العلوي بنستخدم writing = Fd[0]

بال coding بنستخدم Fd[0] ال producer process عشان نحط داتا على ال pipe زي ما عملنا بال shared memory

* بالمقابل ال consumer لما يجب يأخذ الداتا إلي انكتبت على ال pipe بتفرج على ال reading discreptor = Fd[1] فيأخذ النصف السفلي من ال pipe

* إذا الموضوع بشبه ال shared memory ولكن بمسميات مختلفة لازم نقارن:

pipe	shared memory
- العملية transient يعني إلي بنحط لازم يتأخذ على طول لانه نش تخزين من اسمها pipe (زي بريش اعي) (path بين المرسل والمستقبل)	- بتقدر تخزن أكثر من رسالة لمدة طويلة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• ملاحظة : ال Pipe هي Tool بتساعدك تعمل message passing
وبما انه ماعندي تخزين يعني ال Buffersize = 0

Pipe = is an implementation of the path way
beftwen sender and receiver to facilitate (يسهل)
the message passing.

* بحاجة وجود 2-process و 2-discrepter وكل واحد مربوط بعملية قراءة أو كتابة

• We have Tow types of pipes:

① ORdinary pipes : - بتكون بين أب وابن يعني فرضاً
ال producer هو الاب وال consumer هو الابن أو العكس

② Named pipes : - وكأنها shared memory في أكثر من
process مش شرط أب وابن ممكن أي 2-process
يستخدموه. No perant-child relation is nessary
to use Named pipes.

Bidirectinal بدل 4-discrepter ← read/write على الطرف الأول
← read/write على الطرف الثاني

unidirctinal بدل 2-discrepter ← جهة ارسال
← جهة استقبال

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

بال Pipes ال comunication بكون half dublex ولكن بتقدر تختار full dublex

* can the pipe used over the network? Yes,

بس ببطل اسمه pipe بصير على الأغلب بشبه ال socket

أو اذا كانت function بال reslut ← interprocess communication RPC

* ORdinary Pipes: can not be accessed from outside processed tree *

* NAmed Pipes: No limit of how use it

Threads:

* مفهوم مستخرج من ال process يعني اذا

عندك process بكون على الأغلب في threads

اذا ما في process على الأكيد ما في threads

$$\text{process} = \sum \text{resources} + \sum_{0}^{n} \text{executions units}$$

(reg, files, memlocations, PC (address space))

جمل تنفيذية

← الجزئية التنفيذية من

ال process بتصير هي ال threads

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

• لذلك لان مفهوم ال threads نابع من ال process وال OS لما يتعامل
مع ال process كلها ال threads وكأنه ما بشوفها

threads : هي جزئيات تنفيذية داخل ال process نفسها

* How many execution units is in process ? many
* every execution unit is a thread

تعريف ال process الجديد :

process : مجموعة من ال thread
تعبر عن ال execution unit
وتوابعها من ال resources

- لما نسمح cpu utilization مبني اعزول عنه ال resources وال execution unit
بال process ؟ أكيد execution unit اعزول عنها ال threads
وبتضمنلك انه ال cpu يضل شغال لمدة طويلة

- لذلك أي برنامج حديث نظام multithreads يعني البرنامج internally
مقسم إلى tuskes وكل tusk معزول عنها thread

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

- لما تحكي انه البرنامج فيه single thread / ح تستنتج انه برنامج سهل وما بقدم الا وظيفة واحدة فقط

• The majority of applications now is multithreaded.

والدافع لاستخدام ال multithreaded : لو قدرنا نشغل أكثر من thread مع بعض معناته هيك احنا بنشغل ال cpu أكثر ما يمكن و بالتالي يكون ال utilization أعلى ما يمكن ويكون أسرع

* single thread can not keep cpu Busy *

* هذا لا يعني عدم وجود برامج single threaded .

* multithreaded هي وصف ال app أو ال process مش ال cpu

* ال app بي شغال اذا كان عنده subworkers / subexecution components داخله معناته عنا multithreaded

* لو كان عندك single cpu وبصير single process هذا شو أثره على ال multithreading ؟ شبه على الفاضي من ناحية ال multithreading

* انت عامل برنامج وكاتبه منيح بحيث عندك 3-threads ومقسمه هيك على أمل وجود 3-cpu's تعطي كل cpu ← thread ال utilization رح تزيد لانه كل ال cpu شغالين ورح يصير أسرع

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

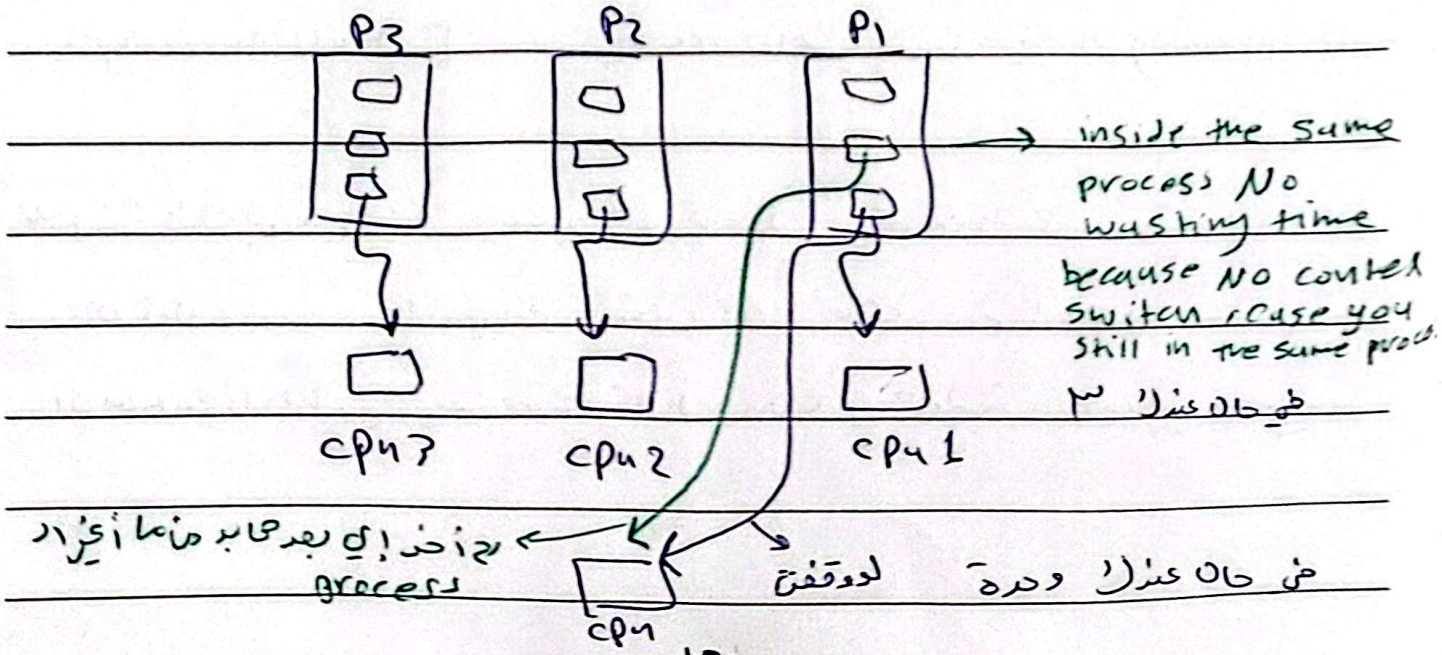
Date / /

* لو عندك single cpu الاستفادة من ال multithreading مارح تصير صفر ولكنها رح تهبط مرقم كبير.

* طيب ليش الاستفادة ما صارت zero ؟ ال execution unit الموجودة اما يكون نوعها cpu-bound أو i/o bound ، هل بتضمن انه ال part إلي شغال عليه ال cpu يضل شغال من البداية إلي النهاية بدون توقف أكيد لا لانه ممكن يصير له fork a child أو waiting for interrupt أو ال Time slice إلي اله تخلص ، لما يكون عندك threads وتوقف ال part1 لسبب ما رح يعمل switch لل part2 ولل part3 وبالتالي اله فوائد.

* طيب لو بدل تقسيم أكثر من thread خليناهم كلهم بـ thread وحدة ؟ اذا توقفت أثناء التنفيذ رح تحتاج تدور على بديل خارجي تمامًا خارج ال process

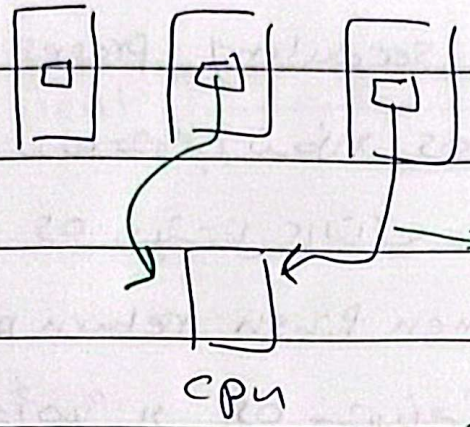
* بينما اذا عندك multipul threads ما بتحتاج تدور خارج ال process



الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* طيب لو عملنا هم Single thread :



اذا
هاي وقفت
لازم تدور على
بديل خارجي

وهذا اسمه contexswitch وله تكلفة زمنية
لانك بدك تخزن ال PCB وتحمل load
لل old PCB بالتالي wasting time

* ال optimal اذا عندك multithreading يكون عندك multiprossers
اذا ماشي بتحصل على استفادة ولكنها مش بنفس الدرجة لي يكون
عندك أكثر من cpu .

* التعامل مع ال process يكون heavy ← لانه في os بتدخل بكل اشي
* التعامل مع ال threads يكون light ← ماشي os بتدخل

المقصود بـ heavy : resources / Time

light : انك تنشأ thread جديدة تكلفتها الزمنية وال resources قليلة جداً

- اذا كان عندك Big task من وجهة نظر ال os بتقدر تعمله بطريقتين :

① اما تقسمها على برنامجين (2-process)

② أو تقسمها على threads ← أفضل من حيث : speed , resources , contex switch

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* process creation, process secduling, process termination
is heavy-weight . لانه بتطلب تدخل ال OS
وال OS بشتغل كالتالي Switching from usermode
to kernel mode and when finsh return back
to user mode . وأصلاً ال OS مش فاضي وبكون
مشغول باشي تاني فبدك تستناه يخلص

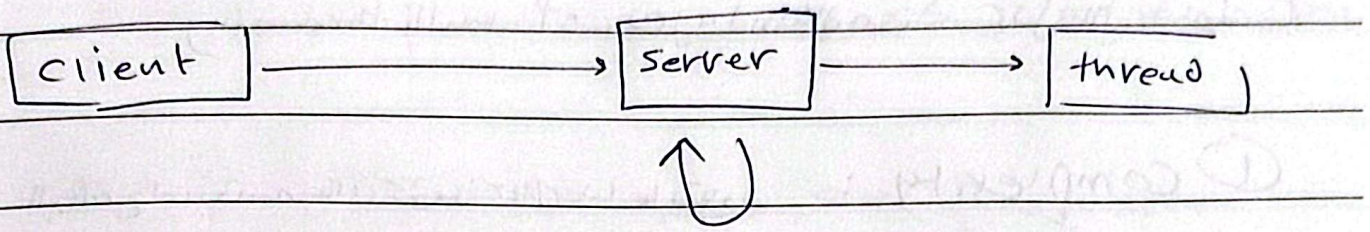
* thread creation, thread scaduling, thread termination
كلها light weight و أيضاً سريعة
* الان صار في switching من multiprograming لل multithreading
multiprocess
أكثر من app
شغالين مع بعض
بتعلمها عن طريق
كل Process جواتها
أكثر من thread بالتالي
العدد الاجمالي لل ~~thread~~ process
رح يقل وتدخلات ال OS بتقل
وأداء الوظائف بصير أسرع

* This can simplify the code:
① increse the efficiency, case no contex switching
— Kernels are mutithreaded ال Kernel اصلاً معقد وسؤال ا
عن كثير اشياء بالتالي ما بنفع يكون عنده single thread وإلا رح تشوه دايماً معلق
لذلك فيه multithreading

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

client → server → thread



ال servers بكونوا

معمولين بالطريقة التالية :

① thread مسؤولة انها تعمل threads

ال main thread بتكون على بوابة ال server أي عميل بيجي بيستخدمة

ال thread بتعمل اشي اسمه dispatch smaller threads لكل واحد وبرجع

يجهز لاستقبال الطلبات

* أي طلب بنعمله secondary thread

* بالتالي عدد ال threads الموجودين بأي server = 1 + n

1 → دائماً موجود واسمه demain thread

n → عددها بزيد أو بنقص حسب كم node موجودة بال server

Will Continuously running threads

- Benefits of multithreading :

① Responsiveness : تقليل ال response time (وقت البدء بالعملية)

② Resource sharing : Same as interprocess communication

ال thread بتستخدم نفس ال resources الموجودة لل process (بتستفيد من الموجود)

③ Economy : مش من وجهة نظر (اعصاري) لا إما time أو resources

(وقت أقل، تكلفة أقل، مصادر أقل) overhead أقل

④ Scalability : multiusers بدعم أكثر

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* Tow major disadvantages of multithreading

① Complexity : البرنامج لما تقسمه على أكثر من worker ما بتعدد معمة المبرمج سهلة

② Synchronization : بدك ترتب وترا من كل ال threads مع بعض

* Threads : هي الجزء التنفيذي ال process ، ال process تحتوي على resources بعضها ل PCB وجزئيات تنفيذية وهي ال threads .

• هل لازم يكون داخل ال process أكثر من thread ؟ لا ، ولكن على الأقل واحدة فقط وتوقع انه ما تستفيد عن ال multiprocessers

• ما يميز ال process عن ال thread :
↓ light-weight ↓ heavy-weight

وجودها يرهق ال OS وجودها لا يرهق ال OS لانه معمام ال thread على عاتق ال process نفسها

• بالتالي تصبح ال architecture كالتالي : كثير من ال threads داخل ال process وال threads مسؤوليتهم على عاتق ال app نفسه (process)

الممسوحة ضوئيا بـ CamScanner

Memo No. الأربعاء
Date 30 / 7 / 2025

الـ OS شايف بس الـ process والـ process شايفة الـ threads

* can simplify code ← جملة غير دقيقة لأنه
لما بدك كود ينشأ أكثر من thread
ويوزع اشغل عليهم ، هذا عبئ
على المبرمج ، بالتالي لا تصبح الأمور أبسط (ممكن تكون أبسط بالنسبة للـ OS)

* Increase efficiency ← ليست مضمونة 100%
عشان زيادة الـ threads تحققلك
efficiency لازم يكون في multipulcores
ولازم الـ threads يكونوا مبرمجين صح

* Kernel as a complex software must be multithreaded

* كل ما صار البرنامج معقد كل ما لزم وجود أكثر من thread

* thread contex switch تكلفة أقل من الـ contex switch العادي

* many processes is expensive not scalabile , many threads not expensive , scalabile

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

- Multiprocessing ← أكثر من processer
- multiprograming ← أكثر من برنامج في ال mem ومش شرط كلهم شغالين ولكنهم حاضرين يشتغلوا
- multitasking ← نفس ال multiprograming ولكن البرامج كلها شغالين مع بعض على الأكيد (Time sharing)
- multithreading ← أكثر من thread داخل process وحدة

- تحديات تواجه ال multicore أو ال multithreads

① كيف المبرمج رح يعمل dividing activities (ال task الكبيرة كيف رح يقسمها بين ال threads)

اعفروض يقسمها بالاعتماد على (data dependancy) ②

قسم الداتا وتأكد انهم ما بعتمدوا على بعض عشان يقدروا يشتغلوا بنفس الوقت وما حدا ينتظر الثاني (مهمة تقسيم الكود على عاتق المبرمج)

③ Balance : انك توزع الكود بشكل عادل على ال threads

- الكود لا يقسم بشكل عشوائي لانه ممكن يصير في مشاكل بال Balance يترتب عليها انخفاض بال utilization ومشاكل بال dependancy

- أي function بشتغل بحتاج إلى داتا وهاي الداتا ممكن تكون محدد لل execution

يعني فرضًا عندك thread 1, thread 2

← بدها تغير على قيم array

↓

بدها تشوف قيم ال array وتشتغل على sorting فرضًا

+ جدول الثنين ما بنفع يشتغلوا (In parall.) مع بعض بنفس الوقت

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لانه الداتا is not spliting (مقسمة) ، المفروض انه ال thread بعد ما تعبي
تقسم اشغل بين ال 2-worker .

* وجود الكثير من ال threads يحققلك أحد المفهومين التالين :-

① parallelism : work at the same time

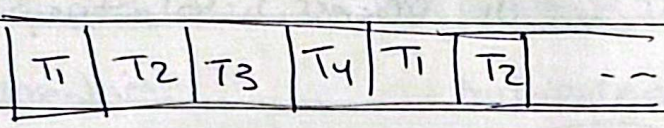
② concurrency : they are coexist

موجودين مع بعض بس واحد

بيشتغل

single core

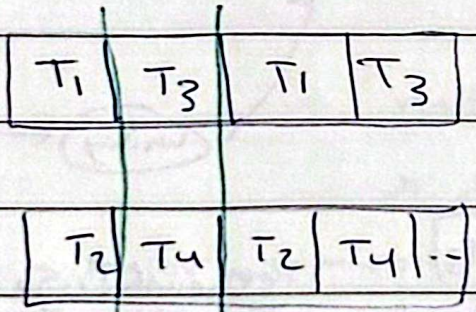
T1	T2	T3	T4	T1	T2	--



(concurrent) at a time one is execute

بسبب وجود cpu واحد ، الفائدة هون من وجود أكثر من thread انه في حال T1 خلصت بسرعة
، ح يجي مكانها T2 بالتالي بتخلي ال cpu شغالة

multicore



T1	T3	T1	T3	--
T2	T4	T2	T4	--

t1 ⇐ T1 ، T2 شغالين

و T3 ، T4 من مبينين
يعني هم بال waiting state

الوضع المثالي عند وجود
4-threads يكون عندك
4-cpu يشتغلوا
In parallel ويخلصوا أسرع
ما يمكن ولكن بالشكل
عنا 2-core

(parallelism)
↓
the ability to run
multipule threads on the same time
because multiprocesser is exist

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

- The degree of multithreading : the number of threads exist , and ready for execution

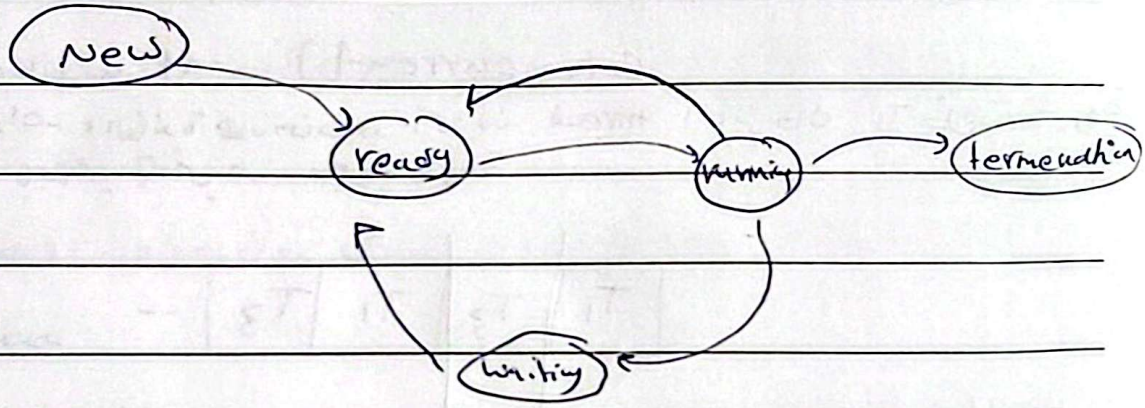
بعض انتظر شغالين أو لا

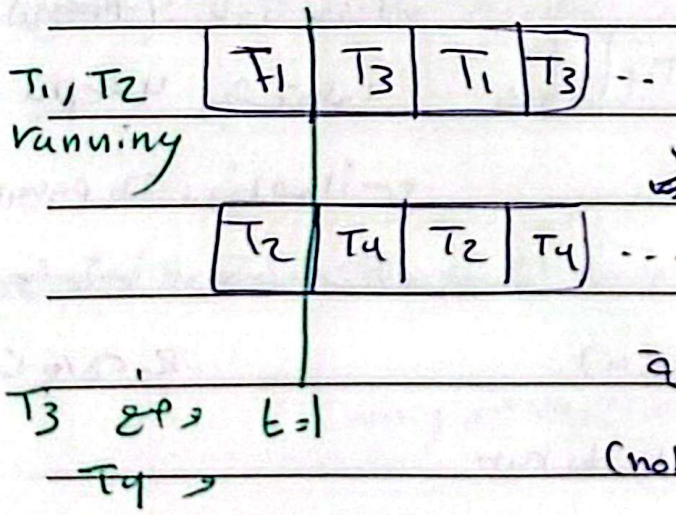
زي بالرسمة الصفحة السابقة :

4-concurrent يعني واحد شغال

4-parallel يعني أكثر من واحد شغال وتعتمد على عدد ال cpu

- ال Thread بتمر بنفس مراحل ال process الخمسة





* اذا process اختفت بتكون termenated
* اذا وقفت لسبب خارج عن ارادتها بتروح ready
* اذا وقفت من تلقاء نفسها بتروح waiting
* ال thread يختفي بمجرد انتهاء المهمة التي وجد من أجلها (not physical worker)

T3 وضع
و T4
في حالة ال ready

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

① Data parallelism

* بحكي عن البيانات إي بدها تشتغل عليها الـ Tasks ، يعني لو عندك database وبدك ترتيبها

Task : عملية الترتيب نفسها

Data : البيانات الموجودين

قسم الداتا إلى أجزاء ورح ينتج عن ذلك توزيعها لأكثر من thread عشان يشتغلوا عليها

Speed up (throughput, utilization)

② Task parallelism

* عندك Task حاب تنفذها parallel

أول اشي : قسمها إلى subtaskes

ثاني اشي : أعمل thread لكل واحد من الـ Tasks

وبستفيد إذا في أكثر من cpu

* Distributing the threads across the core's

* لو حدا حكالك انه عنده Task parallelism معناته أكيد عنده multipule cpu

① speedup ② low response time

throughput utilization

multiput HW threads

بدي أعمل اجراء بالـ HW يدعم الـ multithreads

threads → [ ] [ ] [ ] [ ] P1

SW هذا

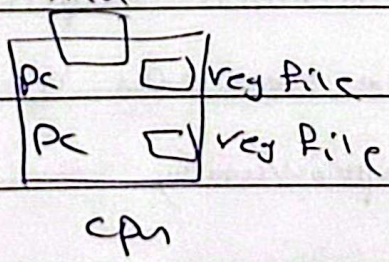
major unit → A1U

عا بصير معنى نسختين

لانه لو ضاعفت كل اشي

بطلت cpu وحدة صاروا

ثنتين



بدك تكثر الوحدات داخل الـ cpu عشان تقدر تعمل parrallel execution

ضيف عدام في A1U وحدة يعني بالآخر ما رح تقدر تشتغل اشين مع بعض

هذا إي بصير ← Standered (SMT)

(لو ضاعفت كل اشي الـ A1U رح تضل وحدة) (simultaneous multi Threading)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بما انك محضر حالك ومضاعف كل شيء بال cpu وناسي ال A\U
رح تلاقي بالآخر انه ال threads شغلين serial ولكن بطريقة توحي
انهم parallel .

* يعني لو اجاك برنامجين ال cpu رح يقبلهم وبما انه عندك 2-PC
رح يعطي كل برنامج PC و يصير جاهز على التنفيذ ، رح يبدلش بواحد منهم
مجرد ما يخلص رح ينتقل على الثاني بدون الحاجة إلى contex switch

* Modern applications (process) mostly contains many SW threads ، عدد ال threads داخل ال app
متغير من برنامج لآخر

* Every cpu (core) has many HW threads

- SW threads: subworker inside process can run or stop
- HW threads: duplication of cpu resources (ماعدا ال A\U)
make the cpu ready for the higher degree of multiprograming

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

cpu1	cpu2	cpu3	cpu4
H.T H.T	H.T H.T	H.T H.T	H.T H.T

H.T = HW thread

لو حكالك عندك 4-cpu

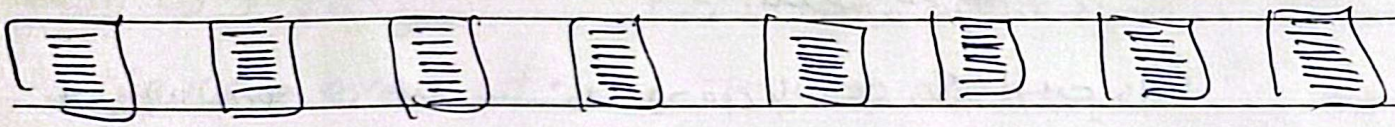
ولكنهم بدعموا لحد 8

يعني بكون بكل cpu عندك 2-HW thread

* صح انه بدعم لحد 8 ولكن في الواقع عدد إلي ممكن يشتغلوا بنفس الوقت

فعلياً هم 4 ولكن بقدروا يدخلوا 8 ويصيروا يعملوا Contex switch اسرع

* oracle SPARC Tu : 8 cores , 8 HW thread



عندك 8-cpu وكل وحدة فيها 8-HW thread يعني بتقدر تستوعب لحد

64 برنامج ولكن إلي شغالة مع بعض بالواقع هم 8 والباقي

جاهز للتشغيل وبدون contexswitch ولو واحد وقف بكون اله 7 بدائل

حل ايد 10 : لو حبيت تعمل multiple thread الـ resourcess ايش موقعها

degree of mutithreading = 3

code	Data	Files
reg	reg	reg
stack	stack	stack
∫	∫	∫

thread

Process

أشياء لازم تضل مشتركة : code , Files , Data

أشياء مستقلة بكل thread : reg , stack

temporary story

عشان لما يشتغلوا مع بعض ما يخربوا على بعض

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

"Amdahl's Law" : $speedup \le \frac{1}{S + \frac{(1-S)}{N}}$

S ← code لازم يشتغل serial لسبب ما ، يعني لو عندك كود 1000 سطر ، 20 منهم لازم يشتغلوا serial

$(1-S)$ → باقي الأجزاء أي شغالة parallel

N ← عدد الـ cores

القانون هذا بقول لي تكثر عدد الـ cpu ايش إي بصير وكم بتحصل على speed up (نسبة يعني 20 من أصل 1000 وطلع النسبة)

* هل نضمن دائماً حدوث speedup ؟ انت جهازك في cpu وحدة وصاحبك عنده 2-cpu هل حد بنجح دائماً أسرع منك ؟ المفروض آه ولكن مش بالضرورة زي لما يكون عندك task has one thread أو يكون عنده 2-thread ولكن الثانية تنتظر الأولى

* speedup هو هدف منشود يحقق بالغالب ولكن ليس دائماً

- لو حكينا $N=1$ هل رح نحصل على speedup

$speedup \le \frac{1}{S + \frac{(1-S)}{1}} = 1$ ما في speedup ، أي رقم أكبر من 1 يعتبر speedup

* لو $S = 75\%$ وزدنا الـ cpu من 1 إلى 2

$speedup \le \frac{1}{75\% + \frac{(1-75\%)}{2}} \le 1.6$ ← speedup

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لو جربت ترفع ال $N=\infty$ ، $\frac{1}{S} = \frac{1}{S+\frac{(S-1)}{\infty}}$ ، $\text{speedup} \leq$

وكأنه بقلك أصبح

الجزء الفصلي 0 و الاسطر التي تنفذ Serial

مش = لانه لوحط

يساوي وكأنه راح

للظروف المثالية

* لحد الآن قمنا بتصنيف ال threads إلى نوعين : HW , SW

SW → at the process

HW → داخل ال cpu ويتم مضاعفته

* عنا تصنيف آخر : user thread , kernel thread

user thread → application مطورة المستخدم وليس ال OS وللمستخدم صلاحية مطلقة عليه زي Posix Pthread , window thread Java threads

ال OS مالها علاقة فيه نهائياً وأي عملية تتصير من معمار ال app نفسه

* هذا النوع ما بقدر يوصل لل HW

* عدد ال user threads أكبر بكثير من ال kernel threads

* وجودهم مؤقت عند عمل الخدمة

kernel thread → جزئية من ال OS بالتالي أي thread من هذا النوع ال mangment , creation , termination , assigning task من معمار ال OS

* موجودين مدام ال OS شغال windows , Solaris linux , Trugu unix mac os x (وأنواع ال OS)

الممسوحة ضوئيا بـ CamScanner

Memo No. الخميس
Date 31 / 7 / 2025

Multithreading Models

① Many - to - One :
user (client) — Kernel (server)

السرعة بطيئة بسبب وجود الكثير من اعتمادين و Kernel واحد

* لو واحد حجز الـ Kernel الباقي رح يكونوا ready

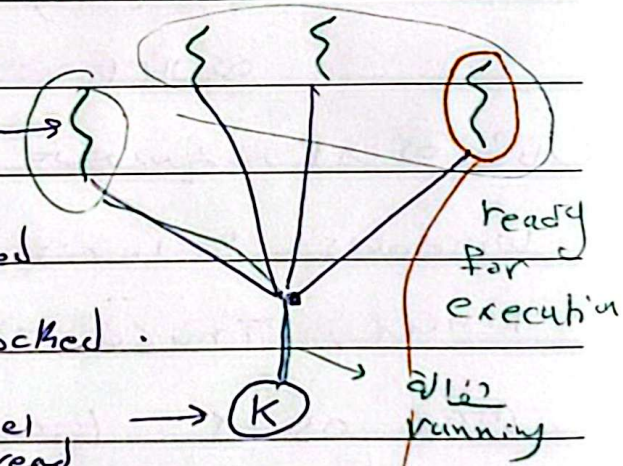
② one - to - one :
user — Kernel

③ Many - to - Many :
user — Kernel

① Many - to - one :
many - user level threads From the same process is connected to single Kernel - level threads

جميع الـ user-level-thread هم بالنهاية ينتموا إلى single process

degree of multithreading :
4 = بالرسمة



* if one user level thread is blocked all the other user level thread is blocked.

كونه عندك single cpu اذاً من اصل الـ 4 اعوجودين واحد رح يشتغل

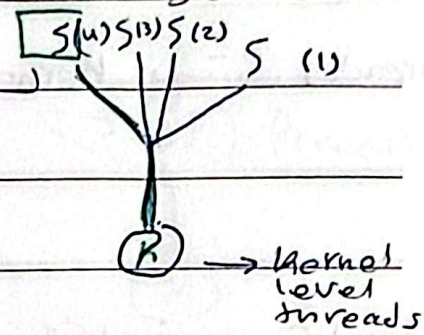
لو حبيت أشغل

هاي ولسا الأول ما خلصت : ما بنقدر لانه الـ Kernel level أصلاً مشغول بالتالي احنا مضطربين نعمله Block

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* اذا ال thread إلي شغالة عملت Block (ممكن لانها عملت ابن وبدها تستناه
أو انها تنتظر i/o action أو ال Time slice إلي انها خلصها)



لـ ٢ هاي إلي شغالة عملت Block ، هيب ليش ما نختار
من ال ٣ الباقين ؟ بما انه ال Kernel level threads مسؤول عنه ال OS
هو ما بعرف أي شيء بخصوص ال user level threads .

* يعني على فرض إلي عليها ربع أخضر عملت run ال Kernel
بعرف انه في وحدة شغالة بس بعرفش أي وحدة أو حتى رقمها
أي هون هي رقم (4)

* بالتالي حتى لو توقفت thread رقم 4 بسبب systemcall
او i/o request ما بقدر ال Kernel يختار وحدة ثانية لانه بعرفش
انه في آخرين

* كل إلي بعرفه ال Kernel انه في process شغالة وبالتالي
في على الأقل one thread شغالة ولما ال thread إلي شغالة
تعمل Block ال OS بعرف انه هاي ال process تعطلت
ال thread الخاصة فيها حالياً بس ما بعرف اذا في ٣ غيرها

* one user level thread is Blocked (waiting some event)
we can not scedual anther one , because OS can't
go inside the process and know it's valid.
(Kernel know the Big picker)

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* طيب مين بعمل manege لل user-thread معناته؟ المبرمج المسؤول عن ال thread library ، طيب ليش ما المبرمج يكون smart ويعمل switch من 4 إي وقفت لـ 3 مثلاً؟ لانه ال Kernel بستنى 4 thread لحتى ترجع لل running

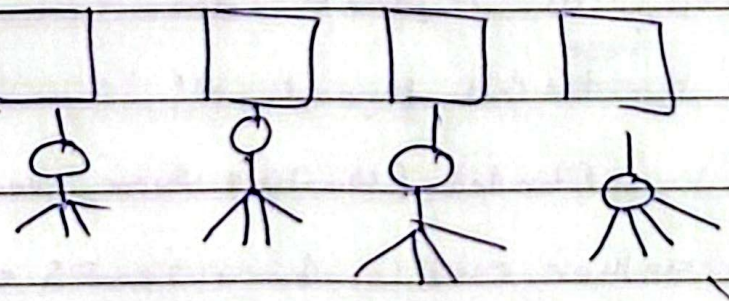
* many to one model غير متداول حالياً لانه مشكلته أنه بيستطيع تشغيل واحدة فقط من ال user-level-threads ويوقف البقية نظراً لانه في multipul clinets و single server وبالتالي لا يحقق استفادة من ال multithreading الموجودين

* حتى لو عندك أكثر من cpu عالفاضي لانه متلقي الخدمة واحد فقط بالتالي هذا (concerancy)

Solaris Green Threads , GNU Portable Threads

أنواع شغالة بنفس الطريقة

أفرض عندك 4-cpu



متى الموضوع بتحسن؟

لو كان عندك multiprograming في جواته multithreading

* لو بتشتغل بهذا النظام حتى على الأقل عندك 4-process عشان تستفيد من ال 4-cpu

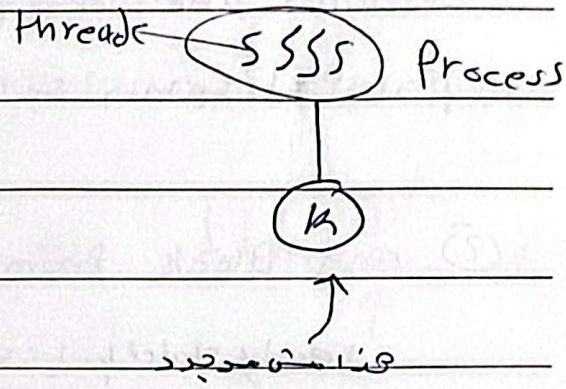
لو كان هيك 2/ يكون concerancy مع شوية parallel

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* كيف module may to one ليتم إنشاءه؟ انت كـ user عملت creation لـ process وداخلها عملت creat لـ أكثر من thread

الآن الـ OS إلي شغال يعمل الـ modul



بروح عامل creation لـ one kernel level thread

وشابكها مع الـ process بجميع محتوياتها

معمول عشان

نخدم الـ process أول

دائماً بالـ system ، لما الـ process تخلص هو بختفي

عالـ threads إلي جدة يخلصوا بوصل notification لـ process نفسها انه

وجود لها (on demand)

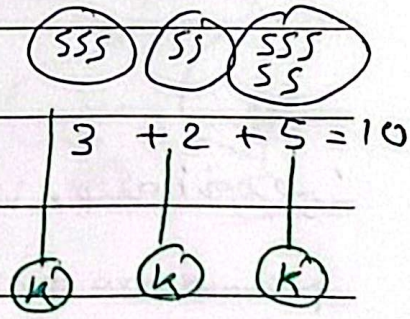
الـ task is done وبالتالي تتحول لـ process إلى tarminated state

وبعد عالـ kernel level thread بتروح

بالـ many to one :

① كم عدد الـ user-level thread؟ ما بعرف ، ولكن

عددهم = ~~process~~ ← كم thread داخل الـ process



② كم عدد الـ kernel level threads؟ واحد لكل process

لذلك عدد الـ kernel = عدد الـ process

* advantegs :

① user level threads is light waight because it maneyed and created by user ② number of kernel level thread is small

③ this modle is the most light waight

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* disadvanteyes :

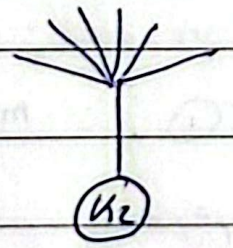
① because they cannot run in parallel, then the execution time for a process does not achieve the parallelism, so it is not fast

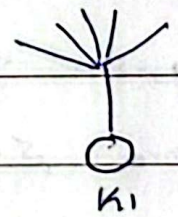
② any Block from any of the threads اح يتحجز ال server ويتعطل والتاتين كلهم قاعدين بستنوا بال ready state

* يحقق ال concurrency أكثر من ال parallelism

عندك $P_1 \rightarrow$ 5 user level threads و $P_2 = 6$ user level threads

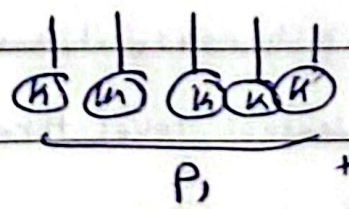
عدد ال Kernel = عدد ال process
2 = 2





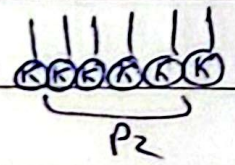
- التكلفة على ال os قليلة لانه فقط أنشأ 2-Kernel ، يمكن وصفه أنه بطيء ولكنه خفيف

if you want to switch to real parallelism :



انشأ لكل thread ← Kernel thread

عدد ال Kernel = عدد ال thread
11 = 11



الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No. _____

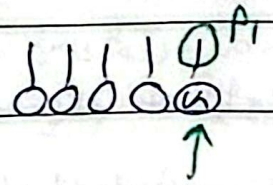
Date / /

② One to one

* لما تحكي انه بدك تعمل ll-Kernel level thread فهاي expinsive من ناحية وقت ال Kernel وانه عطلناه كثير ومن ناحية ال resources ومن ناحية ال mangment ، لانه صار ال os الراعي الرسمي لكل thread

- بينما سابقاً كان ال os الراعي لل process ولا يعلم شيء عن ال thread

* علماً انه ال os ما بعرف انه ال Kernel-level thread مشبوكة مع user thread تابعة لـ P_1



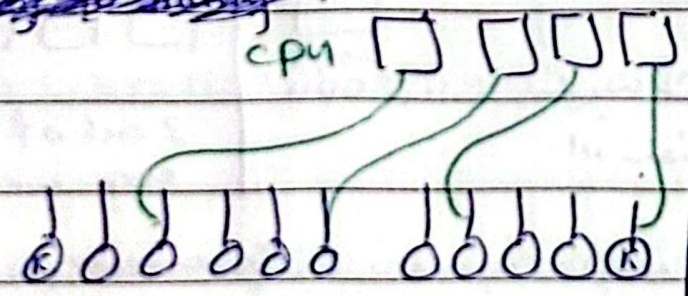
* هو يلعب دور ال server وما يعمله To much information about client بعاملهم كلهم بنفس الطريقة

* وهون كل ما يخلص ال thread ال Kernel يختفي معه

* هو اثقل على ال os ولكنه يوقت utilization عالي وايضاً يحسن ال throughput

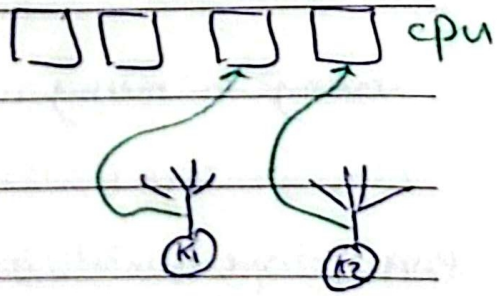
هذا هو مبدأ ال (one to one)

~~③ Many to many~~ :



توزيع عشوائي

4 thread are running ، عشان 4 واي واحد بخلصها روح على الـ بعده على طول وعلى الاغلب هون رح نخلصها أسرع لانه محدش بستمر هدا



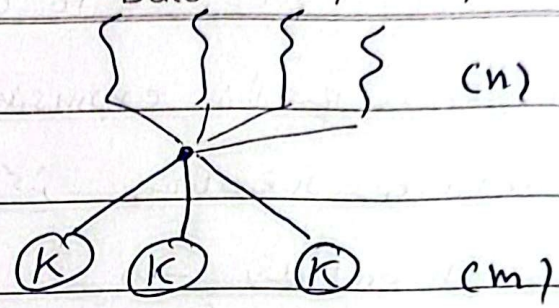
2-core are running

اما هون عشان تعمل switch لـ process ثانية لازم تكون مخلص كل ال thread اي فيها

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

③ many to many :



حاول يدمج بين النوعين السابقين

هو ليس one to one لانه لو كان

لازم ال 4 threads يكون الهم 4-kernel level thread

* عندك أكثر من user thread

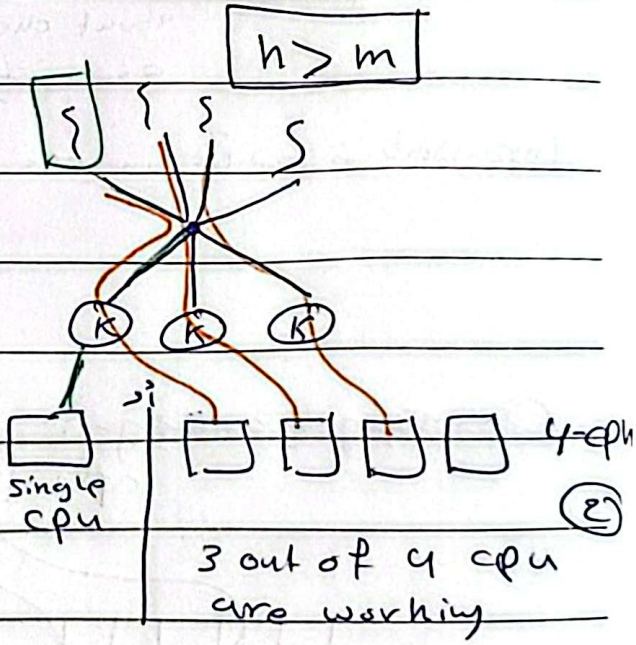
مشبوكين مع أكثر من kernel thread

وليس many to one : لانه ال 4-thread مش

ولكن عددهم أقل من ال user thread

مشبوكين مع 1-kernel level

فهو حالة وسط بين النوعين السابقين



① وحدة بس رح تكون شغالة لانه في cpu وحدة

many to many : ليس شائع الاستخدام

لسانه concurrency ولكن

بحقق درجة عالية من ال parallelisme

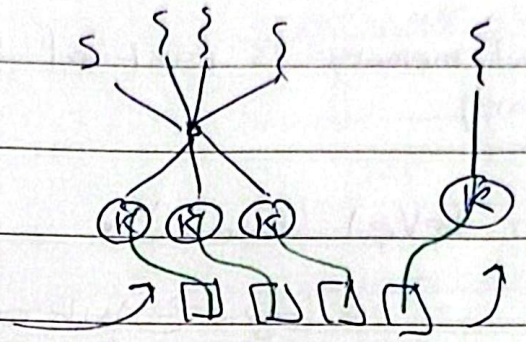
ب parallelisem عالية degree

ب lightwaigh عالي ال OS effecte

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

④ Tow level Model



فرضاً عندك هاي مميزة
وبدها سرعة عالية وما ينفع
تحطها مع الباقي لانه رح تصف عالدور
بالتالي رح أعملها one to one لانها مهمة

والباقي هدول بحطهم many to many

* الآن لو عندك 4-cpu
رح يشتغل 4 من أصل 5 من
ال thread ولما تخلص
وحدة من ال many to many أو
من one to one رح تيجي الآخرة
مكانها

* لسا Concareng ولكن
في parallelism أعلى والدليل
انه عندك 5 thread ولكن إلي شغال
مع بعض 4

كيف المكتبات إلي بستخدمها الجرمج بتساعده : Thread libraries

انه يعمل user-level thread

library entirely in userspace	kernel-level library suported by OS
انت كمبرمج بتقدر تعملها import بتمكنك	احنا بنحكي عن Kernel threads و
تعمل user level thread وتعطيهم tasks	واحنا كمبرمجين we can not see them
وتشوف كيف شغالين	we can not control them

الممسوحة ضوئيا بـ CamScanner

Memo No.

Mo Tu We Th Fr Sa Su

Pthreads (insid posix)

Date / /

* Shared memory is part of posix and Pthreads library Library

* user level thread.

* بساعدك تعمل اي بدك ياه بس ما بقلك كيف

- Pthreads Example (سلايد 20)

include <pthread.h> → متعلقة بكل اشي بخص ال threads

Pthread-t tid; وكأنها → int-t tid (المتغير)

بخزن id لـ thread

* لما تحب تعمل creation لـ thread وتعرف رقمها بتخلي بطريقة معينة جملة ال creation ترجع الرقم لهذا المتغير.

- اذا رقم ال thread ما بعملك انت بس بدك تعمل thread تسلمها task وتشوف النواتج ما بعملك تستخدم هذا ال id

Pthread-attr-t attr;

أي process الها خصائص (attributes) زي ال PCB list of files بالعادة الجملة هاي نتجاهلها وبالتالي النظام بعطيها ال default attributes ولكن ال thread ما الها خصائص من هذا النوع

ولكن الها زي Schaduling proiarity

واذا بتحب تاخد مكان معين بال mem وهكذا

الممسوحة ضوئيا بـ CamScanner

Memo No.

Date

ثاني parameter كتبه اعبر مع screen

Pthread - creat (&tid , &attr , runner , argV [1]) ;

do aditinal thread

اعرف الة

null → defult attrubitte

اسم الـ function اي رح تقوم الـ thread بتنفيذه

لما نحب نعمل تنفيذ لأي برنامج بنحط

gcc abc.c -O L

اسم الملف

علماً بأنه أول ما يبدلش البرنامج بكون يحتوي على single thread

ومع كل fork بصير عندك process ثانية ومع كل Pthread creat بصير عندك thread ثانية

./L → ثان يشتغل

./L - - - - → execution parameter

argc / argV بساعدوك تستغل الـ parameter اي بيجي بعد الـ execution file

./L [- - - - -] عدول

argc : يمثل عدد المتغيرات اي انكتبت بعد الـ command

./L (1) (2) (3) (4) (5) خمس متغيرات فبصير الـ argc كأنه int يحتوي على عدد الكلمات اي كتبتها على الشاشة لما اجيت أنفذ هذا البرنامج

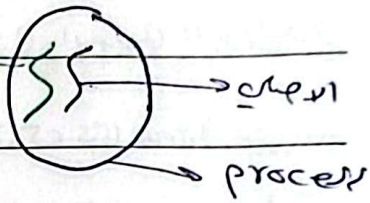
argV : array وظيفتها ترجعلك أي واحد من عدول باستخدام الـ index الخاص فيه ، مثلاً argV[0] بده ما كتب على الشاشة أول موقع اي هو (./L)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

المشكلة انه ال argv[] هي char يعني string وبالتالي
اذا صارفت رقم رح تحوله لـ string

Pthread_creat وتنفذ كود ال runner



* دائماً لما يصير عندك 2 execution Scenario عشان ما تطلع النتائج غلط لازم توقف
واحد وتمشي الثاني

* كود سلايد 12 جزئية ال runner كاملة رح ينفذها t2
وال t1 الاصلية كملت هيك ممكن ال t1 تسبق t2 وانا
t1 خلصت البرنامج كله بروح لانه هاي ال thread الاساسية
وهيك كانك تعمل discard محتمل لـ t2

* اذا t2 كانت أسرع من t1 رح تخلص الجزء المسؤولة عنه
وهذا الاشي إلي بدناياه أصلاً

* عشان ما يصير عندي خطأ رح نستخدم Pthread_join(tid,Null)
wait
إلي بوصل لهاي الجملة لازم يعمل wait لـ
اسمه مكتوب بين الاقواس
وهيك مستحيل الاب يسبق الابن

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

Pthread-exit(0) تخبر إلي بستن فوت انه كمل شغل خلص
وكأنه دور على أي join فيما bid واحكيله انه خلص تنفيذ
بتقدر تكمل . وهي غير ضرورية لانه تسكيره القوس أخر شي
بتعمل نفس التأثير

بكود ال runner اختلافي upper , i هدول local value
تخص ال bid كـ thread فرعية
وفي global value زي ال sum وال void* runner(void* pran) سلايد
وهي مشتركة بين الاب والابن Function

Thread local storge : سلايد 39

- ال thread بكون عندي نسخة خاصة من الداتا زي ال (i , upper)
وتنتمي لل thread الفرعي فقط .
- ولكن القيم ال globle ممكن كل ال threads تشوفيها

Implicit Threading : سلايد 28

* يعني معمة عمل ال thread تكون ليست على المبرج ومعمة اعطاء
ال thread ال task وتنظيم العمل بتكون على المبرمج

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* فكانة مهمة انشاء ال threads واعطائهم tasks وتتبعهم قد تصبح complex اذا البرنامج صار كبير وعدد ال thread كثار

- عشان هيك فكرنا بطريقة انه ال run-time libraries او ال compilers نترك مهمة عمل ال thread عليه من خلال ٣ طرق :

① Thread pools ② openMP ③ Grand contral Dispatch

ونترك على عاتق المبرمج استخدام ال threads عملية انشاءهم وانهاءهم مش من مهامة

* بالكود السابق كان من مهام المبرمج كل اشي

① Thread pools : ال workers جاهزين وانت بس اعمل ال task وسلمة ياها (collection, group of threads pre-created, the programer use them not to creat them) • اح نحقق ٣ فوائد

Ⓐ أسرع لانه وقت ال creation لا يحسب من ضمن البرنامج

Ⓑ اذا بتستخدم modul one to one ، لو برنامجك بعمل انشاء للكثير من user level thread اح ترهق ال OS انه يعمل Kernel level لكل واحد • لما يكون ال thread اصلاً موجودين انت ما بتقدر تتجاوز هذا الحد بالتالي ما رح ترهق ال OS بعدد كبير من ال Kernel level (لانه في upper limit)

Ⓒ لما تقل المهام إلي المبرمج بعملها بصير أسهل اشغل عليه وبركز على مهامه

الممسوحة ضوئيا بـ CamScanner

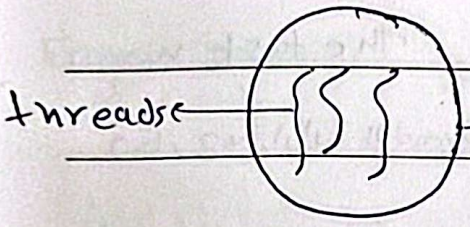
Mo Tu We Th Fr Sa Su

Memo No.

Date / /

Threading Issue

* الموارد المتعلقة بال threads تكون عام



fork() بتعمل child هي نفس الـ parent الـ threads بتجوا الـ child فيه thread واحدة وممكن تعمل thread جديدة واذا احتجت لـ thread أخرى ممكن تستخدم pthread creat او الـ Kernel level thread library

* باما الابن بياخد كل مواصفات الاب او جزء منها

* exec() الـ thread اللي الـ child عملها بياخذ task جديد او بياخذ نفس الـ task تبع الاب

pthread-creat تعطي الـ thread الـ task

Signal Handling:

event happen in OS during execution code of some thread, الـ event هاد بدو الـ thread يرجع error بالاضافة كمان signal بترجع error بترجع action معينة

Signal i error occurred during the execution of the code requested by the thread

بالنسبة للـ Kernel level-thread: الـ thread لما بترجع error بترجع الـ thread طلبت منها تعمل عملية معينة وكان الـ error هاد (error) بتعمل عملية معينة وكان الـ error هاد (error) بتعمل عملية معينة وكان الـ error هاد (error) بت

Memo No. الأحد
Date 3 / 8 / 2025

Thread Cancellation

Cancellation : the thread has not finshed and we are going to stop it, the task no longer needed. (خارج عن ارادة الـ thread)

termination : the thread's task is done, so we don't need the thread anymore.
لانه مدام المهمة خلصت يعني فش اشي تعمله بالتالي احنا مش بحاجة الها

* اح نحكي عن الـ Cancellation : يعني الـ thread لسا عندها task تعملها ومع ذلك رح يجيها أمر انه stop execution بالتالي عنا 2-thread بالموضوع
- thread بدنا نوقفها (target thread)
- thread التي ستقوم بايقافها (perant, kernel)

زي ما قلنا انه الـ perant thread بتقدر تعمل Park لـ child thread

زي لما تبحث عن اشي في محرك البحث ، قبل ما يظهر تيسكر الصفحة

* كيف الـ target thread بتتصرف لما يجيها أمر stop ؟

(1) Asynchronous cancellation : ليست طريقة شائعة بالـ OS ، هي توقف الـ thread على طول وهذا يسبب مخاطر خاصة واذا كانت الـ thread بتشتغل بالـ HW أو بتكتب على مكان معين

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

(2) Deferrd cancellation : طريقة أكثر شيوعاً ، بعماي الطريقة

بيوصل الأمر لل thread انه stop بس هي From risk point of view

ما بتوقف حالاً ، هي بتخلص ال task اذا كان بحاجة

انه تخلصه وعندها بتختار safe point توقف عندها أما اذا كان

ال task فش داعي يتكمل فبتوقف على طول وبتصير شبه

ال Asynchronous .

- ال thread بتقدر تتعامل معهم من خلال ال pthread library

وشفنا زي : creat , join , wait والآن

رح نشوف ال cancel وعملها بشابة ال exit كنتيجة بس cansel

معنا there is apart of the job not finshed yet

Pthread-cancel (tid) ؛ إلي لازم بوقفه

مين إلي لازم ينعميها ؟ إلي بعمل issue لهذا ال command

Modes of cancellation :

mode	state	type
off	Disabled	→
أي thread بتخلي هذا ال Flag off لما تعملها cancellation ما بصير لها cancellation		
متى تتوقف هاي ال thread؟		
اذا غيرنا ال Flag لـ Defferd أو Asynchronous		

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Mode	state	Type
Defferred اذا اجاها أمر انه kill yourself بتستنى لحد ما توصل لـ safe point بحدين بتنهي حالها	Enabled	Deferred
Asynchronous مباشرة بمجرد أنا يأتيها أمر التوقف نتوقف	Enabled	Asynchronous

* Safe place to kill it self named as (cancellation points)

* the Best part of the code to do cancellation when you reach a (system call)

← يعني function وراه أقواس زي: wait() , join()
read() , writ() ليش ضيب؟ لانه عندهم الـ OS هو زي
اح يشتغل اذاً الـ thread وقفت تلقائياً بتالي هو مكان آمن
to kill it self

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

cleanup handler: انك ترجع ال resources اي كانت ال thread
تستعملهم عشان يستفيدوا منهم thread آخرين

Thread-Local Storage:

* أي thread داخل ال process هاي ال thread تحتاج
إلى resources ويا إما بكونوا shared أو privet

* يعني الكود كله لو وزعناه على 2-thread مضافة صار ال code segment
shared

* بما انه العنوان بحكي local يعني الاشياء المخصصة لل thread
ولا تشارك منهم البقية أي ال stack / reg ولكن لاحظ
شكل سلايد 10 كيف ال code, data, Files هي أشياء
مشتركة بين ال 2-thread لذلك لازم تكون حذر وتعملهم sychronization
بالتالي مثلاً ما بنفع 2-thread يقوموا بتعديل ال Data بنفس الوقت

* بنتذكر ال process كانوا يتشاركوا بال shared memory لذلك كان في
synchronization بينهم أي بده يكتب بده يتأكد انه فش حدا غيره بعمل
أي action ولكن مستوى التشارك بين ال process أقل بكثير من ال threads
لانه ال thread تابعين لنفس ال intity أما ال process تقريباً
معزولين بالنظام

الممسوحة ضوئيا بـ CamScanner

Memo No.

Mo Tu We Th Fr Sa Su

Scheduler Activation

Date / /

* this concept is exist when you have n-kernel threads :

multithreading modes : ④ tow-level

① one to one ② many to one ③ many to many

* many kernel : يعني الحالة الثالثة أو الرابعة بصير عنا مصطلح اسمه Scheduler Activation وفكرته كالتالي :

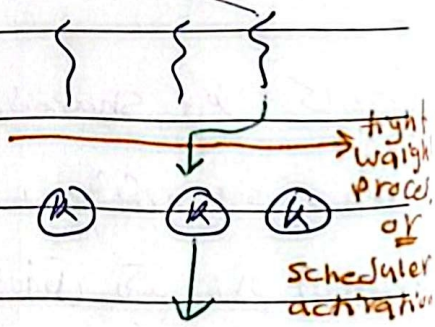
ال thread هاي شغالة وحاجزة ال kernel وقلنا اذا وقفت هاي ال thread ال kernel بعرف انه أحد ال threads وقفت ولكن ما بعرف مين

وبالتالي ما بقدر يساعد ال thread library انه يختار وحدة أخرى

لذلك اجا ال Scheduler Activations

أفرض التالي :

افرض انه هاي ال thread وهي شغالة عملت Block ووقفت وما كملت ال execution بالتالي الاشي اللي لازم يصير look for another one عشان نشغل ال kernel بالتالي نضمن انه ال cpu شغال



نظرا انه ما بعرف انها وقفت ال 2 بضل بنتظرها لحد ما ترجع تشتغل

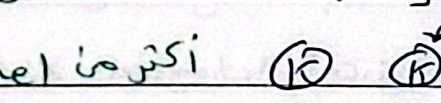
الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

light weight أو ال scheduler activation : هاي كأنها interface
process (LWP)
بين ال usere level إلي فوق وال Kernel-level إلي
تحت ، اذا وقفت هاي ال thread معناته بعطي اشعار thread Blocked
هذا الاشعار بوصل من ال (LWP) لل user library عن طريق
اشي اسمه upcall ومن اسمه واضح اتجاهه :
من ال Kernel ← LWP ← thread library

* لما تخبر ال thread library انه في وحدة وقفت انت وراء الكواليس
كأنك طالب هل في وحدة ثانية اشتغلها

اذا ال K وقفت ما في اي K تاني يشتغل ↓ (K)	عن أكثر من thread ← (Blocked) ممكن تشتغل على أكثر من Kernel (K) (K) (K)



* اذا كان عندك many Kernel level threads ووحدة عملت Block ← you can look For ather Kernel

* هل هذا الاشي دائماً بعطيك فوائد ؟ لا ، لازم يكون عندي ather user thread
حاية تعمل run ومن servers جاهزين

الممسوحة ضوئيا بـ CamScanner

ch 6:

Memo No.

Date / /

Mo Tu We Th Fr Sa Su

CPU Scheduling

* كم scheduler كان بـ chu ؟ ٣

short term ← جزء من الـ Kernel بشيل الـ process من مكان وبحطها بمكان ثاني

long term ←

mid term ←

- لما نحكي cpu scheduling يعني short term بشيل الـ process من الـ ready إلى الـ run

* طيب مين إلي بدك تختارة من الـ ready وتحوله لـ cpu ويصبح running ، اذا أحسنت الاختيار :

① good utilization

② good throughput

③ speed up

④ تستفيد من الـ logical والـ physical cores اعو جودين

- اذا ما أحسنت الاختيار بصير كل ما تحط وحدة تطلع لانها عش مناسبة فبزيد الـ waiting time وبقل الـ utilization وبالتالي بصير أداء الـ OS أسوء

* أي الـ process يعمرها بتقسم ما بين :

① execution on cpu

② waiting for some event

* الـ process أول ما بتلش بتكون execution والـ waiting بيجي لاحقاً ولما تنتهي بتنتهي من execution

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* عمر الـ process بيلش بـ execution وبنتهي بـ execution وبالنص في multipule execution and waiting .

* فترة الـ execution يطلق عليها (cpu burst) يعني وهضة تتنفذ فيها الـ process على الـ cpu

* فترة الانتظار بغض النظر عن سببه يسمى I/O burst

* الـ Scheduling لازم يصير قبل ما يبدأ الـ waiting

* لو بإيدك مين بتعمل أكتر cpu Burst ولا I/O burst ؟ cpu Burst عشان تضل تنفذ على طول ويضل الـ cpu شغال والـ utilization يكون بأعلى الصفيح (الخيار مش بإيدك لانه البرنامج مكتوب وله متطلبات معينة)

* الشكل سلايد 5 : محور x لتبل احصائية عن فترات الـ cpu Burst كم كانت وبعكس هذا الاشي على الـ freq (محور y)

- كأنه عمل دراسة كم بالمتوسط عدد الـ process إلي الـ cpu burst العما بالمتوسط 8 ؟ تقريبًا 9

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* نفس الاشي لما عمل الاحصائية وجد انه عدد ال process اي ال cpu burst الهم 32 تقريبًا 9

* لاحظ باشكل التعداد الأكبر من ال process تفضل يكون ال cpu burst قليل وهون تقريبًا 6 فأقل

* cpu burst is usually small.

* بما انه الوقت التنفيذي قصير معناته رح توقف كثير اذًا contex switch كثير وتدخل ال short term كثير

* ومن هون اجا اسم ال short term أصلاً ؛ أخذ process من ال ready عملها running على أمل انها تطول ولانه ال cpu burst قصير رح تنفذ شوي وتوقف.

short لانه كل شوي لازم يدخل على الخط ويجيب بديل

ال short term ممكن تدخل نتيجة 4 حالات :

① certive process is running converted to waiting

بدنا ندور على وحدة ثانية تيجي مكانها

② switches from running to ready

زي ال interrupt أو time slice خلص

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

③ Switching from waiting to ready

لما يخلص ال i/o event إلي وقفت عشانه بترجع على ال ready

طبعًا ما بترجع على ال running على طول لانه بكون مشغول وفي غيرها

④ Terminates : ال process إلي شغالة خلصت

وبالتالي طلعت من ال running إلى

ال terminated وفضي ال cpu ال short term رح

يشغله بوحدة جديدة

* هدول ال 4 حالات ممكن نقسمهم إلى :

preemptive	non preemptive
إلي وقفناها كان ممكن تكمل زي حالة (2,3) لاحظ أي اشي بتحول لـ ready بكون من هذا النوع لتوقفت غصب عنها time slice	ما كان ممكن تكمل لانه وصلت لمرحلة لازم توقف زي حالة (1,4) توقفت من تلقاء نفسها لانها احتاجت event معين لاحظ أي اشي بتحول لـ waiting أو بخلص بكون من هذا النوع زي i/o event, fork

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

مشاكل اذا قطعت ال process وهي حاية تكمل (Preemptive) :-

① shared data : ال process كانت شغالة بتعمل writing على File أو على shared data كان ممكن تكمل ولكن نتيجة توقف اجباري رح تلاقي الداتا نصها انكتب ونصها لا . حل هاي المشكلة تختار مكان آمن cancellation point ، فاذا اجاها أمر التوقف بتستنى شوي لحد ما تخلص الاشي إي بتعمله

② Preemption while in kernel mode : لما تكون شغالة بال kernel mode ال OS بكون بعرف كل اشي فهو تلقائياً مش غبي انه يعمل preemtion على طول ، بالتالي رح نرجع لنفس النقطة انه لما يخلص ال kernel ال HW access بصير في kill ال process

③ صار في interrupt : لما توقف process معينة في مجموعة من ال command بدهم يصيروا (interrupt subroutine) اذا نفذت جزء منهم ما بنفع توقف now اذا ما خلصت معالجة ال interrupt

- كل المشاكل حلها انه التوقف يكون بمكان آمن وصحيح

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

Dispatcher

* هو الايد اليمين لـ short term schudler (إي هو بختار من الـ ready وبودي على الـ running)

* هذا الحكي يلزمه contex switch إي يتبعه عدة اجراءات :

① بوقف process قديمة ② يختار الجديدة

③ بحمل Reloud لـ process الجديدة

وهذا مش من اختصاص الـ short term (خطوة 1, 3)

هاي من اختصاص الـ dispatcher

* تسريع الـ contex switch عن طريق :

① PCB نحطه بالـ cache حتى يصير storing old value أسرع

والـ Reloading أسرع هيك انت بتسرع الـ dispatcher مش الـ Short term

* لو بدي أسرع الـ short term لازم يكون عندك خيارات أقل بالـ ready

Dispatch latency : كم أخذ ليوقف القديم ويحمل الجديد ويشغله

الممسوحة ضوئيا بـ CamScanner

Memo No.

Mo Tu We Th Fr Sa Su Scheduling criteria (metrics) Date / /

* شو ال factors إلي بدك تهتم فيهم وانت تختار process

① cpu utilization : مش معيار تقييم لانه أي process بتختارها رح تبقي ال cpu شغالة (يزيد أفضل)

② throughput : عدد ال process إلي خلصوا تنفيذ خلال فترة زمنية محددة ، وهذا ما بدخله بالتقييم برضه لانه ال process ممكن تفوت تتنفذ شوي وما تخلص أو process ممكن تتنفذ كثير وبالاخير الناتج هو انهاء وحدة فقط (تشتغل فترة طويلة وما تخلص tasks كثيرة) (يزيد أفضل)

③ Turnaround time : الوقت الزمني من بداية ال process لحد ما تنتهي فأي waiting time بنحسب بال Turnaround فاذا انتظرت كثير هذا ال factor بصير سيئ (يقل)

④ waiting time : كم انتظرت بال ready أما ال (waiting state) لا يحسب (يقل)

⑤ response time : الوقت الزمني لأول مرة بلبثت ال process انها تتنفذ (From creation time to First execution time) من أول ما أنشأت لحظة بدء تنفيذها (يقل)

الممسوحة ضوئيا بـ CamScanner

Algorithem (1)

Memo No. _____

Mo Tu We Th Fr Sa Su

First-Come First Served (FCFS)

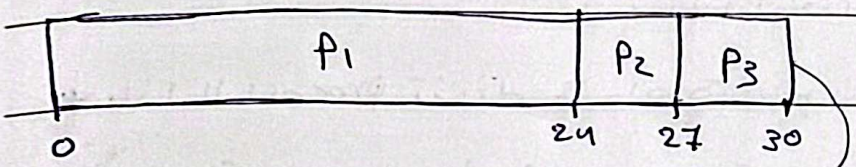
Date / /

Process	Burst Time (cpu)
P_1	24
P_2	3
P_3	3

* عندك ٣ (Process) موجودين بالـ ready queue كل واحد معروف كم وقته على الـ cpu.

* بـ (FCFS) أول واحد بفوت على الـ queue الـ Short term لح يختاره أولاً

* اذا ما أعطاك arrival time (وقت وصول كل وحدة) الـ $P_{\boxed{1}}$ (id) يدل على ترتيب الوصول ولكن اذا في arrival time يصبح هو المقياس وليس الـ id



Gantt Chart ← هاي الرسمة اسمها

* بما انه P_1 أول الواصلين رح نشغلها ونعطيها الـ cpu لمدة 24 time unit

* P_2, P_3 ← ready وينتظروا إلي قبلهم تخلص

* بعدين رح يفوت P_2 لمدة 3 unit time والـ P_3 نفس الاشي

* طبعًا احنا ما بنعرف كم الـ process بدي وقت على الـ cpu ولكن الارقام محفوظة للتوضيح.

* كم الـ utilization ؟ 100% لانه من الثانية 0 - 30 هو شغال وهذا نسبيًا غير منطقي

* الـ throughput = خلصنا 3 خلال 30 ثانية (بنقدرش نعرف اذا منيح أو لا)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

	response time	waiting	turnaround
P1	وصلت وبلشنا فيها عند 0	0	wait + execution 0 + 24 = 24
P2	وصلت عند 0 وبلشنا فيها عند 24	24	24 + 3 = 27
P3	وصلت عند 0 وبلشنا فيها عند 27	27	27 + 3 = 30
		avg = $\frac{0+24+27}{3}$	avg = $\frac{24+27+30}{3}$

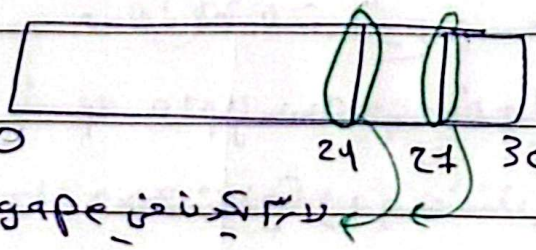
ضروري
عشان اذا شفت واحد منهم ممكن تقرر قرار خطأ

* اذا ال process تنفذ لمرة واحدة بكون ال waiting نفس ال response
أما اذا بتفوت وبتطلع أكثر من مرة ممكن يختلفوا

* هل ال schudling إلي هون preemptive أو nonpreemptive؟
nonpreemptive (لانه ال process دخلت و ضلت لحد ما خلصت بدون مقاطعة) بالتالي هو less complex

* FCFS بالأغلب غير مستخدمة بال OS لانه العمل Bade performance in termes of waiting time

* هون افترضنا وجود (single cpu)



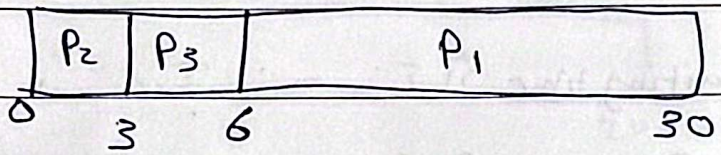
لازم يكون في small gape ال Contexswitch

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* غير مستخدمة للسبب التالي : (اختلاف ترتيب وصول ال process)

		waiting
P_2	3	0
P_3	3	3
P_1	24	6



$$avg = \frac{0+3+6}{3} = 3$$

مقارنة بالترتيب السابق صار في فارق كبير
مع انه نفس المعطيات

* ال execution time لعماي ال Algo يكون أفضل لو بالصدفة ال process القصار اجو بالأول . تعتمد على الصدفة لذلك مش منيحة الحالة الاسوء لو القصار اجو بالاخر

* اسمة convoy effect : cpu bound , i/o bound

P_1 ، P_2 , P_3

طويلة لانه تركيزهم على ال cpu

لانه تركيزهم على i/o بالتالي وقتهم قصير على ال cpu

* بالتالي i/o bound الافضل يجيب بالبداية بعدين ال cpu-bound

الممسوحة ضوئيا بـ CamScanner

Algorithem (2)

Memo No. الاثنين

Date 4 / 8 / 2025

Mo Tu We Th Fr Sa Su

Shortest Job First (SJF)

* لو كان الك الخيار مين بنحط بالأول ؟ الـ shortest

* هاي الـ Algo بتقلك ما بعمني متى وصلوا لانسي رح أحط الأقصر بالبداية.

- اذا عملت هذا الـ action انت بتكون حسنت الـ waiting time لانك لما تجيب الصغيرة بالأول الباقي ما رح يستنوا كتير واذا قل الـ waiting time رح يقل الـ turn around

- لذلك تصبح الـ Algo ← optimal in terms of waiting time, and turn around

- بالتالي هاي الـ Algo ما بتتطلع على الـ response time ، مش مهم متى تبلش اعمم انها ما تطول كتير

process	arrival time	Burst time (cpu)
P_1	0	6
P_2	2	8
P_3	4	7
P_4	5	3

لو أخذنا هذا العمود بعين الاعتبار لما تيجي تعمل schudling at time zero عند P_1 فقط بالتالي الـ schudulen ما عنده خيارات واح يتم تنفيذها لحد وصول وحدة ثانية

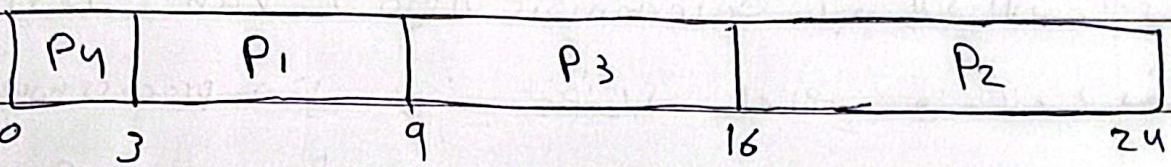
* إي هي P_2 يعني P_1 رح تنفذ 2 unit فقط لحد ما توصل P_2 وساعتها بنقارن مين أقصر من مين (preemptive)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لكن للتبسيط رح نفترض كل ال process وصلوا عند time = 0
وبدك تختار بينهم الاقصر وهو P4 = 3

SJF Scheduling chart



P4	P1	P3	P2

0 3 9 16 24

* non-preemptive (كل مرة اختار الاقصر وخليها تنفذ لحد ما تخلص)

	waiting
P4	0
P1	3
P3	9
P4	16

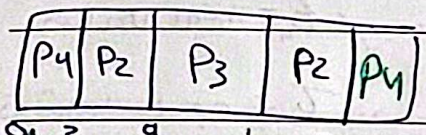
$$Avg = \frac{0+3+9+16}{4} = 7$$

* بالنسبة لل waiting time وعشان يكون ال consept ← general ال process لو كانت موجودة أكتر من مرة بدك تقيس الانتظار في كل مرة

مثلاً :

P4	P2	P3	P2	P4

0 3 9 16 24



waiting = 0 + waiting = 21 = 21

ليش 21 ؟

لانه P4 أول مرة خلصت عند 3
وثاني مرة بلشت عند 24
بالتالي الانتظار 24-3=21

* لما تشوف ال process موجودة بالرسمة مرتين معناته preemptive لانه يكون تاركة قبل ما يخلص

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* كيف نعرف وقت الـ cpu burst ؟

① اما الأرقام هي للأمثلة فقط وغير حقيقة

② أو عملت run لـ determnisic Algo بتقدر لك الوقت التي تحتاجه

الـ process now بمقياس زمني بنظر إلى أعماضي فهو ذا نعمل

(determinig length of Next cpu Burst)

تنبأ بالقادم بالاعتماد على السابق

* هل بتقدر تتوقع أول process كم بدها ؟ لا ، لانه مش واتا قديمة

* معادلة الـ prediction ← $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$

next period

t_n = real value ← على الأغلب ما بكونوا نفس بعضها

توقعك السابق = τ_n ← مش شرط تكون صح

كم بتحسب تعطي = α ← لو $\alpha = 50$ ← للتوقع الحقيقي

نسب لكل من τ , t

$50 = (\alpha - 1)$ ← التوقع السابق

(نسبة التأثر)

$\alpha = 30$ ← الحقيقي

$(\alpha - 1) = 70$ ← توقع سابق

* لاحظت $\alpha = 1$ $\tau_{n+1} = 1(t_n) + \cancel{(1-1)\tau_n}^{0}$ اذاً التوقع للقادم = real value

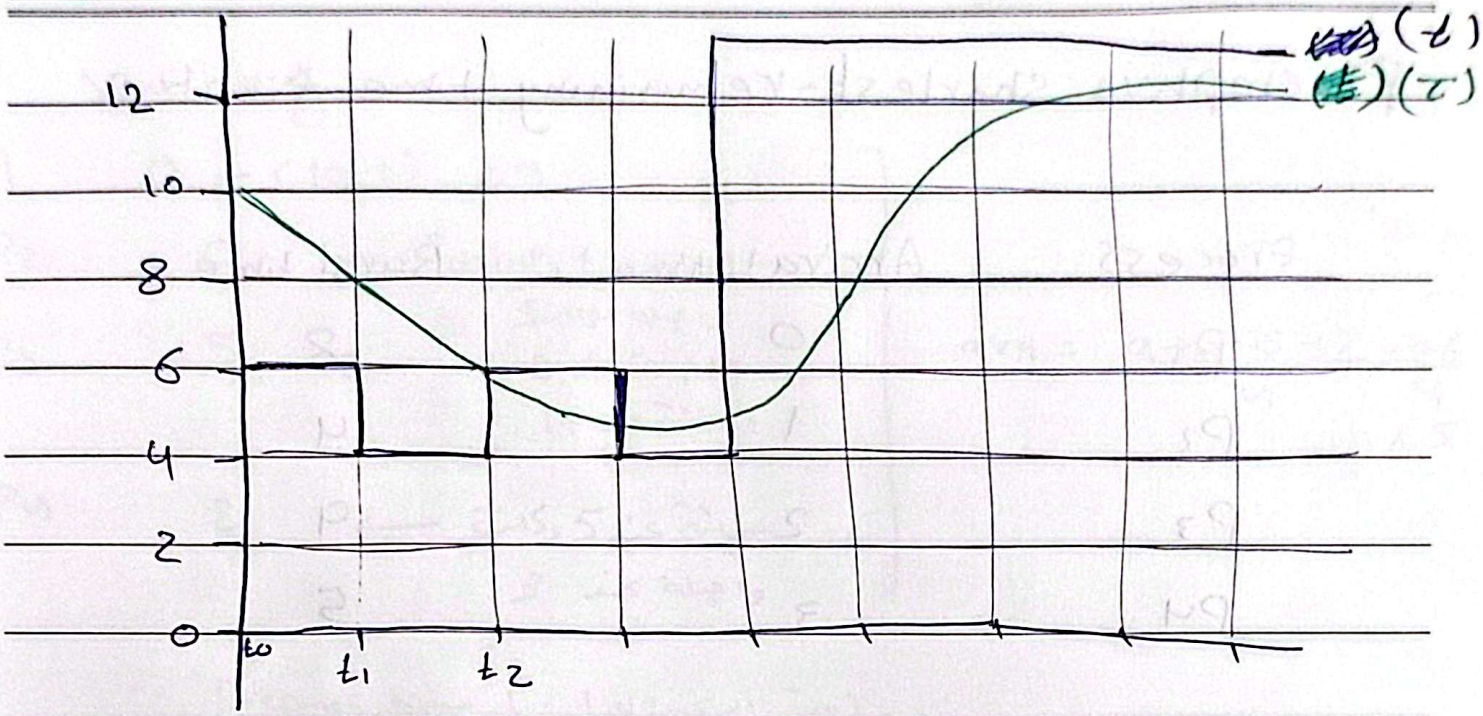
وهو غير منطقي

الممسوحة ضوئيا بـ CamScanner

Memo No. _____

Mo Tu We Th Fr Sa Su

Date / /



cpu burst (ti)	6	4	6	4	13	13	13	...
guess (τi)	10	8	6	6	5	9	11	12 ...

t القيمة الصح

τ القيمة اعتوقعة

عند 0 توقعنا يكون 10 ولكن لما شغلناه طلع أخذ 6 بالتالي ماكان التوقع صح وبينهم error factor = 4

ولكن لاحظ مع مرور الوقت

القيم اعتوقعة تقترب من القيم الحقيقية

$\alpha = 50\%$: جرب على المعادلة

$\alpha t_n + (1-\alpha)\tau_n$

$0.5 \times 6 + (0.5) \times 10$

$a t_0 = 3 + 5 = 8$

$a t_1 = 0.5 \times 4 + 0.5 \times 8 = 6$

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

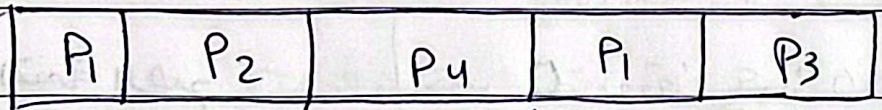
Preemptive shortest-remaining time first :

Process	Arrival time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

* بدنا نتأكد : ① مين واصل ② ومن الواصلين نختار الأقصر
بالتالي من الـ 4-process عند time 0 مين كان واصل؟ P_1

P_1	P_2	P_4	P_1	P_3

Gantt chart



0 1 5 10 17 26

* ما رح نعطي P_1 كل الوقت رح ننفذها لحين وصول process ثانية
اذا شفنا انها أقصر رح نقطع P_1 و نحطها مكانها
بما انه $P_2 = 4$ أقصر من P_1 رح نوقف P_1 وتصير 7 لانها تنفذت
1 time unit

* اثناء عمل P_2 وصلت P_3 ولكن لانها أطول من P_2 ما بنوقف P_2
وأيضا وصلت P_4 وأيضا أطول من P_2 لذلك ما بنوقفها
$P_1 = 7$, $P_3 = 9$ $P_4 = 5$
أقصر

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Waiting

P1 $0 + (10-1) = 9$

P2 0 ← بلشنا على ال 1 وهي أصلاً واصلة على ال 1

P3 15 ← واصلة عند 2 ومبلشة عند 17

P4 2 ← مبلشة عند 5 وكانت واصلة عند 3

$$avg = \frac{9+0+15+2}{4} = \frac{26}{4} = 6.5$$

Respons time (متى استجبتلها)

P1 0 ← وصلت عند 0 واستجبنا عند 0

P2 0 ← وصلت عند 1 واستجبنا عند 1

P3 15 ← وصلت عند 2 واستجبنا عند 17

P4 2 ← وصلت عند 3 واستجبنا عند 5

* Priority Scheduling:

* يوجد رقم يعكس الأولوية وكل ما قل الرقم كلما زادت الأولوية "وبناءً" عليه بنعمل preemptive أو nonpreemptive

تابع →

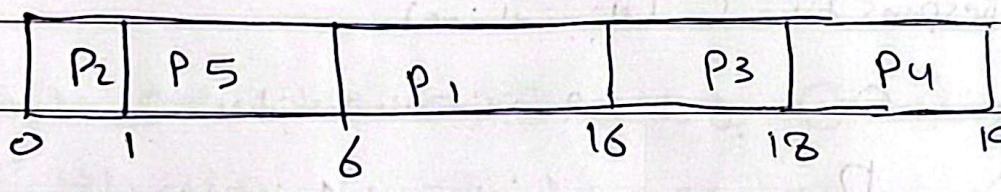
الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Process	Burst time	priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

كلهم وصلوا عند وقت 0 بفارق زمني لا يذكر بدل كلية الترتيب

أعلى أولوية ← عشان أقل رقم



P2	P5	P1	P3	P4

0 1 6 16 18 19

Gantt chart

ال waiting time مرة لكل process لأنه preemptive

	waiting time
P1	6
P2	0
P3	16
P4	18
P5	1

$$avg = \frac{6+0+16+18+1}{5} = 8.2$$

* هذا الاشي ممكن يعمل starvation :

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

مجاعة

* ايش يعني Starvation : افرض انك غيرت أولوية P1
لـ 300 والـ system كل شوي يدخل عليه process جديدة والأولوية
دائما أعلى $P_6 = 20$ ، $P_7 = 10$ ... (كل ما كبر الرقم كل ما قلت الأولوية)
(هاي الـ process : 300 متى رح يجي دورها بالتنفيذ ؟)
متأخر كثير أو يمكن ما يجي أصلاً بالتالي هو (event not happend)

الحل : كل 10 ثواني قلل الرقم بـ 5 بعد فترة زمنية رح تعلى
الأولوية مع تنقيص الرقم وبالتالي رح يجيها دور بالتنفيذ

Aging : تزداد الأولوية مع طرح الرقم

Round Robin (RR) Algorithem :

هاي تجيب عن سؤال ليش الـ process ممكن توقف لوحدها بدون ما يصير
interrupt أو child أو i/o requst .

* بقلك في timer وما بعمني الـ process كم بدها ، رح أطلبش
تنفيذ الـ process بالترتيب ورح أعطي فرضاً 2 ثانية لكل وحدة
اذا خلصت P كان به ما خلصت علاي بتركها وبروح على أي بعدها
وتعتبر هاي الـ Algo الأكثر عدالة لانه ما بصير فيها (Starvation)
* لا يوجد معيار للاختيار سوى العدالة وتستند على اختيار الـ Timer
(Time Quantum)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

Process	Burst time
P1	24
P2	3
P3	3

time Quantum = 4

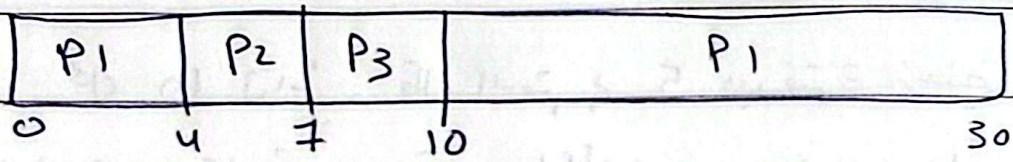
الحد الأعلى إلي 2

أعطيه لل process

(بعطي كل واحد أقل من على قد حاجته)

P1	P2	P3	P1

0 4 7 10 30



Waiting time

P1	0 + (10-4) = 6
P2	4
P3	7

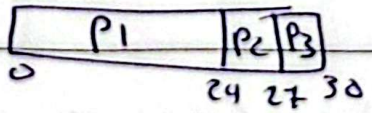
ال critical point بال Algo

هي هو اختيار ال time Quantum

* تخيل لو تختار ال مثلاً 50

بصير زي مبدأ FCFS إلي بوصل أول بنخدمة

P1	P2	P3



0 24 27 30

* تخيل لو اخترنا 1 (كثير قليل) : بضله يقلب process بالتالي بزيد

ال contex switch وبخلي اعوضوع أسوء

* n- process (عدد) , time q

each process cpu time = $\frac{1}{n}$, max waiting time = $(n-1)q$

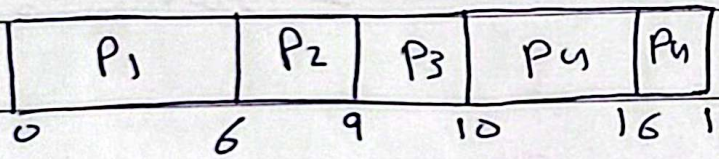
الممسوحة ضوئيا بـ CamScanner

Memo No. الثلاثاء
Date 5 / 8 / 2025

* سلايد 23 : لو أخذنا عند الـ 6 ، وبالنسبة لـ process
رح نبلش فيهم بالترتيب لانه فش arrival time

process	time
P_1	6
P_2	3
P_3	1
P_4	7

Time quantum = 6



P_1	P_2	P_3	P_4	P_4

0 6 9 10 16 17

Turnaround = متر خلصت - متر وصلت

P_1	6-0	= 6
P_2	9	
P_3	10	
P_4	17	

$$\frac{6+9+10+17}{4} = 10.5$$

روح على الشكل رح تلاقيها

* من الشكل بقلك أنسب خيار لـ Quantum هو إما 6 أو 7 لانه بعطيك أعلى turnaround أما أسوء خيار هو 5 لانه بعطيك أعلى turnaround

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

ميزات كل Algo :

① FCFS : الـ response time لأول وحدة بتوصل بكون قليل

② SJF : بختار الأقصر بالبداية بالتالي بقلل الـ waiting time وبالتالي الـ turnaround أقل شيء

③ Priority : الأولويتين أعلى بتخلص بسرعة

④ Round Robin : عادل بين كل الـ process لأنه بعطي وقت محدد لكل وحدة response time لكل بكون مبيح لأنه بعطي دور

* فلتر واحد أحسن من الثاني *

• لو كان عندك many process بالنظام مين الخيار الأحسن ؟

hybrid model (اخلط بيلعم)

عدل ← RR

وقت انتظار قصير ← SJF

* ممكن تصبح الـ ready queue أكثر من وحدة : سلايد 25

إلي اهم أعلى أولوية ، ح يدخلوا على أول queue

وأقل أولوية يتوزعوا على إلي تحت

* كل وحدة من الـ queues بطبق عليها Algo مختلف

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

ايش الفوائد	ايش المساوئ	
① إلي جاب تخلصه بسرعة	① high complexity	رح نعط timer مجرد ما يخلص
رح يخلص بسرعة	② starvation	رح نطرح من الرقم عشان نرفع
	للي إلهم أولوية قليلة	الأولوية وهذا اسمه

feedback انن ترفع الأولوية

وتنقل ال process من ال queue

إلي تحت ال queue إلي فوق

* تصنيف multilevel بحد أدنى ممكن نستخدم :

① Foreground queue	② Background queue
الأعلى أهمية ، طبق عليها	الأقل أهمية ، طبق عليها
preemptive , Round Roben	non preemptive , FCFS

* لو بدك تشغل ال scheduling ساعة رح نعطي 80% للأول

و 20% للثانية

min response time

وبينهم رح نضمن ما يصير starvation

garantee of arrival time

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

سلايد 27 : 3-ready queues

لو فرضت عندك 4-process

وأعطيناكل وحدة 8 time unite

في جزء منهم رح يخلص

وفي جزء لا لانه ال Burst time

أعلى من 8 لذلك رح

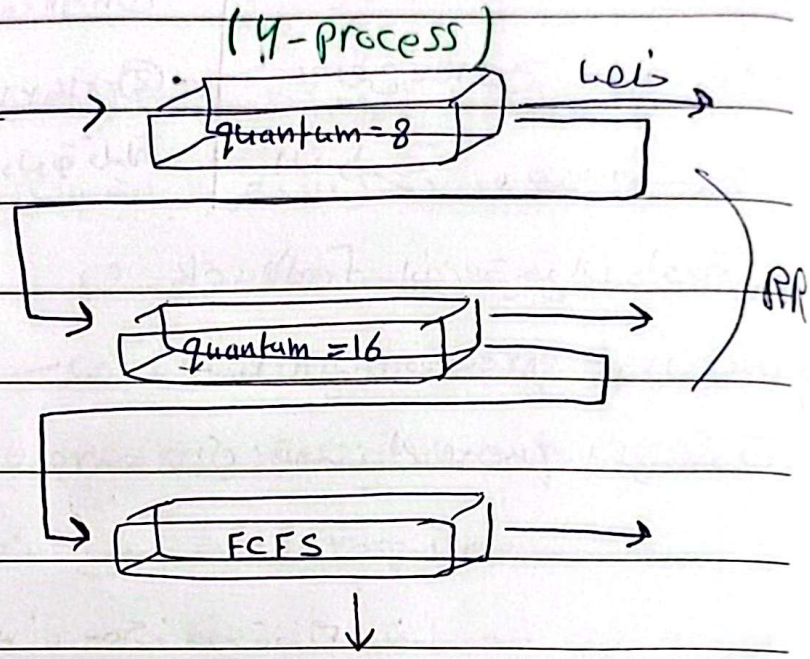
تنزله على ال queue الثانية

ونعطيهم 16 timeunit

في جزء رح يخلص وفي جزء

لسا وقته أكبر وما خلص بالتالي

بروح على آخر queue



هاي ال queue إلي

بفرت عليها لازم تخلص

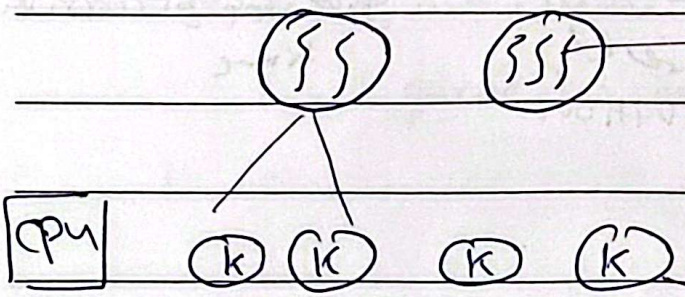
لانها تبعطي ال process الوقت إلي بدها

ياه على ال cpu

* هون بفترض وجود single cpu بكل ال Algo

* Thread scheduling :

threads تنقسم حسب اعالك إلى : Kernel , user



ال threads إلي جوا تتنافس

عشان تاخد Kernel (تنافس على مستوى

ال process) process contention

بتنافسوا على ال cpu

system, resources contention

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

صح ، خطأ :

[1] user libirary thread the type of computision is process computision : T

• انت كمبرمج الاشي إلي بتقدر تتحكم فيه هو : process contention

• أما الـ system contention مبني داخل الـ os بالتالي ما بقدر أتحكم فيه

* ليش في تنافس : لانه عدد الـ thread أكبر بكثير من عدد الـ resource

Pthread library : تتيحلك تعمل creation ، تمشي الـ thread أو توقفها
وأيضاً رح تقدر تعملها scheduling

كود سلايد 50 : Pthread-t tid [Num Thread]
→ creat array of threads
ومن الكود تبعرف انهم (5)
واسمهم ... tid0, tid1, tid2 ونوعهم (user level)

خلي الـ defut attribute
أي ما هي (مساحة x mem
(# of reg , PC

Pthread-attr-t attr;
Pthread-attr-init (&attr);

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

if(pthread_attr getscope (&attr, &scope) != 0)

Process contention — system contention

اذا رجعت 0
معناته كله تمام اما
اذا رقم غير الصفر
يكون في مشكلة في
هاي ال thread من ناحية
ال scheduling أو ال scope

if (scope == pthread_scope_process)
printf ...
process contention
user level

else if (scope == pthread_scope_system)
printf ...
system contention
kernel level

* ليش بسترجع هاي المعلومة و بطبعها ؟ لانه اذا كانت
kernel فش مجال تعمل عليها أي اجراء (اذا كانت user أو حولتها
إلى user) بقدر أعدل عليها

تكمله الكود سلايد 31 ؛ في تعديل

Pthread_attr_setscope(&attr, Pthread_scope_system)
let this thread to be kernel / و إلي بعملها scheduling هو الـ os

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

• حسب معالم الكود احن عن readyqueue فيها 5 ؟

وعندي ال thread الأصلية

for(i=0; i< Num_thread ; i++)

إي عملت ال 5

pthread_join(tid[i], Null) ; → مهمة ودنه لدال perant خلص رح يصير بكل ال thread termination

tid[0]
tid[1]
tid[2]
tid[3]
tid[4]

Perant مارح يتحرك لحتى ال 5 يوصلوا

(داخل ال Forloop)

* Multiple-Processor Scheduling : كنا نفترض وجود cpu واحد ، طيب اذا ما كانوا

① Homogeneous : (cpu من نفس النوع) واحد ممكن يكون نوعهم كالتالي

② Asymmetric : (واحد بعمل task معينة الثنايتين ما بعرفوا بعملوها زي كرتبو)

③ symmetric multiprocesser : (أكثر من cpu من نفس النوع) مشبه النوع الأول

بتقدر تخلي ح P ما تعمل شغلها إلا على cpu 1

وهذا اشي اسمه affinity

(A) (P2)

cpu

(من غير اعتطف اذا كانوا ال cpu نفس بعض انك تحدد هذا الاشي)

الممسوحة ضوئيا بـ CamScanner

Memo No. الاربعاء
Date 6 / 8 /2025

affinity
- soft : انه هاي الـ process يا حبذا لو تروح على cpu معين ولكن ما بتقدر تجبر النظام
- hard : انه هاي الـ process لازم تروح على cpu معين

** migration (الهجرة) : طريقة حل مشكلة وجود جميع الـ process عند cpu واحد والثاني فاضي

الحلول :

1. load balancing : تقسيم الـ process على الـ 2-cpu
2. push migration : انك تودي الـ process من إلي عنده كتير إلى إلي عنده قليل (المضغوط هو إلي بودي)
3. pull migration : سحب الـ process من المكان اعليان للمكان الفاضي وإلي بعملة هو الـ cpu الفاضي

هدول إلي حكينا عنهم لا يتطبقوا اذا كان عندك 1-cpu

الممسوحة ضوئيا بـ CamScanner

Memo No.

Date / /

Real time cpu scheduling

* ال process إلي لازم تنفذها معاها وقت لازم تخلص فيه
وبالتالي اذا طولت ممكن تعمل Kill لنفسها لانها No longer needed
(dead time) متى لازم تخلص

* تخيلي اعو ضوع زي: الامتحان مدته ساعة، انت خلصت بعد ساعة اذاً
لا قيمة لشغلك

* مثلاً 30 = Burst time وال dead line = 40 → أعلى عشان
اذا صار في انتظار
فيب لو خلصت على ال 45 أو 50 رح نعتبرها
unsuccessful وما بندعدها من ضمن إلي خلصوا

* after the dead line finshed, the process is not include in throughput.

أنواع ال Real time :

① Soft : لو خلصت بعد الوقت المطلوب بشوي مش مشكلة
(التأخير مسموح قليلاً)

② Hard : لازم تخلص على الوقت بالزبط (التأخير غير مسموح)
بتحط load أكبر على ال os

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

نوعين من ال delay بأثروا على الأداء :

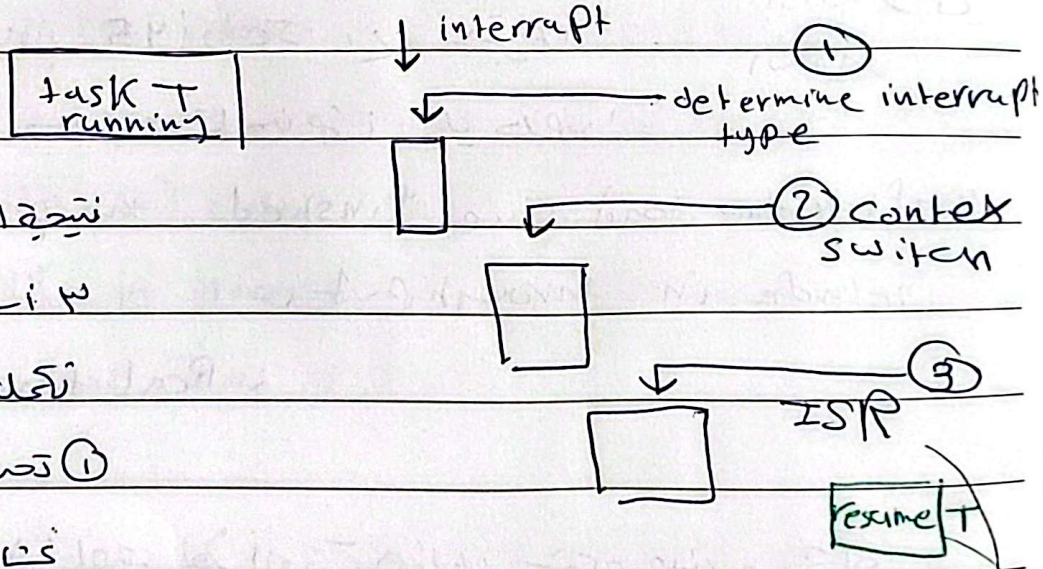
① أي شغل مش من ضمن ال execution بصير يحط ال process real time على المحك

أي نوع من أنواع التأخير ممكن يخليها ما تلحق تخلصها

زي ال interrupt delay

② Dispatch latency : وقت جزئي من ال context switch (الأفضل لو يقل

عشان ال latency تقل عشان real time process يلحقوا يخلصوا)



نتيجة ال interrupt ال 2 يصير

٣ أشياء لحد ما نرجع

نكمل شغل على ال process

① تحديد نوع ال task الفرعي

عشان تعرف نوع ال action ; ي ح

توخذه

② context switch

بوقف القديمة وبخزن النواتج

ومكان ما وصلنا بالتنفيذ

③ ISR (interrupt service routine)

هدول ال ٣ سببهم ال interrupt

نفذ ال task

الفرعي يعني بعمل interrupt

هدول ال ٣ كلهم delay

ارجع نفذ T ، مع كل هذا التدخل هل

ليس T قادرة انها تخلص قبل ال deadline

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ال OS ممكن يكون ذكي ويأجل ال interrupt اذا شاف انه ال process قرب ال deadline انها وخلص وبعدين يرجع يعالج ال interrupt

* ال scheduling Algorithm إي بتزبط مع ال real time process

1. Priority based scheduling
2. Rate Montonic scheduling
3. earliest deadline first scheduling

1. Priority based scheduling

* رقم الأولوية بكون أكبر لل process التي تحتاج ال cpu أكثر

* Period P = الوقت الذي يعطى لل process ويتكرر زي ما هو (كل كم بتتكرر)

* d = deadline الوقت المفروض ال process ما تتجاوزه

* Burst / execution time = t

* كم بتحتاج period P = freq ال cpu هل بتردعالي أو تطيل وبتدلك هل هي i/o bound أو cpu bound

i/o bound: بتحتاج ال cpu كتير لانها بتوقف كل شوي تتردد كتير

cpu bound: بتاخد ال cpu لمدة طويلة بعدين بتخلص

* لازم تكون $0 \le t \le d \le P$

اذا ال $t > d$ معناته خلص بعد ال deadline وشغله عالفاضي

الممسوحة ضوئيا بـ CamScanner

Memo No.

Mo Tu We Th Fr Sa Su

Ch5: Process Synchronization

Date / /

* process synchronization : هي حل لمشكلة بتصير بال OS عنوان هاي المشكلة هو ال Data inconsistency

* بال OS ال resources الموجودة (ال HW) على الأكيد انها shared بين كل ال processes وعملية المشاركة بتخلي في منافسة على ال resources .

* بالتالي عدد كبير من ال processes يتنافسوا على memory وحدة مثلاً أو على ملفات معينة أو على reg أو i/o devices بالتالي عملية التنافس اذا لم تتم بشكل صحيح رح تؤدي لمشكلة اسمها Data inconsistency

مثال : لو افترضت عندك 3-processes أو أكثر بتنافسوا انهم يكتبوا معلومة على ال memory (في location بال memory لنحكي انه ال address الها [10] وفي 3-processes بدهم يتنافسوا على امكان عشان يكتبوا عليه ، القيمة إي رح تكون موجودة بالاخر يمكن ما تكون منطقية)

* مثلاً التعديل على قيمة counter وهذا ال counter مشترك بين أكثر من process ممكن بالنهاية يعطيك ناتج غير صحيح لو كانت العملية محكومة بـ (Race condition)

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

* لو تنافس أكثر من process على ال resourcese (وهو أمر مؤكد) رح يصير في Race condition يعني الناتج النهائي رح يعتمد على ال process الاسرع

- اذا كانت النتيجة النهائية منطقية بعد حدوث ال Race condition بالتالي ما عنا مشاكل

- اذا كانت القيمة غير منطقية معناته صار عندك Data inconsistency (يعني انه القيمة النهائية لا تدل انه العمليات تمت بشكل صحيح)

* حل هاي المشكلة بكون عن طريق ال process synchronization

وكلمة synchronize بال computer معناها to arrange the events يعني ما تترك اعنافة وإي بسبق بنفذ لذلك رح يتدخل المبرمج وال OS لحتى يلغوا عملية اعنافة مؤقتاً عن طريق اصدار قرار مبين ال process إلي رح تتنفذ أولاً ومبين ثانية وهيك يتم الغاء ال Data inconsistency

* لما نعمل Rearang يعني بنخلي ممنوع 2-processes يتنافسوا على نفس ال resource بنفس الوقت

* ايش سبب مشكلة ال Data inconsistency ؟

① عدد ال resources أقل من عدد ال processes ، ممكن تزيد عدد ال resources وهو غير منطقي

② انه ال OS ممكن يشتغل بـ preemption mode (تخيل في process قاعدة بتكتب على ال mem وتروح توقفها بالتالي الداتا المكتوبة بطلت مفهومة لانها غير مكتملة وبالتالي غير صحيح)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* حل المشكلة رقم (2) انه تخلي الـ OS يشتغل بـ nonpreemptionmode وهو
حل يحد من كفاءة الـ OS ومن كفاءة الـ utilization
بالتالي هذا الحل أصعب من ترك مشكلة الـ synchronization قائمة

* الحلين السابقين غير منطقيين لذلك الحل إلي رح نستخدمه هو
الـ process synchronization والغاء الـ Race condition

* أي كود بنشوفه بهذا الشابتر رح يحتوي على مكونين

do {

entry section

(مش معلم كم حجمة)

كود يتم فيه التعديل على متغير (resource) (1) critical section (خطر)

مشترك واسمه critical

exit section

(خطر) لانه ممنوع يتم (2) remainder section

التعديل من أكثر من

} while (true);

process : التعديل ممكن

يكون كتابة على ملف أو

reg أو mem location

أو تغير status (HW)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لذلك لازم نغلف الـ critical section بحماية عشان نمنع 2-processes
يعدلوا على المتغير المشترك مع بعض لذلك دايماً هو محمي ببوابة دخول
(entery)
وبوابة خروج (exit)

* وعلى عكس الـ critical من اسمي اسمه remainder section
هو كود ولكن لا يحتوي على أي update لمتغير مشترك لذلك ما في داعي
تكون حذر معاه.

مثال: لو في 2-processes بدهم يحملوا (print out) جملة طباعة على الشاشة
لو الثنين بعثوا القيمة بنفس اللحظة على الشاشة لا يمكنك توقع
اسمه رح يصير، ممكن يطلع الثنين صح، وممكن يطلع الثنين خلط

معناته في وحدة
سبقت الثانية

لو سرعتهم فرضاً
طلعت متساوية
ممكن يصير في خلل
بالـ output

* اذاً بالتالي الـ 2-process ينطبق عليهم
مفهوم الـ critical section

* أوامر القراءة هي (remainder) لانه يمكن تطبيقها in parallel
* أما أوامر الكتابة (critical section) يصير في مشكلة اذا تم تطبيقها in parallel

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

entry section : هو رح يضمن تنفيذ ال critical section ← in serial

كيف ؟؟

لنفرض وجود أكثر من process بحاولوا يوصلوا ال critical section بنفس الوقت و entry section لازم يحمي ال critical عن طريق انه يسمح لـ process وحدة توصل لل critical والباقي يستنوا (Block) (waiting) status حيك بكون صار التنفيذ in serial .

* ليش بدي (exit section) : عشان إي بستنوا برة اذا ضلوا بنفس الوضع رح يصير لهم starvation لانه ما بنفع تخليهم يستنوا للأبد . لذلك وظيفة ال exit التأكد من انه إي بستنوا بره رح يجيهم دور بعد ما إي داخل ال critical وخلص .

* ال (exit) بعطي اشعار لل process إي بستنوا انه بدنا نختار واحد منكم لانه إي داخل ال critical خلص

* كود سلايد (9) هو standerd للحل وليس الحل نفسه entry انت رح تكتبه وال exit انت رح تكتبه : الاجزاء إي المبرمج رح يكتبها رح تتقيم من خلال ٣ معايير .

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

(single accesse)

① Mutual Exclusion : يقيم ال entry section وهاي خاصية

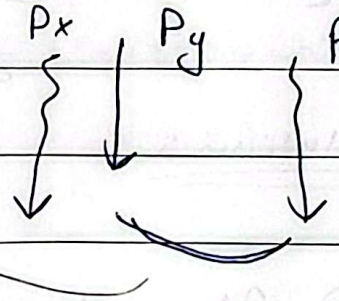
بتحكي انها موجودة اذا ما في 2-processes ولا بأي حال من الأحوال

اح تقدر توصل ال critical يعني انت لغيت ال ~~competition~~ race condition

ولغيت ال accesse على ال critical section

② Progress : في حال خلص أحد ال process من ال critical

افرض Px خلصت



لازم واحد من إلي بستنوا هو إلي بده يحكيلنا

مين إلي رح يفوت Next

* يعني ما يجي Pw مش مهتمة بال critical هي إلي تحدد

مين إلي رح يفوت على ال critical بعدين

③ Bounded waiting : في حال انتظر أحد ال processes في محاولة

الوصول إلى ال critical section زمن معين ، لازم هذا الزمن

بالآخر يخلص (يعني وقت الانتظار ما يكون لانهائي) .

يعني ما يقعد يستنى وما يجي دورة ، طول الفترة مش مهم

المهم انها تكون محدودة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* اذا عملنا هاي الاجراءات كلها بنضمن انه ال critical تنفيذه serial مش parallel ، ساعتها بعني سرعة ال process لانه انت لعينت ال Race condition يعني ماني ثنين رح يتسابقوا ويخربوا على بعض

* دايماً كنا نستخدم ال join وال wait عشان نلغي عامل (الاسرع بسبق) !! واستخدامهم يضمن انه الكود يتنفذ بالترتيب الصحيح

Solutions

① Algorithm for process P_i

```
do {
    while(turn == j);
        critical section
    turn = j;
        remainder section
} while(true);
```

* حل يأخذ بعين الاعتبار وجود 2-processes فقط

* المشكلة فيه انه خاصية ال progress غير متحققة فيه

* الحل الي بخلي هذا الخيار منيح هو تعديله لهكل التالي

* بال process synchronization لازم نفكر بكل ال process الموجودين عشان نفهم الحل.

الممسوحة ضوئيا بـ CamScanner

Peterson's Solution

Memo No. _____

Date / /

Pi	Pj
do {	do {
flag[i] = true;	flag[i] = true;
entry turn = j;	turn = i;
while(flag[j] && turn == j);	while(flag[i] && turn == i);
}	}
critical section.	critical section
flag[i] = false; exit	flag[j] = false;
remainder section	remaindersection
} while(true);	} while(true);

* الكود هنا يقدم في حال وجود 2-processes only

* الكود هي فوت شغال بطريقة تمنع دخول اكثر من 2-process لـ critical في نفس اللحظة

* وهاي بصير اذا حاول اكثر من 2 يدخلوا بنفس الوقت

* يعني شغلنا Pi و Pj لنبلغ concurrently بحاول يدخلا لـ critical

* وهاي بصير اذا حاول اكثر من 2 يدخلوا لـ critical بنفس الوقت

Memo No.
Date / /

* توضيحات بتخص الكود:

① عندي 2-Flag
- الأول اسمه (i) خاص بـ P_i وقيمته bolean
- Flag[i] قيمهم ممكن true أو False
- والثاني اسمه (j) وخاص بـ P_j وبرضه bolean

• دائماً بالـ critical section رح تلاحظ انه اغتفرات هي bolean لانه عندي حالتين فقط true أو False يعني واحد بحتي وواحد يوقف فعمل (most suitable data type to use in critical)

② اذا كان Flag[i] مختلف عن Flag[j] معناته قيمهم غير مشتركة أما بالنسبة لـ turn هي متغير مشترك (static variable)

③ أي process بتحاول توصل لـ critical section رح تعمل التالي:
- P_i مثلاً عملت الـ Flag[i]=true هاي الجملة معناها:
I am process[i] and I am intersted interring the critical section
بالتالي هذا الـ Flag يعني عمل الـ process مهمته توصل الـ critical أو لا
- بالكودين الصفحة السابقة رح تلاحظ P_i, P_j اللذين true يعني both are intersted

* اذا انت intersted لا يعني الك امكانية الوصول على الـ critical

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* رح نبلش تنفيذ بكود Pi , Pj لنشوف اذا الثنين وصلوا ال critical

بتكون هاي ال Alg عديمة الفائدة لانها عاملت وصول حصري لل critical

Pi	Pj
turn = j ;	turn = i ;

* رح نلاقي بالكود سطر

هذا بقلل مثلاً : Pi صح اني interested عشان i وصل ال critical

بس مش مشكلة بتقدر تعطي الدور لـ (j)

و Pj بتحكي صح اني مهتمة i وصل ال critical ولكن بتقدر تعطي الدور لـ (i)

* الآن مهما تفكر بالتنفيذ مارح تكون قيمة ال turn تساوي i و j بنفس الوقت

يعني ال turn = i

- اذا process(i) سبق process(j) رح يصير التالي :-

Pi رح تحكيلي خلي ال turn = j ورح تيجي تشتغل على جملة ال while

while (flag[j] && turn == j) ;

اوجد على اوجد على كود Pj رح تلاقي قيمتها = ture

كود Pj رح تلاقي قيمتها turn = i بالتالي i ≠ j معناته Pi رح تتخطى هذا

الشروط وتوصل ال critical

أما Pj : لما توصل جملة ال while ← { } empty loop while (flag[i] && turn == i)

بما انه Pi سبقت من كود ال Pi رح

Pj معناته قيمة i تلاقيه true → i == i

يعني Pj رح تعلق

داخل empty loop ومارح

توصل ال ...

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

while(flag[i] && turn == i) هاي ال loop جداً مهمة في

ال synchronization وبتعطينا اشي اسمه Busy waiting أو spin lock

ومعناهم انه ال process مشغولة في انها بتعمل ولا اشي لانها في emptyloop

* إذا هل وقتها مستغل ؟؟ لأ ، بس هاي Busy بتضل تشيك على اشرط

وترجع تفوت على emptyloop تعمل ولا اشي

* فائدة ال while باختصار : تأخير الثاني لان الأول استطاع الدخول

* بالكود السابق لو كان ال turn = j ال Pi رح يعلق وال Pj رح يوصل critical

* لو توقف الكود على ما هو عليه رح نحكي عنه

* This Alg shows clear implementation of Mutual

ولكن باقي الخصائص تحتاج إلى دراسة ، execlution

- قلنا انه خاصية ال mutual تقاس من ال entergsection ارجع للكود

الجزء إلي محطوط عليه مربع ، كيف عرفنا ؟؟ لانه بحمي ال criticalsection

* لو بقيت Pi داخل ال critical و Pj بال Busy waiting رح يصير في starvation

لـ Pj بالتالي الحل اعقدم ساعتها من ناحية ال process sychronization بكون فعال

ولكن العدل وال effecincy وال starvation مش شغال

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

لذلك تكملة هذا الحل رح تكون بال exit section ، لما تخلص ال Pi من شغلها سواء طويل أو قصير هذا يعتمد على شرط ال critical وشو هو ال shared resource إلي بعملوا access عليه ، رح تعطي اشعار لل process التانية انه : I am done you can proceed

• وهذا الاشعار يكون عن طريق :

while(flag[i] && turn == j) جعل أحدهما False

لسه خاص ب Pj

خلي واحد منهم false

* لانه بالاساس هي معلقة داخل emptyloop لانهم الاثنين true

* بالتالي بتخلي flag[i]=false وهذا معناه : Pi is no longer intersted وبنفس الوقت بتكون Pj بتشيك على شرط ال while وبالتالي ببطل صحيح وبتبطل معلقة بال Busy waiting وبتوصل ال critical

* دائما بالبداية بكون flag[i]=flag[j]=false

* طريقة اخرى لتنفيذ الكود افرض انه واحد مهتم والتاني لا وشيك عن طريق ال (flag)

* هذا الكود بحقق ال mutual exeduty وال progress وال Bounded waiting

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* Is the previose Alg is a perfect solution ??

No,

① In OS you can not just have only Tow processes.

الخوارزمية السابقة لـ 2-process بس لهذا هي ضعيفة جدًا وبالانظمة الحديثة عندي كتير processes

② If 2-processes try to modify the (turn) in the same time

بالاخير كانه صار الـ turn متغير مشترك وكانه رجعت مشكلة الـ Data inconsistancy بالتالي هاي الـ Alg عديمة المنفعة.

* هذا الحل هو SW تمامًا فش تدخل لـ HW ولسا في مشكلة الـ Race condition

* ح ننتقل لحل مبني على الـ HW Synchronization hardware:

ح يكون عندنا 2 or more HW instr. (privilege inst) أو Atomic

* هاي الـ instrection بتصير زي جزء من الـ Assemply code تبع الـ cpu يعني ضمن instrection set arch.. (ISA) مجموعة الـ opcode إلي بشغلوا الـ cpu بالتالي الـ HW. instrection ممكن نسميها low level instrection

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

HW instrection : instrections directlly related to HW and directly executed on HW without compiler بدون اسمي وحولها.

* رح نضيف 2-instrections من ضمن ال JSA والهدف منها أعمل Synchronization.

طيب ليس ما استخدام اعوجدين زي ذ ل = turn ؟؟

① شكلها لا يوحي انها low level ، لان ال low level لا يوجد فيه samples وهون عندك (=) بالتالي هاي high level ولازم تتحول ل assemply واذا حولناها ممكن تصير سطرين أو ٣ أسطر واذا أثناء التنفيذ تم مقاطعتها - رح نرجع لمشكلة ال Data inconsistancy

② هاي ال instrection ليست Atomic : يعني non-interruptible + executed serially . مهما كانت أولوية ال instrection القادمة (الجديدة) لا يتم مقاطعة القديمة أبدًا وأيضًا هذه ال instrection ممنوع تتنفذ حال 2-cpu بنفس الوقت ، لازم تتنفذ وحيدة بال HW

* إذًا بال (Synchronization Hardware) : معناه زودني و low level instrection - ممنوع توقف اذا بدأت التنفيذ وممنوع تنفذ منها أكثر من نسخة في ال computer ككل . لازم نسخة واحدة وواحدة فقط

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

2- Atomic instrections :

① test-and-set ② compare-and-swap

* ليش عنا Atomic ؟؟

لانه لما نحكي انه non-interruptible معناته اشرط الثاني أصلاح غير متحقق وكانه ال OS شغال بـ non-preemption mode وبالتالي سبب حدوث ال Data inconsistancy غير قائم

وتنفيذ نسخة واحدة فقط وهذا يعني وجود process وحدة شغالة

* وجود ال Atomic تلغي وجود ال data inconsistancy وال Race condition

* اذا جهازك في cpu واحد مشكلة ال data inconsistancy بتصير لو ما لغين ال interrupt يعني في (preemption)

* اذا جهازك فيه أكثر من cpu بكون مسبب ال data inconsistancy له مرجعين : preemption من جهة ووجود أكثر من process من جهة ثانية بتحاول توصل ال critical

وجود multiplal cpu's يعني احتمالية وجود أكثر من process شغالة بنفس الوقت فهون حتى لو عملت disable لل interrupt هو غير كافي لانه لسا عندك أكثر من وحدة شغالة بنفس الوقت

وجود uniprocesser بس مش عامل disable لل interrupt ساعتها عمكن يصير preemption تؤدي لـ data inconsistancy

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* أي Atomic inst الكود تبعهما من الداخل بحاول يعمل lock لل critical

أي كود سلايد 17 : الجزئية إي بتعمل lock بنحطها قبل ال critical
في ال entry والجزئية إي بتعمل unlock بنحطها بال
exit section .

* تخيل عندك many process بحاولوا يوصلوا لل critical section ال Atomic
بتفيد بال entry كأنها بتسكر الباب ؛ بتفتح الباب لـ process
وحدة وبتسكر عالباقي بالتالي بتحول التنفيذ من parallel إلى serial

* Test-and-set :

boolean test-and-set (boolean * target)
{ بخزن فيه قيمة ال lock
boolean rv = * target ; (temperary value)
* target = TRUE ;
return rv ;
}

ماضي : 0
في : 1
قيمة ال lock

افرض قيمة ال lock = 0 واجتت
process محظوظة وباشرت التنفيذ
لأنه ال critical مفتوح

① بحسب ال 0 وبخزنها بال rv
② بروح بجعل ال lock = true
(كأنك سكرت الباب وراك)
③ وبرجع ال old value وهي (0)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

افترض على الكرد السابق وجود P_2 و P_3 لما P_2 يجي بعدين

رح يلاقي قيمة ال lock=target = 1 وتعني انه ال critical is locked

فما رح يقدر يوصل ال critical لحتى P_1

تخلص وتعطي اشعار.

• كيف نستخدم ال (test-and-set) : كل الكرد السابق بدي

أحطه بال entry section

```
do {
    while (test-and-set (& lock))
        ; /* do nothing */
        /* critical section */
    lock = false;
        /* remainder section */
} while (true);
```

entry

① lock أول قيمة الها بتكون = 0 لانه ما حدا وصل ال critical

لهدا

② اذا حدا وصل رح تاخذ الصفر وترجعهم as a return value وتغير قيمتها لواحد

* لو حطيت (0) داخل شرط ال while رح يكون خطأ ورح تتخطى ال empty loop

وتروح لل critical

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لو وراة مباشرة اجا P2 و استدعي test-and-set رح يلاقي قيمة ال lock=1 وبالتالي تحقق شرط ال while وبالتالي رح يفوت بـ empty loop ورح يرجع يتأكد من شرط ال while ويرجع يفوت بال loop

* test-and-set اشي بتعمل ؟؟ بترجع القيمة القديمة لل lock وبتحاول تعملها = 1

* الكود السابق يضمن ال mutual exclusion لانه اي يوصل بالاول بروح على ال critical وبسكر الباب وراه

* لو توقف كل لهون رح يكون في mutual exclusion ولكن ما في progress و Bounded waiting

* لعملية لما P1 تخلص رح ترجع ترجع ال lock = 0 = false وهذا يؤدي انه P2 مثلاً وهو معلق بين ال empty loop وال الشرط تبع ال while في احد اعرات يطلع معه الشرط false فيخرج من ال loop ويروح على ال critical

في progress : لانه اي سمح لـ P2 تفوت كان P1 لما خلى ال lock = 0

وفي Bounded waiting : لانه وقت انتظار P2 كان وقت تنفيذ P1 ورح يخلص

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* حتى لو وصل P_1 , P_2 مع بعضا عند ال test-and-set
ما رح يكون في مشكلة لانه ال HW رح يمنع حدول ال 2-process
يعمل test-and-set بنفس الوقت ، رح يمشي واحد ويوقف الثاني

* لو الثنين مسموح يتنفذ واسع بعضا رح تختفي خاصية ال mutual exclusion
لانه الثنين رح يشوفوا ال lock=0 والثنين رح يوصلوا عند ال critical

* Compare-and-swap نفس ال inst السابقة
ولكن على أوسع وبتشبه
اشي اسمه Counting Semaphor

```
int compare-and-swap(int *value, int expected, int new value)
{   int temp = *value;
    if (*value == expected)
        *value = new-value;
    return temp;
}
```

تخزن في متغير وهي رح تكون ال return value

رح تقارن حدول مع بعضا اذا متساويين غير ال value بال new value

* تغير قيمة ال value لن يؤثر على

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

القيمة الابتدائية لها دائماً = 0 يعني unlocked

P_1 P_2

```
do {
    while (compare-and-swap(&lock, 0, 1) != 0)
        ; /* do nothing */
        critical section
    lock = 0;
        remainder section
} while (true);
```

* تخيل وصول 2-process بنفس اللحظة عند ال while
وبما انها Atomic رح تنفذ من قبل process واحدة فقط

* لنفرض تم اختيار P_2 لاستكمال التنفيذ و P_1 رح تنتظر

① قيمة ال lock=0 في البداية وهي القيمة الراجعة calling ال
comparand swap (&lock, 0, 1)

رح أقارن جدول اذا متساويين رح أبدل
قيمة ال lock بال الرقم الثالث

0 = 0 ✓ صارت
$\boxed{lock = 1}$

قبل هذا الحكي رح تقارن
0 != 0 خطأ لذلك رح يتخطى ال loop ويروح على ال
critical ، P_1 معلقة داخل ال loop لانه P_2 وصلت ال
critical

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

2/ تضل P_1 معلقة داخل ال loop لحد ما يرجع ال 0 = lock

* يرجع = 0 عما P_2 تخلص شغل ، يرجع فتحقق من الشرط كمان مرة

بعماي ال Alg في mutual execlution وفي ال progress وفي Bounded waiting

Alg سلايد 22 :

```
do {
    waiting [i] = true ;
    ;
    ;
```

* عشان تفهم أي Alg لازم تحدد ال critical عشان تعرف بين ال entry وال exit

* جميع الاسطر ما قبل ال critical هي entry والهدف منه ، انه يخلي one process only تعمل ال critical والباقي يعلقوا داخل جملة while Busy waiting أو ال spen lock

لاحظ ال critical section محفوظ بين */ * / ويكتب as psoudo code يعني مش مبين ايش جواته ولكن لو اجينا ندور رح نلاقي جمل بتعدل على shared variable

* ما بعد ال critical ولحد ما نشوف كلمة ال remainder هذا جزء ال exit والهدف من هذا الجزء البحث عن أحد ال process إي تعطلت بال entry عشان تمشيها تجنباً لحدوث ال starvation

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لو شلنا ال entry section النتيجة بتكون ؟ يفقد ال critical الحماية ويبطل في mutual execlution

* واذا شلنا ال exit section ؟؟ بصير في starvation وإي معلقين مرة رح يضلع معلقين للابد

* عشان تفهم أي حل لازم تعرف حل هو لـ 2-process فقط ولا لـ m-process
* وعشان تعرف انه إي بتقرأة كود حل لمشكلة لازم يكون في critical section ولازم entrey و exit و remainder

* كود سلايد 22 : لو بدك تعرف الكود شغال على أي process تطلع على variable إي بكون index لـ waiting array

- ال array إي هون اسمها waiting لانه أي process قبل ما توصل ال critical section بتوقف وبتشوف ايش وضع او critical اذا فاضي وكانت هي أول الواصلين رح تكمل أما اذا كان مشغول رح تنتظر

* ال deflut انه ال process رح تنتظر : waiting[i] = true ;
logical meaning : process i wish to enter critical section
- هل هي وصلت لـ critical ؟؟ لا ولكنها دخلت على اعنافة عليه

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بما انه ال process قاعدة بتعمل initilization ; key = true

القيمة بـ true معناته بالأساس كانت false

not shared
bettwen processes

waiting key

- أي process غير مهتمة رح يكونوا العام false كل وحدة الها waiting و key خاص فيها

- بما انه ال 2 flags = true معناته

(i) process مهتمة انها تدخل critical

لاحظ انهم الثنتين → while (waiting [i] && key)

True لذلك تحقق اشرط معناته رح تنزل على الجملة

أي بعدها وتعلت key = test-and-set (&lock) ;

lock بالبداية بتكون قيمة صفر أو false (قبل ما يبدأوا كل ال process) رح تاخد قيمة ال lock القديمة وترجعها وتعدل عليها وتصير (1)

بالتالي اصبحت ال 3 معطيات كالتالي:

① true = waiting

② true = key

③ true = lock بعدما غيرناه من صفر

* رح تضل تكرر ال loop لحد ما اشرط يصير false ، فأول مرة لما دخلنا كان عندك waiting and key ← true and true لذلك كان اشرط true

وتنزلنا على ال next statment

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* الآن لو دخلنا مرة ثانية :

waiting[i] لساتها true

key = false لانه أخذ القيمة القديمة لل lock إلي كانت (0) وبعدين

عدل القيمة بعدما أخذها

كان true وصار false وبناء على ذلك ال loop انتهت لانه شرط ال while

أصبح false (key and waiting[i] true) while false انتهت ال loop

ح تقفز للجملة إلي بعدها waiting[i] = false ;

* Pi كانت معتمة ونجحت بالوصول وبالتالي

حالتها بطلت انتظار لعميك صارت false

* لو شلنا هاي الجملة شو رح يصير ؟؟ بتكون ال Pi وصلت لل critical

ولسا حالتها انتظار فبصيرين contradiction بين اعمى وإلي حصل

على أرض الواقع

* عند هذا السطر : give the statues about the hole

① ال waiting هاي array فتحتوي system i

على Pi , Pj , Pk وحالتهم كالتالي :

Pi ↓ false (نجح بالوصول) ، Pj ↓ true ، Pk ↓ true

② قيمة ال lock = 1 (لانه Pi غيرها من 0 إلي واحد) القيمة الجديدة

③ عند key بكل process ← key(i) , key(j) , key(k)

0 → القيمة القديمة لل key = lock ، 1 ، 1

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* هل يوجد Mutual exclusion ؟؟ يوجد لانه لو حاولت تنفذ P_j رح تعلق لحتى يتحول الـ lock = 0

* بتحول الـ lock = 0 لما الـ P_i تخلص من الـ critical بالتالي بالـ exit ضروري يكون موجود ; lock = false

* P_i رح تظل تنفذ لمدة زمنية محددة ، رح تخلص بعد ما تخلص إلي لازم يصير :

① تحكي لـ P_j و P_k انه همي خلصت ولازم حدا مكاني يفوت على الـ critical section وينتهي الـ Busy waiting (شاغلة الـ cpu باشي عالفاضي)

② اختيار البديل يكون بالترتيب :

معناها الـ (P) إلي بعد (i)

$j = (i+1) \% n$

circular

عشان بعد ما الاخيرة تخلص تنادي الأولي

while ((j != i) && !waiting[j])

معتمين نشوف process i فرق لانه ممكن j = i بسبب الـ mod

اذا ما حطينا (!) رح يصير شرط الـ while true بالتالي يعلق

waiting إلي رح نختارها لازم تكون true عشان تكون interested

* بالـ while إلي قبل ما حطينا (!) عشان يعلق هون حطينا عشان يطلع شرط الـ while خطأ ويكمل

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بالتالي مواصفات الـ (P) إلي رح تعدي الـ while التانية :-

① ON turn : الدور واصل لعندها

② waiting = true

* اذا مش دورها مارح نمشيها واذا الـ waiting = false فهي أصلاً not intersted

- j = (j + 1) % n ; → هي الـ next process إلي وصل دورها

if (j == i)

lock = false ; → بما انك بتلغي الـ process مين إلي كان أصلاً داخل على الـ critical ؟ Pi

اذا هي فعلاً إلي دخلت على الـ critical اعمل الـ lock = false

* لما يصير الـ lock = false شو النتيجة؟ الـ Key تبع أحد الـ (P) رح يصير false وبالتالي رح تنتهي الـ loop ورح تدخل (P) جديدة (P) الجديدة بما انها بطلت تنتظر رح يصير الـ wait اللي false

هذا الـ طرلا ← waiting [j] = false ;

يؤثر على Pi أبداً ← موجود لتعديل flag الـ (P) إلي دخلت

* الـ progress موجود لانه إلي بطلع بدخل غيره ، هون Pi لما طلعت دخلت غيرها وهو Pj . بالتالي إلي مش الـ Pj هو واحد من الـ team الموجودين

* الـ Bounded waiting موجود لانه وقت انتظارهم يعادل وقت تنفيذ الـ process بالنظام أو لحدها الـ lock يرجع (0)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* الكود السابق كان سلايد 22 صح انه حقق الخصائص ال ٣ ولكن عليه الملاحظات التالية:-

① complex

② ال test-and-set هاي تكون مدعومة بال HW فاذا شريت جهاز وهاي ال inst مش مدعومة بال assembly تبعته بالتالي رح يتعطل الحل

وهاي الجملة قد لا تعمل بال pthread بالتالي اذا هاي الجملة عشو شغالة رح يبطل هذا الكود حل مقترح

* هذا كله تمهيدات للحل الحقيقي اي رح نستخدمه:

* Mutex Locks (semaphore) extintion

mutual execlution

→ : class with : ① Bolean variable

② 2-member functions : Ⓐ acquire

Ⓑ relase

Ⓐ acquire : بتزيد قيمة ال Bolean variable عشان تحجز ال critical section تستخدم بال entrysection

Ⓑ relase : تبطل القيمة تبع ال Bolean variable بعدما تخلص تستخدم بال exit section

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Memo No.

Date / /

كود سلايد 24 : لاحظ كيف تم استبدال كل ال entry

بال acquire (get to critical section)

لو عندي n-process لازم واحد يقدر يعمل acquire والباقي لا

يعلقوا ب Busy waiting او spenlock

ولاحظ كيف شلنا كل اشي قبل ال remandir وحطينا release

لاحظ بكود سلايد 22 جملة ال release هي : lock = false

اذا شلنا هاي الجملة كل ال process إي معلقين رح يضلوا معلقين

ويصير لهم starvation

* اذا ما في release في starvation

* واذا ما في acquire ما في protection لل critical section

* ال acquire بغض النظر عن كودها رح تكون ميزتها :

اذا أكثر من process وصلوا ال (acqire) واحد رح ينجح

والباقي رح يعلقوا

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* while (! available) *
↳ Bolean Value

بالبداية بتكون True كتعبير انه ال critical section فاضي
وأي واحد بحصل على ال critical section بخليها false

- ال acquire بخلي ال available = false لما حدا
ينجح بالوصول وال release برجعها True لما إلي جوة
ال critical بخلص

- لو وصلوا أكثر من process عند ال acquire أول واحد
بشوف ال available رح يلاقيها true و بعدين رح يفوت
على شرط ال while ← false = ! true بالتالي لم
يتحقق شرط ال while لذلك ما بفوت على ال Busy waiting
وبروح على ال critical طبعاً بخلي ال available = false
عشان كل حدا بعده يعلق لحد ما يخلص ويرجع بجملة ال release
يعمل ال available = true

* من اهم معرفة انه ال acquire وال release هم atomic inst..
يعني non-interruptable يعني بمجرد بدء التنفيذ حتى لو
اجاهم أمر بالتوقف ما بترك ال cpu وهاي ال inst.. مسموح
لـ process وحدة تنفذها يعني serial

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* يعني لو وصلت أكثر من process عند ال acquire رح يختار وحدة بس.

* لو مسموح لأكثر من process تنفذ ال acquire اشي رح يصير ؟؟ كلهم رح يشوفوا ال available=true وبالتالي كلهم رح يوصلوا لل critical وهذا اشي ما بدنا ياه

* بالتالي لازم ال release وال acquire يكونوا atomic يعني لو عندي 10-cpu's و 10-processes شغالين و 10 بحاولوا يعملوا acquire رح نمشي وحدة ونأخر الباقي

لساتنا بكود سلايد 24: بالكود جملة ال while إي بتخلينا نعلق عي دلالة على ال Busy waiting

* دايمًا ال Busy waiting يكون بالكود الذي يسبق ال critical section

هل ال Busy waiting اشي منيح ؟؟ آه و لأ بنفس الوقت آه لانه بحمي ال critical و لأ لانه ال process معلقة ب empty body بالتالي مش قاعدة بتعمل اشي فهي حالتها العامة executing ولكن فش task حقيقية قاعدة بتصير بالتالي cpu utilization وال cpu throughput بنزل كثير

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ال process داخل ال Busy waiting هل تحولت ال status تبعها ل waiting ؟؟ حسب حالات شابتر 4 ؟؟
لا، هي run بالتالي No contex switch وهنا اشي منيح لانه تقلل ال overhead

- عشان تقرر اذا ال Busy waiting منيح أو لا بدك تشوف ال process كل كم بتشيك على شرط ووقت ال contex switch
- اذا وقت ال contex switch كان أطول معناته خليها تعلق على ال cpn أوفر من اني أفضلها وأرجعها
- بالمقابل اذا كان ال contex switch قصير كثير ووقت الانتظار طويل جدًا بكون من الأفضل تفضلها واذا بدك ترجعها اعمل contex switch

- بالنهاية ال Busy waiting شغلة لابد منها الهاميزات وسلبيات مقابل حماية ال critical لازم تضحي

* ال mutex على التطبيق غير موجود ولكن يوجد extention عنه اسمه semaphore
* ال mutex هو ممكن نعتبره زي الاصدار الأول وال semaphore هو الاصدار الثاني إلي بعمل كل اشي بعمله ال mutex وزيادة

الممسوحة ضوئيا بـ CamScanner

Mo Tu We Th Fr Sa Su

Semaphore

Memo No.
Date / /

* ال protection الحقيقي إلي بصير هو باسم ال semaphore
وإلي رح نعتمدة ك synchronization tool حتى تزامن الوصول
لل critical section

- كنا عرفنا ال mutex انه class وفيه قيمة و 2-functions

- ال semaphore برضه فيها قيمة ولكن غيرنا اسماء ال 2-function
releas → signal() , acquire → wait()

كود ال semaphore مشابه لكود ال mutex ولكن عشان نحقق ميزات
أكثر : ال (available) كان Bolean يعني محصور
بقيمتين true, false ما عندك مرونة بال critical section
أما بكود ال semaphore رح نعطيها قيمة وتضل العمليات إلي عليها
اخصم أو زيد

(من ال resource)

① wait(s) بتشيك على قيمة ال s وبتطرح اذا استطعت الوصول لل
critical

② signal(s) بتزود على القيمة (s)
(نضيف لل resource)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

اسم المتغير بالعادة (S) من كلمة semaphore

wait (S)

الحل إلي هون ليس حصرًا

على 2-process هو بشتغل critical section

بعدد process لانهائي

Signal (S)

لو فرضت عندك P_1, P_2, P_3

كلهم بتنافسوا انهم يمروا عن ال wait

~~ويوصلوا~~ ويوصلوا على ال critical

+ اشي إلي لازم يصير حتى نحكي انه هذا

is a valid solution ؟ اذا واحد من ال process

نجحت وللباقي علقوا.

0 ↖ ↗ 1

- بالبداية تكون قيمة S = 1 لنوع معين اسمه Binary semaphore

افرض P_2 وصلت أولاً فهو بده يحاول يعمل wait أول اشي

يعني رح يعمل execute لكود ال wait ؛ wait(S) ؟

while(S <= 0) → (1 <= 0) false

لم يتحقق شرط ال while // busy wait ؛ ؟؟

قيمة ال $\boxed{S = 1-1 = 0}$ → S = - ؛

بعد كود ال wait بيجي ال critical

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لنفرض وصول P_1 الآن : ال $S=0$ الآن بالتالي
رح يعلق بال $0<=0$ ✓ ← while ($S<=0$)
Busy waiting راح ال empty loop (فاضية) راح رجع يشيك اشرط
اشرط ما تغير لازالت القيمة (0) وبالتالي رح يضل
يلف لحد ما تتغير قيمة (S)

($S=1$) يعني ال critical مش محجوز ، ($S=0$) يعني محجوز

* لو وصلت P_3 رح يكون مصيرها زي P_1

* الآن (P_2) بعد ما تخلص من ال critical رح تعمل signal :
signal (S) {
S++ ⟶ $S=0+1=1$ (غير محجوز)
}

من إي process المعلقين اختار أحدهم لنفرض P_3 رح يشيك على
كود ال wait رح يلاقي while ($S<=0$)
$1<=0$ False رح يخرج من ال Busy waiting
وينزل على ⟵ $S=-$ ⟵ ($S=1-1=0$)
ورح يفوت على ال critical section

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* ال wait وال signal ضروري يكونوا Atomic ؛ انه لو كانت $S=1$ ووصلوا 2-process بنفس اللحظة واحد رح ينجح بالعبور والآخر لا

* النوع السابق اسمه Binary semaphore اله $S=0, 1$ فقط
مجوز (غير متاح) غير مجوز (متاح)

* الخصائص:

① mutual exclusion : ينفذ واحد والباقي يعلقوا ✓ موجود

② Progress : إلي بمشي المعلقين واحد من المجموعة ✓

③ Bounded waiting : وقت الانتظار نفس وقت تنفيذ أحد ال (P) ✓
اذا تجاهلنا الأولويات واشتغلنا بـ FCFS (على الترتيب)

* لذلك بالأغلب ال semaphore بكون معها queue بنخزن فيها إلي ما نجحوا بالوصول لل critical ، ترتيبهم داخل ال queue حسب مين وصل بالبداية (لانه لو بدك تشتغل على الأولوية ممكن P4 توصل جديد وتتنفذ والباقي ينتظر ويصير لهم Starvation)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لو غيرت الفرضية وحطينا عنا 2-critical section :

افترض انه أحدهم هو تعديل على ملف ، هل اعلن لازم يكون منه نسخة واحدة فقط ؟؟ من شرط ممكن تكون عامل منه نسختين بعني عندك 2-instances on the same resource ، فالآن من غير اعتراض بوجود 2-instances تمشي process واحدة فقط واعوجود يكفي (2)

بالتالي رح نمشي (2) والزيادة هم بتعطلوا

* الكودات السابقة كانت تفترض انه ال critical section موجود منه نسخة واحدة فقط بالتالي ماني مجال أخدم (2) بنفس الوقت

مثال : إلي بتنافسوا بالتعديل على ال critical هم P_1, P_2, P_3, P_4 اعوجود يكفي (2) بالتالي 2-must accesse critical ولل other 2-process must Block :

* الآن ال semaphore السابقة (Binary) ماتخدمك لانه عندها قيمتين فقط 0, 1 لذلك رح نستخدم counting semaphore

counting semaphore : لازالت class بمتغير و 2-function's (signal, wait)

بس ال initial value لل counting semaphore = عدد النسخ المتوفرة

* بما انه عن نسختين من ملف بالتالي $\boxed{S=2}$

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لذلك انت كمبرمج متى تستخدم ال Binary ومتى ال counting ؟

* اذا انحكالك انه ال resource في منه نسخة واحدة رح تستخدم ال Binary وال S العاقيمة مبدأية تساوي (1)

* اذا انحكالك انه ال resource في منه كذا نسخة استخدم ال counting وقيمة S الابتدائية تساوي عدد النسخ.

① ال counting semaphore رح يكون نفسه بال 2-critical section

P1 P2 P3 P4

wait(s) wait(s)

critical critical

signal(s) signal(s)

~~critical~~ (1)

P1 ;
S1 ;
signal (synch) ;

~~critical~~ (2)

P2 ;
wait(synch) ;
S2 ;

هذا الكود بعدين

* الآن 4-P بتنافسوا وبدنا نسمح لواحد يعمل wait على كل نسخة مين هو هذا الواحد ما بعمني اعمم ما يوصل أكثر من واحد لل critical عند كل نسخة

* طبعًا ممنوع 2-wait يشتغلوا بنفس الوقت لانها Atomic

لنفرض P4 عملت ال wait (كود ال wait بعمل while (S<=0) ; //Busywaiting

2 يتخطى ال Busywait ← NO 2<=0

S -- ;

رح يطرح من S ← $\boxed{S = 2-1 = 1}$

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

بعدين P4 رح يوصل عند ال critical وهذا معناه انه حصل على
واحد من أصل (2) من ال resource

* الآن رح نختار P1 فرضاً ورح يعمل wait ويتأكد من الشرط
while(S≤0) ← No , 1<=0
رح يتخطى ورح يوصل ل ; // Busy waiting
S--
$\boxed{S=1-1=0}$

ورح يوصل لل critical ورح يحصل على النسخة الأخرى
ال critical هون ما صار عليه data inconsistancy لأنه P4
قاعدة بتكتب على سلف و P1 بتكتب على سلف آخر

* الآن لو حاولت P2 الدخول while(S<=0) ;
yes , (0<=0) ✓ بالتالي تحقق
شرط ال while ورح يعلق بال Busy waiting
ونفس الحال ل P3

* المثال للتوضيح احنا أصلاً ما بنعرف مين إي رح يوصلوا لل critical
* اذا حكالك استخدم FCFS رح تمشي على الترتيب من P1 وتكمل
أو اذا أعطاك أولويات مشيهم حسب الأولويات
* بال critical ما بهمني الترتيب اعمم اني أحميه من 2-access
at the same time

الممسوحة ضوئيا بـ CamScanner

Memo No. الاثنين
Date 18 / 8 / 2025

* على فرض انه P4 خلص قبل P1 رح يعمل signal يعني
S++ ← S=0+1=1 ، المعلقين P2 و P3 كانوا يستنوا
القيمة تبطل (0) بتمشي واحد فيهم حسب FCFS او الأولويات

* هذا الحل لسامش optimal لانه لسا في Busy waiting رح نشوف
حل لهاي المشكلة :

* للتذكر : ايش ال command المستخدمين لادارة ال process في
ال posix : wait , exit , join , Fork

* طيب تبعون ال threads : Pthread-creat (activate)
Pthread-t (x ...) (انشاء thread ولكن مش شغالة) , Pthread-join (wait)
اسم ال thread
Pthread-exit : تستخدم لتنهي اشي قبل نهايته الطبيعية
Pthread-self : بترجع ال id تبع ال thread بتشبه بال process (get pid())
وهو رقم يميز ال thread داخل ال process
ويمكن ان يتكرر في ال process مختلفة
(لو عندك ال 2-process وفيهم thread ممكن
ال id تبع ال thread يتشابه) لانه
ما يميز كل thread وجودها في ال process الهم id مختلف
* thread الهم نفس ال id : عادي
* ال process الهم نفس ال id : ممنوع

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* Pthread-self ممكن نستخدمه اذا بدك thread معينة تعمل جزء من كود أو انك تنهي thread معينة

* Pthread-cancel : تنهي thread نتيجة مشكلة صارت (exit) لنهاية عادية ما صار error

حكينا انواع ال cancelation : synchronous , deferred
(توقف مباشرة) (بتوقف بنقطة آمنة)

* لو بدنا نراجع الحلول السابقة بالتدرج لحتى نوصل ال optimal

① بتستخدم variable اسمه ال term وهو غير فعال لانه فقط بدعم ال 2-process وما بحقق ال progress

② بدمج 2-process اي هي Pi و Pj كان بحقق ال ٣ شروط : Mutual execlution , progress , Bounded wait ولكنه يكفي لل 2-process فقط بالتالي مش فعال وعنده مشكلة ال Busy waiting (بتضيع وقت ال cpu وما في انجاز)

③ انتقلنا لل (HW-solution) عن طريق test-and-set , compare-and-swap وهم function ممكن نستخدمهم لتطوير الحل النهائي ولحماية ال critical (مساعد وليس الحل) عندها مشكلة Busy waiting , مشكلة اذا هاي ال instrection مدعومة بال assembly أو لا فممكن ال HW تبعك ما يكون يدعم هاي ال instruction وبالتالي ببطل الهم فائدة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

④ ال Mutex في مشكلة انه بدعم only one instant يعني
لازم ال critical يكون في منه نسخة واحدة حتى يشتغل ال Mutex

* على أرض الواقع ممكن تلاقي many instanses من ال resource أي عليه مشكلة

⑤ اكل النعماني semaphore: وهي class يحتوي مجموعة من المكونات

(V1) الاصدار الأول لل semaphore فيه متغير و 2-functions
wait: بتطرح من المتغير ان أمكن
بتزيد على المتغير دائما (signal)

ليش بال (V1) بال wait بنطرح إن أمكن ؟
لانه أقل قيمة ممكن يوصلها المتغير هي (0)
بالتالي عملية الطرح --S بتم اذا القيمة (positive)

ال Binary semaphore بتشبه ال Mutex لانهم الثنين شغالين على نسخة واحدة
من ال critical

* هي بعمل signal بزيد ال (S) بغض النظر عن ال statuse بالتالي
انتبه- تحط S++ بموقع خاطئ يعني لو اعبرمج بدأ كودة ب

① Signal (S) → S=1+1=2
② Signal (S) → S=2+1=3

بالبداية قيمة $\boxed{S=1}$

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* حكينا عن الـ Counting semaphore انها بتخلي قيمة S تساوي عدد نسخ الـ resource اعو جودة معناته لو بدأنا بـ S=3 معناته عندك ٣ نسخ من الـ shared resource لذلك رح نشغل أكثر من process / thread فبتبدأ بـ S=3 وكل واحد بحجر رح تنقص الرقم S=2 ← S=1 ← S=0 إي شغالين ٣ طيب لو اجا واحد رابع ؟ رح يفوت على الـ Busy waiting

* فائدة الـ semaphore تتعدى الـ critical ويستخدموها كمان to rearrange the code اذا عندك أكثر من part في الـ program وبدك تمشي الأول وتخلي الثاني يستنى رح تبدأ بـ

semaphore = synch = 0

وعندك لـ 2-process بحاولوا ينفذوا

code (1)
- P1 ;
- S1 ; → any programing code رح ينفذ لانه لا ينتظر شي
- Signal (synch) ;

حطهم مكان S1 عند code(1) [S1, S2, S3] رح ينفذوا بدون عائق ورح ترتفع synch=1

code (2)
- P2 ;
- wait(synch) ; → wait (0) Block P2
- S2 ; لحد ما ترتفع قيمة synch عن صفر

حطهم مكان (S2) عند code2 [S4, S5, S6] → رح يضل عامل Block حدون لحد ما (code1) يخلص ويرفع قيمة الـ synch=1 بمجرد ما توصل (1)

كود [S4, S5, S6] (P4) رح يطرح عنها ويميل * لا وجود لـ critical section

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* لو الكود السابق شلنا ال semaphore يعني شلت ال signal, wait

S_1
S_2 P_1
S_3
S_4
S_5 P_2
S_6

ممكن يصير تنفيذ S4 بعدين S5 بعدين S1, S2, S3
وبعدين S6 أو أي شكل آخر يعني رح يصير
في race condition ويعتمد الناتج على الأسرع
بالتالي اذا بتعمل أكثر من run رح يتغير الناتج كل مرة
تقريبا

* ال semaphore تجبر الكود يمشي بالترتيب اعطوب نعطي النظر كم مرة
رح تعمل run للكود.

V1 من ال semaphore عندها Busy waiting لذلك: أول process
بتوصل ما رح يكون عندها Busy waiting وال (n-1) process رح يكون عندهم

* last version of semaphore implementation ما فيها Busy waiting

الفكرة: أي process بتوصل عند ال semaphore وتبلاقي قيمة
ال value أكبر من صفر هي بطبيعة الحال ما رح تعمل Busy wait
فما بلزمها تعديل

* بالمقابل إي بتوصل عند ال semaphore وتبلاقي قيمتها صفر بدل ما تعمل
Busy waiting (هون ال process بكون حالتها run) لو شلناها وحطيناها بال
waiting state لحد ما حدا يعملها signal هل بتأثر على طبيعة العمل ؟؟

الممسوحة ضوئيا بـ CamScanner

Memo No. _____
Date / /

* بدل ما تكون تنتظر وشاغلة ال cpu رح نخليها تنتظر خارج ال cpu
بالتالي تتحول ال status ابعها من run ← wait ساعتها ال signal
بصير معناها ؛ روح صحي ال process من ال waiting وعملها
Pick up من ال queue هون رح نضيف على ال semaphore ؛

① Block ② wake up

Block ↓ بدل ال while
wake up ↓ صحي إلي قاعد بنتظر

يعني فش Busy waiting

```
wait (semaphore *s) {
    s -> value --  ;
    if (s->value < 0) {
        add this process to s->list ;
        Block() ;
    }
```

① رح يطرح بالأول
② رح يحيل بعد الطرح
اذا القيمة سالبة معناته كانت اما (0) أو رقم سالب وبالتالي لازم يعمل Block لأن ال resource محجوزين
↓ convert process to waiting

① contex switch ② suspend old one ↓ إلي وصلت ولقت القيمة (0)
③ find new thread or process to run

التعديل إلي هون مكلف لانه فيه contex switch (وقت زيادة رح يضيع) وبدنا queue (extra HW) رح تخسره ولكن كل هنا عشان ما نضيع وقت ال cpu

Memo No.
Date / /

* هل يتحسن ال utilization ؟؟ مش شرط لانه اذا كان وقت ال Busy waiting أصلاً قصير كان الخيار السابق مش أفضل ، لانه كان عملت spenlock شوي وكملت بدل contex switch و queue و waiting أما اذا الوقت طويل لانه في تنافس على ال resources الأفضل تعمل Block وتتحول waiting state

* if implementation code is short : Busy waiting

* if the process spend lots of time in critical section : Block and go to wating state

* لذلك هل النسخة النعمانية من semaphore على الأكيد رح تعطيك قيمة مضافة ؟؟ مش شرط ، يعود لحجم ال critical section

* بالنسخة الأولى من ال semaphore كان بال wait يشيل كل S بعدين يطرح لانه ممكن ما يقدر يطرح اذا القيمة (0)

* بالنسخة النعمانية يطرح بالأول بعدين بشوف كم صارت القيمة اذا سالبة رح يعمل Block اذا موجبة رح يكمل

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* signal معناته حدا قرل resourse

```
signal (semaphore *s) {
    s->value ++;
    if (s->value <= 0) {
        remove aprocess P from s->list;
        wakeup(P);
    }
}
```

زيد القيمة

وصحي واحد من

إي بستنوا

(اذا في واحد نايم أصلاً)

قلنا قبل شوي إي بوصلوا

بعدين وبنعمل لهم Block تتكون

قيم الطرح بالسالب واذا بعد

Value ++ وصلت (0) معناته

هدول إي بستنوا

(0 أو أقل)

ليش قبل كان Block() أقواس

فاضية

وهون wakeup(P) أقواس

مليانه

① لانه إي عمل Block هو نفس ال P

إي وقت

② إي بده يصحي مش نفس إي شغال

إي بين الأقواس هو أول واحد قيد

الانتظار

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* سوء استخدام ال semaphore ممكن يترتب عليه starration و Deadlock (مشاكل من عدم حسن استخدام ال semaphore) مش دائمًا بصيروا

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
...	...
Signal(S);	Signal(Q);
Signal(Q);	Signal(S);

* كود بين مختلفين ممكن كل واحد ل process أو ل 2-thread بنفس ال process ممكن يكونوا بنفس اعلف أو منفصلين مش مهم

① قيمة semaphore(S) و semaphor(Q) المبدأية اح تكون (1) وبالتالي أي محاولة wait اح تنجح

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

② لو اجيت تنفذ on-parallel كونه في 2-cpu

P_1	P_0
wait(Q) ما بعمل Block وبمشي	wait(S) ما بعمل Block وبمشي
$Q = 1-1 = 0$	$S = 1-1 = 0$
wait(S) بما انه $S=0$ رح نوقف عندها	wait(Q) بما انه $Q=0$ رح نوقف عندها
signal(Q) بتوقف تعليقة Q عند P_0	signal(S)
signal(S)	signal(Q) بتوقف تعليقة S عند P_1

P_0 ، P_1 تنينهم معلقين ومش عارفين ~~يوصلوا~~ يوصلوا الـ signal لانه لحتهم Block ، هو اشكلة قائمة على 2-process لذلك ما بنسميها Starvation ، رح نسميها Deadlock ؛ أكثر من واحد بعاني (واحد بعاني) أو أكثر والسبب بسبب وضعية هم مسؤولين عنها من بعض

الحل ممكن انه تعمل Kill لاحدهم وترجع الـ resource إلي كان آخذهم عشان تنفك الحلقة

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

+ افرض P_1 العمل أولوية قليلة ، P_2 العمل أولوية متوسطة
P_3 العمل أولوية عالية

	P_1	P_2	P_3
	low	mid	high
Semaphore(S)	1:00	1:01	1:30

افرض الساعة وحدة مين إلي رح يمشي ؟ P_1 بالتالي رح
تطرح من $S=1-1=0$ وتروح على الـ critical
فرضاً وصلت P_2 العمل أولوية أعلى من P_1 عند ساعة 1:01 رح
تتوقف P_1 ورح تروح P_2 تعمل إلي بدها يا 0 وفرضاً كانت
طويلة وأجت الساعة 1:30 هل رح تقدر P_3 تشتغل ؟
لا ، لأنه يعتمد على الـ S إلي توقف عملها (0)

preemptio
لوقف بدون ما
تترك الـ resource
على أمل أنك ترجع

* اعفروض أنك بتعمل priority scheduling ولكن إذا
بتتطلع على إلي صار مش هين
P_2 عامل Block لـ P_3 فصار P_3 بيستنى P_2 يخلص
حتى يرجع P_1 ويخلص عشان ترجع الـ $S=1$
فصار الأعلى (P_3) ينتظر الأدنى P_2 ، P_1 وهذا اسمه
(priority inversion)

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

في هون حلين :

① P_1 لما عملت preemption توخذ منها كل الـ resource وهذا الحل
لخلي P_1 (canceled) ورجعتها لنقطة الصفر ساعتها
بترجع $S=1$ ، لكن هذا الحل مكلف لانه ممكن P_1 يكون طايلها
10% وبتخلص

② Priority inheritance : P_1 استخدمت (S) و P_3 استخدمت (S)
هاد resource مشترك بينهم وهم أولوياتهم مختلفة ، الحل هذا
بقلك اذا الأولويات مختلفة خلي الكل بنفس أولوية أعلى واحد
بالتالي P_1 رح تصير high زي P_3 بالتالي اذا دخل P_2 على
الخط P_1 ما رح يقطع لانه أعلى أولوية وبالتالي لما
يخلص رح يرجع $S=1$ وبدخل رقم P_3 وبشتغل
وبعد ما يخلص السيناريو اللي فيه منافسة على (S) بترجع الأولويات
زي ما كانت . P_2 رح يتعرض لشوية delay بدل ما يصير له
starvation مطلق

Priority inheritance حل مشكلة سببها الـ semaphore

الممسوحة ضوئيا بـ CamScanner

Memo No.
Date / /

* بال Pthread كيف تعمل Binary semaphore, counting

lock يعادل ال wait بالسلايدات وال unlock يعادل ال signal
بما انه النوع الأول Binary semaphore أول ما تعمله initialization ما تختار له
قيمة لانه مجرد ما تنشأه رح تكون قيمته تساوي (1) ال lock
دخلي قيمة (0) ال unlock برجعها (1)

* بالمقابل بال Counting semaphore بتحط القيمة اي بدك ياها
(Semaphore Value) = حسب عدد النسخ الموجودين بصير كل wait ما تطرح
واحد وكل Post (بديل ال signal) تزيد (1)
* عليك تستخدمهم ومش لازم تعرف كيف بتنفذوا

الممسوحة ضوئيا بـ CamScanner

Pthread_mutex_t : عشان تعمل creation لـ Binary semaphore اسمها (X) أو أي اشي بدك ياه once declared بتقدر تعمل عليها ٣ أشياء :

1) pthread-mutex-init-(& semaphore-name , null)

Keep defult attributes the same

الاسم إلي اخترته مثلاً (X) ← initialize

بعد عملية الـ initialize قيمة semaphore(X) = 1

2) Pthread-mutex-lock (& semaphore-name) بتشيك على القيمة اذا مش صفر يعني (1) بتطرح منها وبتكمل اما إذا صفر ح يصير Bloking عند هاي الجملة

3) Pthread-mutex-unlock(semaphore-name) بتزيد القيمة بدون أي تشيك مسبق ولا عملت هاي الجملة مرتين ورى بعض طرح تزيد القيمة عن (1) لانه (Binary) بالتالي :

not absolute increment opration

* استخدامه اذا عندك نسخة واحدة من الـ resource

* أكثر من نسخة بدك تحول على الـ Counting

الممسوحة ضوئيا بـ CamScanner

Date Page

Sem-t : to declare semaphore name ممكن يكون

اسمها (S) أو أي اشي بدك ياه

① Sem-init (& semaphore-name, 0, semaphore-value)

① حسب عدد النسخ — Null — مثلاً (X)

② حسب كم process

بدك تمشي

② sem-wait (& semaphore-name) نفس ال lock

بطرح من القيمة بعدين بتأكد اذا صفر أو أقل ال process

بتعلق هون مجرد ما تصير (1) بتروح بتطرح وتكمل

③ sem-post (& semaphor-name) نفس ال signal

بزيد على القيمة بمقدار لا يتجاوز عدد النسخ

---

* Bounded-Buffer problem :-

نفس ال shared memory : ال producer وال consumer

كانوا يتشاركوا بال Buffer وقلنا ال producer بشيك اذا

مليانة ما بضيف وال consumer بشيك اذا فاضية ما باخذ

① اح يستخدم wait — ① اذا فاضية بضل عامل Block

لن تنجح — بال wait لحد ما حدا يعطي عنصر

② اذا فاضية اح يستخدم — ويعمله signal

wait تنجح ويبقى العناصر

الممسوحة ضوئيا بـ CamScanner

Date Page

لغايات الـ synchronization بين producers والـ consumers
(عدد الكودات / function إلي بدك تزودني فيها يعادل
عدد اعتنافيين)

* يعني اذا عندك producer و consumer بتوقع أشون كودين

* عدد الـ semaphores إلي بعملهما implementation يعتمد
على الاجراء

سلايد 32 : الحجم الكلي للـ buffer : n-buffer ①

لما تعرف (n) : الـ producer بضل يعمل produce لحد
n-times بدون عملية consume الـ n+1
اح يعمل Block
produce item يعني يعمل genarate لقيمة عشوائية
ويخزنها بالـ Buffer بالتالي تخسر (1) من المساحة الكلية
للـ Buffer

* (n) متى تبقل ومتى تتزيد : ① تتزيد اذا الـ consumer أخذ
② تبقل اذا الـ producer حط

* لذلك بحاجة نعمل 3-semaphores :

الممسوحة ضوئيا بـ CamScanner

Date Page

① Mutex : لما نشوف انه مسمي ال semaphore

mutex بدك تعرف انها Binary بالتالي قيمتها

(0, 1) والهدف منها Block shared buffer

* ال lock بصير قبل الاجراء عشان نمنع

بصير في 2-accesse عليها (القيمة الابتدائية = 1)

الكود (mutex-t)

فاضي ، مليان

② full : بتشيك كم موقع معبي (بالبداية عدد المواقع

المليانة = 0) بتهم ال consumer عشان يشوف

اذا عدد المواقع المليانة = 0 معناته فش اشي بتاخذ

③ empty : بتشوف عدد المواقع الفاضية وبالبداية تكون

كلها فاضية يعني (n)

* full, empty من نوع ال counting (كودها sem-t)

تبدأ بـ n وتنقص

لم تبدأ بـ (0) وتزيد

الممسوحة ضوئيا بـ CamScanner

Bounded Buffer Problem

Date Page

Producer process : سلايد 33 :

① ال critical section موجود تحت ال wait الثانية

وقبل ال signal الأولى

do ؟

...

/* produce an item in next_produced */

← هاي الجملة من critical لانه لسا ما حاول يحطها بال shared memory

بالتالي هون بس بعمل genaral-contue fly

لما يحاول يخزنها بمكان مشترك بصير بدة يوصل ال critical

② أي اشي قبل ال critical هو ال entry part وهدفه الحماية

وأي اشي بعد هو ال exit يسمح لواحد من إلي بستنوا

wait (empty) ; → empty semaphore لتعد كم مكان فاضي

بالتالي اذا عدد الاماكن الفاضية مش صفر

اح يمشي ، أما اذا صفر رح يوقف

* اح نفترض وجود m-producer

(ليس 1 فقط)

* افرض بعد عدة اجراءات على ال shared buffer أصبح عدد الاماكن الفارغة (0)

بالتالي ال producer لو حاول يضيف على

ال shared buffer لن ينجح ورح

يعلق عندها

* ايش إلي رح يرفع عدد الاماكن الفارغة ؟

① producer إلي كان ماشي وصل عند

ال signal تبع ال full وعمل release

لعدد الأماكن بـ (1)

② consumer أخد اشي .

* رح يضل عامل wait لحد ما واحد ياخد

اشي فبصير عدد ال empty = 1

اذا أخد (2) بصير عدد ال empty = 2

و هكذا

الممسوحة ضوئيا بـ CamScanner

Date Page

* لما الـ consumer ياخذ اشي بعمل signal لـ empty
وهذا يعادل empty++

* signal لـ mutex يعادل mutex++

* wait لـ empty يعادل empty--

* لو فرضنا عندك 5-producers و 4-consumers بالبداية ولـ buffer فاضية مين متاحلة انه يشتغل ؟؟
① any producer
② كل الـ consumer رح يكونوا معلقين

بالتالي لو افترضنا انه wait(empty) والـ empty=0 هاي مش بالبداية أكيد لانه لو عدد الأماكن الفاضية صفر معانه الـ buffer مليانه .

* (wait(empty معناها :
① if empty = 0 : اعمل Block لـ producer الجديد لانه آخرين سبقوه وخلصوا كل الـ shard buffer ومضطر يستنى
② لو كان أول producer بيوصل : empty=n مافي داعي يستنى رح يطرح واحد من الـ n ويمشي
③ عشان يحكي انه حصل على mutual execlusion لازم يعمل wait(mutex)
هذا يعني انه في حدا رح يكتب على الـ shared فبعمل Block للباقي

الممسوحة ضوئيا بـ CamScanner

* لما نخلص الاجراء لازم نعمل شغلتين :

① هو حجز مكان فلازم نخصم من ال empty أو نزيد على ال full

② لازم نعمل release لل resource

شيل ال lock عن ال buffer → Signal (mutex)

↓

لاني خلصت critical

زيد عدد الاماكن المليانة → Signal (full)

لانك producer فأكيد دخلت اشي

* Consumer process

Do {

بشوف عدد الأماكن المليانة → wait (full) ;

انا full=0 يعني فاضية فرح يعلق

* رح يعلقوا كل إي بعد ال → wait (mutex) ;

consumer إي مشي /* remove an item ..

* بعدما ياخد العنصر بنجاح

= اشعار انك خلصت من ال critical → Signal (mutex)

= زيادة عدد الاماكن الفاضية → Signal (empty)

لانه ال consumer بياخد

هذا الاجراء اي consumer رح يعمله → while (true) {

فبحطوه داخل do-while loop لانهما بتخلص ورح يظل ماشي بهاي الطريقة للابد

الممسوحة ضوئيا بـ CamScanner

Date Page

هل يوجد الخصائص :

① Busy waiting : ما بقدر أحكم لحتى اشون كود ال wiat ان افيه while او لا

② mutual execlusion : يوجد ، لانه wait(mutex) بمنع أكثر من واحد يوصل critical
علمًا انه Binary semaphor

③ progress : يوجد ، إلي بمشي الباقي واحد من الفريق

④ Bounded waiting : إلي معلق بضل يستنى لحد ما يزيد مكان
او يفضى مكان

Readers-writers Problem:

ممكن تفكر انها بتشبه ال producer - consumer

الفكرة هون : هل في مانع أكثر من reader يشتغلوا هون ؟ لا
لانه لما عرفنا ال critical قلنا هو مكان في تعديل على shared
resource فممنوع تين يشتغلوا عليه بنفس الوقت

* لو في many reader قاعدين بقرأوا معلومة هل في تعديل على
shared resource ؟ لا ، لانه القراءة not acritical
opration

* اذا اشتغل writer أي اشي ثاني لازم يوقف سواء reader او writer

* اذا اشتغل reader ممكن يشتغل معه reader اما writer لا

شغال	شغال
(reader with writer) = lock	(writer with writer) = lock
(~~writer~~ reader with reader) = lock	(writer with reader) = lock

الممسوحة ضوئيا بـ CamScanner

Date Page

① Semaphore rw-mutex initialized to 1

↳ protect read and write current values

② Semaphore mutex initialized to 1

protect the critical section بتحكي محجوز أو لا

③ integer read-count initialized to 0

ليست semaphore فما بنفع تيجي تحكي wait على ال read أو signal على ال read فاي بتعمل tracking read count يعني عدد ال readers اعرجوديت بالبداية بكونوا صفر

* rw-mutex بالبداية (1) لانه ماني مشكلة بالبداية reader أو writer يشتغل (لانه فش حدا شغال لسا)

الممسوحة ضوئيا بـ CamScanner

Date Page

Writer process :

do {

wait (rw-mutex) ;

try to accesse the critical section

my type is writer and

I must lock the critical

section after me بغض النظر

إي جاي reader أو writer

* rw-mutex

0 : في writer ثاني شغال أو reader

1 : فش حدا شغال

ولما يدخل رح تتحول لـ (0)

Signal (rw-mutex) ;

بعمل release بما يعني انه خلص عملية

الـ writing فبترجع rw-mutex = 1

عشان تسمح لاحد الـ reader أو الـ writer

يدخل

كود سلايد 36 :

① اذا اشتغل writer

لازم يمشي والباقي lock

② read-count

غير معممة هون لانه writer

③ mutex هي

لتنظيم الـ reader

فغير معممة هون

④ بضل الـ rw-mutex

إي بتحمي الـ reader

والـ writer

الممسوحة ضوئيا بـ CamScanner

Date Page

Reader Process :

كود سلايد 37 :-

① اذا اجا وراه reader زيه عادي بمشي لانه ما نجرب عليه

② اذا وراه writer رح يحمله Block

③ رح نمشي بالكود ونتخيل انه ال reader أول واحد وصل وبعدين نتخيل وراه وصلوا writers , readers

* wait (mutex) → استنى لحد ما اشون الوضع (ما بتعمل real synchronization)

* read-Count ++ → بما انه افترضنا انه أول واحد وصل (القيمة بالبداية بتكون 0) رح تصير (1)

* if (read-Count == 1) → أول واحد وصل لما كان ال critical متاح

* wait (rw-mutex) → القيمة الابتدائية العما (1) فلما أول reader يوصل بعملها (0)

* Signal (mutex) → فيمتعما (1) مع انه محجوز (القيمة هون لا تدل على الحجز أولا)

/* reading is performed */

افترض وصول P1 لعمون

الحالة 1 تخيل وصول P2 وهي أيضاً reader اي رح يصير:

① wait(mutex) → هي أصلاً (1) رح يطرح ويمشي

② read-Count ++ → عدد القارئين = 2

③ if (read-count == 1) → لا تنطبق عليه

④ signal (mutex) → انتهينا من القراءة رح ينتقل لعماي

الوضعية وهماي معناها ان القارئ الثاني وهماي عشان critical opration خليني أعمل وصول (فصاروا P1 , P2 بعملوا reading)

الممسوحة ضوئيا بـ CamScanner

Date Page

* لو اجا reader ثالث : بعمل lock لحد ما يشوف اشو الوضع
بشيك على ال read-count وبشوف اذا هو الأول أو الثاني
أو الثالث ... وجملة ال if لا تنطبق عليه وبالتالي مصيره
يوصل للـ critical

* لو وصل writer : لازم يصير Block عليه وما يقدر يوصل ال critical
كودة سلايد 36 : رح يوصل عند wait (rw-mutex)
وقيمتها صفر لانه أول reader خلاها صفر
وبالتالي رح يعلق هون لحد ما يخلصوا ال 3-readers
والاخير هو برجعها لـ (1)

ارجع لكود سلايد 37 :

* wait (mutex) → بشيك اذا هو القارئ الوحيد
فيتركها بالمرة ولا أنا أحد القارئين
فأخليها lock

* read-count ... ; → تقليل عدد ال reader بـ (1) لانه
واحد خلص

* if (read-count == 0) → بشوف عدد ال reader
اعتبقين اذا مش صفر
ما بترك ال critical وبخليه
lock

* signal (rw-mutex) → اخر واحد إلي ينطبق عليه شرط ال if
برجعها (1)

الممسوحة ضوئيا بـ CamScanner

* ملاحظة إلي بقرأ هو فقط بقرأ ما باخد عشان هيك لو قرأ 1000 واحد الداتا رح تضل موجودة وما تخلص و إلي بكتب بكتب وبضيف على الداتا الموجودة

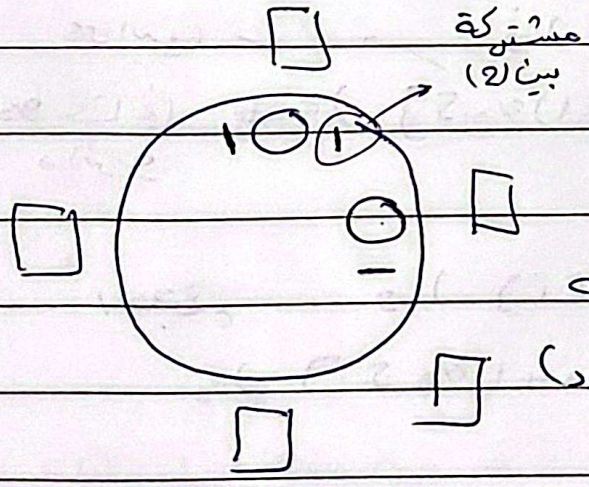
* الآن بال producer-consumer problem وبال reader-writer problem

↓ لازم تكتب كود لكل واحد فيهم

↓ لازم تكتب كود لكل واحد فيهم

* بالمشكلة الاخيرة (Dining-philosophers) كم كود

اح تكتب كودات بنفس عدد الاشخاص الموجودين



اح تكتب هون لـ (5) بالسلايدات رح تلاقي كود واحد لانه كودا تعم متشابهه باختلاف واحد فقط (index العود)

* أي واحد جالس على الكرسي بده ياكل

بالتالي بحاجة يوصل لـ 2 من عيدان الاكل لـ العود اشترك هو لـ critical ولازم يصير عليها wait و signal

wait : كانه مسكها بايده وشالها

Signal : رجع رجعها على الطاولة

الممسوحة ضوئيا بـ CamScanner

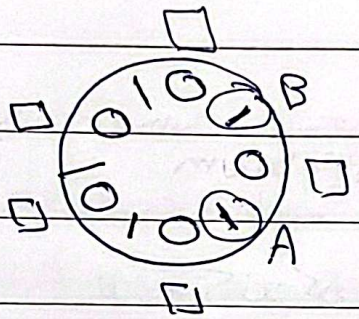
Date Page

بالتالي لما بدنا نكتب كود لأي واحد على الطاولة بدنا نتأكد انه ال

العودين متاحين اح نكتب (اسم العود) wait

فرضا (A)

اح يتغير بعدين



ال critical Action هو

ال eting لانه التعديل

على متغير مشترك هو اخذ

العود)

بعد ما نخلص اح يعمل Signal(A)

Signal(B)

كود سلايد 40 لكل ال philosophers بس بغير ال index تبع العود

do ؟

العود على اليمين

wait(chopstick[i]) ; على اليسار

wait(chopstick[(i+1)%5]) ~~cycle~~ 5% عشان دائرية

// eat

Signal(chopstic[i]) ; اجمعهم

Signal(Chopstic[i+1%5]) ;

* عدد ال semaphores الكلية لهاي الحالة هو 5

عدد الكودات = عدد اعتنافين

عدد ال semaphores = عدد ال shared resources

(Sem-distroy تلغي ال semaphore عشان تمسحها من ال memory

الممسوحة ضوئيا بـ CamScanner

Ch7: Deadlocks

Date 24/8/2025

* ال deadlock هل هو مشكلة ولا خدمة ؟ هو مشكلة

* اعشاكل :

① process synchronization

② deadlocks

* خدمات يقدمها ال OS :

① process ② threads ③ scheduling

- سبب ال deadlock نفس سبب ال process synchronization :

① كثرة ال processes ② قلة ال resources

① يعني ال deadlocks ما بكون موجود نهائياً لو ال processes قلّوا وأصبحوا process وحدة

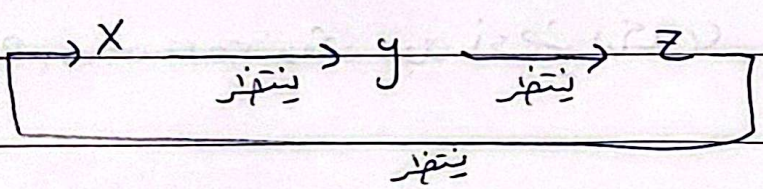
② أو لو زدنا عدد ال resources لـ ∞

* اكل بال process synchronization : انه نعمل critical section وننظم الوصول إليه وما يصير في paralelizem

* اكل بال deadlock اختيار vectime اكل اختيار واحد تعمله

Preemption

عشان الباقي يمشوا



الممسوحة ضوئيا بـ CamScanner

Date Page

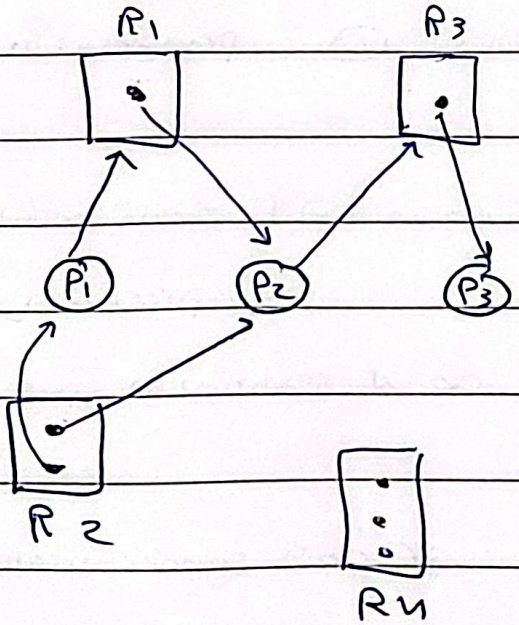
* عشان توصف النظام إلي فيه مشكلة الـ deadlocks لازم وجود ٣ أشياء :

① Processes أو threads (any execution unit)

② resoureces : أي resource بدك ياه keybord , i/o file , memory , cpu (R_1)

③ عدد الـ instances يعني اذا R_1 كان cpu بالتالي عدد الـ instances إلي هو عدد الـ cpu's واذا كان memory يعني كم وحدة عندك منها

* هدول بساعدونا عشان نرسم (Resource allocation graph)



① P_1 , P_2 , P_3 (process) حالتهم running بما انهم بتنافسوا على resource

② المربعات هي الـ resources

③ النقطة هي عدد النسخ من هذا الـ resource

* بنركز على هدول الـ ٣ شغلات لانهم مقومات اعتقلة إلي عنا (لو process بدهم resources وفش مجال أعطي الكل)

الممسوحة ضوئيا بـ CamScanner

Date Page

* آليه استخدام أي resource :

نفس نمط ال semaphore :

① request (بال semaphore كان اسمه entry section)

② use (critical section)

③ release (exit section)

* فأي resource بدك تستخدمه بدك تشيك اذا موجود أو لا

اذا لا بدك تنتظر اذا متوفر تنتقل لمرحلة استخدامه

وبس يخلص تبرك ال resource (release)

* مش بصير starvation أو deadlocks ؟ لو واحد ماخذ resources

وبستنى بـ resources تانين (مش عامل release)

* لحصول ال deadlocks لابد من وجود ٤ شروط و اذا لم يتحقق

أحدهم لا تحدث المشكلة :

① mutual exclusion ② Hold and wait

③ no preemption ④ circular wait

الممسوحة ضوئيا بـ CamScanner

Date Page

① Mutual exclusion : الوصف هون يخص ال resources
فقط لـ process وحدة تستخدم هذا ال resource وممنوع
حدا تاني يجي ويتشاركوا معها ، منطقي لحدها لانه
فرضاً عملية ال reading ماني مشكلة لو تشاركوا اكثر من وحدة
* (الافتراض هون انه معظم ال resources هي مش قابلة للتشارك
الا اذا ذكر غير ذلك)

② hold and wait : عندك اشي وتنتظر اشي : عندك اشي
عشان تكون شغال وتقدر توصل لمرحلة تطلب غيره وال wait عشان
تكون تنتظر واحد تاني .
- لو كل اشي بدك ياه hold (يعني عندك)؟ ماني deadlock
- لو انت فقط wait ؟ ماعندك اشي بالتالي فش حدا بستناك
لذلك انت مش بال deadlock

③ no preemption : لو انت بتستنى واحد بس هو اجا عملك
(kill) اح ينتهي ال deadlock ، ماني واحد رح نتخلص منه عشان
نخدم الباقي عشان يضل ال deadlock موجود

④ Circular wait : $P_1 \xrightarrow{wait} P_2 \xrightarrow{wait} P_3$ هون ماني
deadlock لانه P_3 هو hold فقط فرح يكمل شغل ويترك ال resource
وياخدهم P_2 ويخلص منهم وبعدين P_1 ، في deadlock لو
P_3 ينتظر اشي من P_1 أو P_2

الممسوحة ضوئيا بـ CamScanner

Date Page

* كود سلايد 16 :

عندك work-one و في work-two

بخلي ال first-mutex = 1

بالتالي ال lock مش

رح تكون شغالة ورح

يمشي

رح تعمل Block لـ second-mutex

(حصل على ال first و

ينتظر ال second)

بخلي ال second-mutex = 1

وهون كمان ال lock مش شغالة

ورح يمشي

رح تعمل Block لـ first mutex

(حصل على ال second وبستنى

ال first)

* بما انه الثنين بستنوا اشي من بعض معناته

عندك deadlocks

* كل واحد لحالة عنده starvation مما نطلع عليهم مع بعض بنحكي

عنه deadlocks

* deadlock وصف لمجموعة

starvation وصف لفرد

* عشان تشوف ال deadlock

استخدم resource allocation graph

* سلايد 9 : الرسمة هاي ال OS يعملها on demand (لما يجب يشوف

في deadlock أو لا) وهي ليست شيء مخزن زي ال PCB

الممسوحة ضوئيا بـ CamScanner

Date Page

برسمة سلايد ١٩ عن ٤ شغلات

① processes (P)

② resources ← عدد النسخ

③ arrows / edges

← request edge (طلب) (طالع من ال process رايح لل resource)

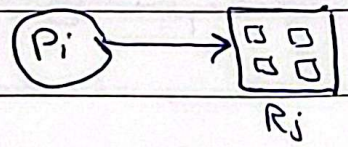
لا يدل على انه ال resource انحجز (wait condition)

← assignment edge (طالع من ال resource رايح لل process)

يدل على اعطاء ال resource

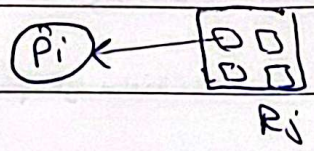
* لاحظ سهم ال request مابكون داخل على مربع ال resource

(مابكون رايح على نسخة معينة)



request

Pi يسعى للحصول على اي نسخة من ال resource



assignment

Pi حصلت على one instante من ال resource (Rj)

الممسوحة ضوئيا بـ CamScanner

* عشان نشوف الـ deadlocks بدك تشوف كل الـ processes ايش وضعهم

الرسمة سلايد ٩ :

① بتقدر تشوف خاصية الـ mutual exclusion : اذا كل نقطة • طالع منها سهم واحد معناته في اما اذا طلع منها أكثر من سهم معناته ما في وبتشارك فيها أكثر من process

② hold and wait : بتقدر تشوفه على الرسمة برضه

مثلاً P_1 ماخذه نسخه (hold) من R_2 وبتستنى بنسخه من R_1 (wait)

③ No preemption : ما بقدر أشوفه ، هو بحكي عن امكانية process تتعدى على process ثانية وتاخذ منه الـ resource وهذا الاشي إلي ما بدناياه حتى تتكمل شروط وجود الـ deadlocks

④ circular wait : بقدر أشوفه عن طريق انك تمشي من أي instante لـ resource معين اذا بتقدر ترجع لعماي النقطة معناته في deadlocks ، اشرط انك تمشي مع اتجاه الاسهم

* $P_1 \to R_1 \to P_2 \to R_3 \to P_3$ ماخي deadlock لانه لم تغلق الدائرة

الممسوحة ضوئيا بـ CamScanner

Date Page

* للتسهيل ممكن نشوف شرط ال circular wait عشان نعرف اذا في deadlock او لا :

- اذا مشيت من نقطة وقدرت ترجعلها يمكن في deadlock

- اذا مشيت من نقطة وما قدرت ترجعلها على الأكيد ما في deadlock لان شرط ال circular لم يتحقق

- مش كل circale تعني وجود deadlocks على الأكيد

- لابد ال : امشي من $P_1 \to R_1 \to P_3 \to R_2$ هاي دائرة

مشكلة P_3 انه ينتظر R_2 و R_2 محجوزه لـ P_1 و P_4

P_4 لا يوجد ما يمنع انها تكمل شغل وتخلص (hold only) ماخذه كل اي بدها ياه (قد التنفيذ) لما تخلص تماماً رح تحرر ال resource ورح ياخذه P_3 ويصير P_3 (hold only)

وبالتالي لا ينطبق عليه ال ٤ شروط كاملة بالتالي الدائرة هاي مش deadlock

* لاحظ P_2 برضه (hold only) لما يخلص رح ياخذ P_1 ال resource وبرضه رح يصير (hold only) ورح تنفك الدائرة

* الدائرة بتوحي اذا في deadlock في حال كانت شاملة كل ال processes اما اذا في أكم process عادية بتكون منفذ لفك الدائرة

الممسوحة ضوئيا بـ CamScanner

Date Page

سلايد 10 : $R_2 \to P_1 \to R_1 \to P_2 \to R_3 \to P_3 \to R_2$

لاحظ هاي دائرة ، همون في deadlock لانه كل ال processes مشمولين بالدائرة وفش حدا يمشيهم زي اشكل السابق

P_1 ← hold من R_2 و wait لـ R_1

P_2 ← hold من R_1 , R_2 و wait لـ R_3

P_3 ← hold من R_3 و wait لـ R_2

اشكل هذا ينطبق عليه الشروط الاربعة

الحلول :

① ignore the problem : هذا لا يعتبر حل ، في بعض المرات بتحس انه ال system عندك علّق مرات ال OS بتجاهل الموضوع وبعمل restart (يعني kill لكل ال processes وبالتالي تخلص من مسبب المشكلة كامل)

② ensure the deadlock never happen ← صحيحة لو ارتبطت مع ال prevention

- prevention : تسعى تخلي واحد من الشروط الـ 4 ما يتحقق بالتالي مستحيل يصير deadlocks

- avoidence : حاول تتجنب احد الشروط ممكن تنجح مرات وتفشل مرات بعده اعمل $\boxed{\text{detection}}$ عشان تكتشف اذا صار لا ارسم ال resource allocation graph) وشوف اذا في دائرة اذا في اعمل $\boxed{\text{recovary}}$

الممسوحة ضوئيا بـ CamScanner

Date Page

③ عشان تعمل avoidance بدك تخلي ال system في safe state (ما بصير فيها deadlock) زي سلايد 21

ال deadlock بصير بال unsafe

طول ما انت بال safe وبتمشيك انه ما في دائرة امورك تمام

اما اذا لمحت دائرة تكون انتقلت لل unsafe state

وهذا لا يعني تأكيد وجود ال deadlock لانه زي ما قلنا مش كل دائرة يعني deadlocks

* اكل إلي رح نحكي عنه مكون من ٣ أشياء:

① try to avoid the deadlocks

② detection

③ recovery

وهدول خطوات ال avoidance

* رح نشوف ليش ال prevention ليس اكل اكتبع ع انه الاسهل لانه ما في داعي تراقب لانه ما رح يصير بتلغي ال deadlocks عن طريق عدم تنفيذ أحد شروط الأربعة

① لو بدنا نلغي ال mutual exclusion (شوف كم رح تكون التكلفة)

لو بدنا نخلي أكتر من process تستخدم نفس ال resource بنفس الوقت

رح يصير في data inconsistency ، يعني حليت ال deadlocks ورجعت لمشكلة

ال process synchronization ، هو أحد الحلول ولكنه مكلف ويسبب مشاكل أخرى

الممسوحة ضوئيا بـ CamScanner

* لو بدنا ننفي الشرط الثاني (hold and wait) يا بنخليهم كلهم hold

بدك توفر عدد كبير من ال resources حتى يكفي وهذا غير منطقي لانه دائماً عدد ال resources محدود

او كلهم wait

كلهم رح يصير لهم Starvation

- لذلك هذا نظرياً هو حل ولكن تطبيقياً ليس بحل ورح يترتب عليه

① low resource utilization (if wait only)

② high possible of starvation

* لو بدنا ننفي الشرط الثالث (no preemption)

لو كان : $P_1 \xrightarrow{wait} P_2 \xrightarrow{wait} P_3$ (wait)

لو خلينا في preemption بصير عادي ناخذ من P_2 ال resources ونعمله kill عشان تنفك الحلقة

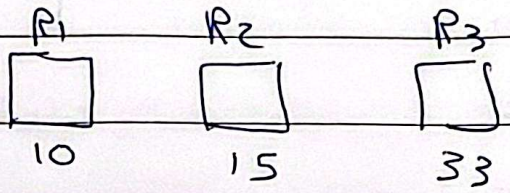
* هون رح نخسر P_2 بإنجازها وهون ال resource utilization ممكن نعتبره صار بالسالب واذا دائماً رح تختار نفس الضحية رح يصير له Starvation

الممسوحة ضوئيا بـ CamScanner

Date Page

* لو فكرنا بمنع الشرط الرابع (circular waiting)

عن طريق ترقيم كل resource برقم عشوائي



وخلي كل process لما تطلب resource لازم يكون اعطلوب رقمه أعلى

من إلي قبله يعني اذا بدك تطلب R2 بعد يعني بتقدر تطلب

R3 فقط (ما بتقدر ترجع رجوع)

↓

$P_1 \rightarrow P_2 \rightarrow P_3$

هذا الخط بصير ممنوع وبالتالي ما في دائرة

* هو كحل موجود ولكنه غير منطقي لانك هيك بتكون بتمنع

process من طلب resource معين (قاعد بتتحكم بأدائها) وهيك

ممكن توقف ال process ويصير لها starvation وما أخذ من ال resources

لن يستغل

- بالنهاية أي prevention أثرها :

① low resource utilization

② starvation لذلك هذا خيار

③ data inconsistency غير محبب

الممسوحة ضوئيا بـ CamScanner

Date Page

* حل ال avoidance هو منطقي أكثر، ويعني أي شيء ممكن يقود لل deadlock نحاول نمنعه ولكننا لا نضمن عدم حدوث ال deadlocks على الاطلاق.

- بالتالي لما تعمل استراتيجية مبنية على ال avoidance لازم تضل تراقب هل هاي الطريقة نجحت أو فشلت.

- ال avoidance يستند على ايش بالزبط:

1. لازم تكون تعرف ال (available resources) متاحين ما حدا حجزهم
2. allocated resources (المحجوزين) ومين حجزهم
3. maximum demands: ال process إي أخدت من ال allocated هل بدها كمان ولا لا. (ممكن ينطلب مستقبلاً)

* بقدر أعرف المأخوذ وإي مش مأخوذ ولكن كيف بدي أعرف المتوقع طلبه

* ممكن أعرف المتوقع طلبه عن طريق scann سريع على كودها وتعرف ايش بتعمل وايش رح تحتاج

بدك تضلك تعمل tracking لل resources لانها معلومات متغيرة الاشي المحجوز هسا يمكن كمان شوي يرجع وممكن

الممسوحة ضوئيا بـ CamScanner

Date Page

* ال OS بعمل طريقة ال avoidance ← on demand يعني
مش دايماً بضل يشيك ، لما يصير في طلب جديد بحط ال ٣
شروط و بتفرج عليهم

* الطريقة كالتالي :
أي process بتطلب resource بدنا نعمل اشي اسمه check statment
اذا ال request تبعها بحافظ على ال safty لل system ممكن
نسمح فيه أما اذا بخليه unsafe رح نرفضه أو prevent

* رح نحط قداعنا ال available وال allocated وال future
الآن اجاك request جديد بدك تشوف حسب الوضع إلي قائم ، النمشيه
ولا لأ ، اذا تحققنا من الطلب وانه بخلي النظام unsafe لن نقوم
بالاجابة عليه ، أما اذا بخلي النظام بال safe state بنقدر نجاوب عليه

* اذا جاني مثلاً 3-request طلب لاستخدام resource معين
ما بنقع كلهم مع بعضنا ، رح أصير أمشي واحد واحد
ورح نعمل اشي اسمه ال safe sequence
$\langle P_1 , P_2 \rangle$

* لو مشيت P_1 وحيدة أولاً بتقدر تنجز شغلها باستخدام ال resources
المتوفرة ولن تحتاج غيرها

* عند انتهاء P_1 وتركها لجميع ال resources ممكن P_2 تستفيد
من إلي كان متاح ومن إلي صار متاح بعد ما تركته P_1

الممسوحة ضوئيا بـ CamScanner

Date Page

* < , , Pn > بنحاول نرتب ال process هون

كالتالي:

① إلي بده resources قليلة ومعكن نجاوب عليها بالايجاب أدلة" بحطة بالبداية

② Pn المتأجلة للآخر هاي بكون بدها كثير resources فحطيناها

بالاخر عشان لي يجي دورها يكون كل ال processes إلي قبلها تركوا

ال resources

* We have 2- avoidance algorithms version 1 *

and version 2: (الناتج بتبعهم واحد لنفس الاشي)

والهدف منهم دراسة الطلب في حال يبقي النظام safe فيوافقوا عليه

أو يحوله لـ unsafe فيرفضوه

* Alg version 1 *

(1 copy)

تستخدم اذا كان في Single instance في كل resource : ⊡

- اذا لقيت أحدهم عنده 2 copys أو أكثر لازم تحول على Banker's Alg

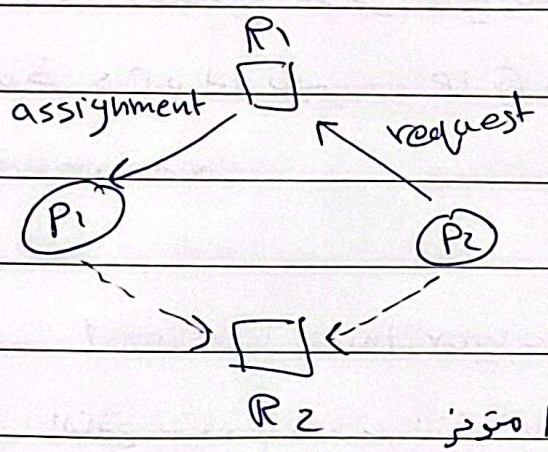
resource type : ١ مربع ككل ال instance النقطة داخل ١ مربع

الممسوحة ضوئيا بـ CamScanner

Date Page

* خيار رقم واحد: resource - Allocation graph وهاي تختلف عن اي ذكرناها سابقاً لانه السابقة كانت فقط بتشوف اذا في ال deadlock يعني ما بتعمل avoidance

اما هون زي سلايد 24 :



ح تلاحظ وجود خط متقطع

قلنا بدنا ٣ معطيات عشان نعمل avoidance :

① available : مثلاً R1 محجوز , R2 متوفر
مش طالع منه سهم

② allocated : زي R1

③ future : الخط المنقط

* P2 could request R2

لذلك P2 حالياً طالبة R1 (wait)

و P2 ممكن تطلب R2 (No wait)

* السهم مارح يضل منقط طول ال life time بعد شوي لما ينطلب ح نعمله سهم متصل وهيك بكون طلب وتحول wait اذا وافقنا عليه ح ينعكس اتجاه السهم ويصير طالع من ال resource ومتجه لل process

* ال avoidance هون بكون كالتالي : بنجرب اذا حولنا السهم المنقط هل بتحول اتشكل لدائرة اذا آه معناته هذا السهم critical ولازم تمنعه

الممسوحة ضوئيا بـ CamScanner

Date Page

* سلايد 24 : لو أجبنا على طلب P2 رح يتحول اتجاه السهم وحيك
رح يصير في cycle محتملة نتيجة تحول اتجاه السهم و طلب
P1 (حتى لو جاوبناه بالرفض) ولكن تحقق عناصر الـ hold and wait

السهم اعنقطة اسمه (claim)

* لو P1 طلب بالأول و وافقنا عليه رح يتحول اتجاه السهم ولكنه لن
بسبب أي دائرة بالتالي فش خطر ولكن لو P2 طلب رح نرفض

كلمة لو طلب يعني تحول الـ claim لـ request

بالتالي الآمن الآن انه يشتغل P1 بعدين P2

safe sequance $\langle P_1, P_2 \rangle$

ممكن يشتغل بالأول

مش ممكن يشتغل بالأول

request i (granted) يتحول لـ assainment

بدون ما يعمل دائرة معناته آمن ومسموح

الممسوحة ضوئيا بـ CamScanner

Date Page

Banker's Algorithm

* انسى كل الكودات إي بالسلايدات وركز هون:

عشان نشتغل على هاي الطريقة ح نحتاج 4-array

① Allocation ② Max ③ available ④ need

لاحظ سلايد 31 :

Available

A	B	C
3	3	2

single dimension

حاطلك كل resource عدد النسخ إي ضايله

باقي من A ← ٣ نسخ

ومن B ← ٣

ومن C ← 2

هاي المعلومة لو ما اعطاك

ياها بتقدر انت تطولها عن

طريق : تشوف عمود الـ allocation

وتحدد كل process كم اخذت وتجمعهم

وتطرحهم من العدد الكلي بالسؤال

افترض A = 10

B = 5 , C = 7

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

$10 - (2+3+2) = 3$

$5 - (1+1) = 3$

$7 - (2+1+2) = 2$

* الـ Max array ما بقدر

أحسبها لانها توقع

للمستقبل

الممسوحة ضوئيا بـ CamScanner

Date Page

Max

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

هذا معناه: أتوقع خلال
الـ life time لـ P0 مثلاً
انها تطلب بالكثير 7 من A
و 5 من B و 3 من C
ولن تتجاوز هذا العدد تحت
أي ظرف من الظروف

وهذا العدد لا يدل انها رح تطلبهم
ولكنه الحد الاقصى إي ممكن يصير

بناء على ذلك ممكن نحسب الـ
Need (إي ضايل بدها ياه)

	allocation A B C	Max A B C	Need A B C
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

- 2) لحسبة عن طريق: P0 مثلاً كم أعلى حد من A بدها ياه؟ 7
طيب كم آخده؟ (0) لذلك Need هي (7-0=7)
- طيب كم بـ أعلى حد لـ B = 5 وكم آخده = 1 بالتالي 5-1=4
- طيب كم أعلى حد لـ C = 3 وكم آخده = 0 بالتالي 3-0=3
وهكذا لكل الـ process

الممسوحة ضوئيا بـ CamScanner

Date Page

* فكرة الـ avaidance انه تحاول ما تشغل جدول الـ 5-process مع بعض حاول شغلهم واحد ورى الثاني عشان نطلع الـ safe sequence : يعني من ضمن الـ 5 اعوجدوين هل في واحد من الآمن تشغله بالبداية (تحتاج تطول الـ Need عشان نعرف تجاوب) (الـ max خادع شوي)

* رح نتطلع على الـ Need فقط ونحط جنبه الـ available ونصير نقارنهم مع بعض

	Need A B C	available A B C
P0	7 4 3	3 3 2
P1	1 2 2	
P2	6 0 0	
P3	0 1 1	
P4	4 3 1	

* اختار الـ process الـ resource إلي ممكن تطلبهم موجودين :

1. P0 بقدرش اختارها لانه من البداية بدها A=7 وفش غير 3
2. P1 بقدر أشغلها لانه بدها A=1 وفي 3 وبدها B=2 وفي 3 وبدها C=2 وفي 2
3. P2 لا يمكن لانه A=6 وفي 3
4. P3 ممكن لانه إلي بدها ياه موجود
5. P4 لا لانه بدها A=4 وفي 3

* الآن صار ممكن أبدأ بـ P1 أو P3

* عشان ما تجرب كتير اختار الـ P إلي بنهاية شغلها بترجع أكبر عدد من الـ resources

الممسوحة ضوئيا بـ CamScanner

* لو اجيت اخترت P_1 هل هو الخيار الأمثل ؟ بدك تشوف عند انتهاء P_1 كم رح ترجعلك $A=2$, $B=0$, $C=0$

طيب P_3 كم رح ترجعلك $A=2$, $B=1$, $C=1$

* كون ذكي بالاختيار ولاحظ انه resource A عليه طلب أكثر فاختار الـ process إلي بترجعلك أكبر عدد من الـ A ، طيب هون P_1 , P_3 برجعوا نفس العدد بالتالي ممكن تنتقل لـ B و C تشوف مين برجع أكثر.

الأفضل تبلش بـ P_3 (يجب الذكر انه لا يوجد حل واحد فقط)

	Need A B C	Available A B C
P0	7 4 3	3 3 2
P1	1 2 2	
P2	6 0 0	
P3	0 1 1	
P4	4 3 1	

رح نبدأ بـ P_3 :

تتقدر (0 1 1) A B C

تنفذ بالمعطيات

اعد جدده (3 3 2) A B C

بس تخلص رح ترجع (2 1 1) A B C

فيصبح اعد جدد (5 4 3) A B C

لو P_1 بعدها ، تحتاج (1 2 2) A B C وفي (5 4 3) A B C اذاً رح تتنفذ وبس تخلص بترجع (2 0 0) A B C فيصير اعد جدد (7 4 3) A B C

لو P_2 بعدها : تحتاج (6 0 0) A B C وفي ↑ بالتالي تتنفذ وبتخلص بترجع (3 0 2) A B C وبصير الناتج (1 0 4 5)

وهكذا لحتى تطلع الـ safe sequance (أو أمشي بترتيب اسلايد)

32

الممسوحة ضوئيا بـ CamScanner

Date Page

* بالتالي هاي ال Aly بتحكيلك اذا عملت run لعدول ال process بناء على ال safe sequence رح يكون الوضع تمام وفش خطر

* هذا كله لا ينهي المشكلة

* بعد ما جهزنا ال system وأعطيناه ال recommendation safe sequence

- هسا لو اجاك request هل بتقدر (grant) بعدين

مثلاً اجاك (3,3,0) (A B C) وهي عندك هاردا (2 3 0) (A B C) لم توخذه

بالتالي ما رح تاخد هذا ال request وما في داعي ترجع تعمل rerun لحلك السابق لانه الارقام المطلوبة غير متوفرة أساساً

اجابة أول سؤال سلايد 33 ↑

- قارن ال request مع الموجود اذا الموجود أقل رح نرفض الطلب

السؤال الثاني : (Po)

اذا طلب (0,2,0) والموجود (2,3,0) يدل على اشي ممكن نبلش فيه .

Po كم ممكن تطلب من A ؟ حقاً 7 ، ومن B=4 ومن C=3

وهو حاكيلك انها بدها تطلب (0,2,0) بالتالي مسموح وبصير

ال need ← (7,2,3) بالتالي مبدئياً رح نوافق على الطلب

لحتى نرجع نعمل rerun لـ Aly اذا ضله safe رح نوافق

* لو وافقنا رح يصير المعطيات كالتالي :

Allocation	need
0 3 0	7 2 3

ومن هون اعمل ← rerun اذا لقيت sequence معناته آه ، الدكتور جرب وقال بنفع :)

الممسوحة ضوئيا بـ CamScanner

Date Page

* بالآخر هاي ال Alg بتحاول تعمل Avoidance تتجنب ال deadlock .

* اذا حصل ال deadlock نتيجة انه ال Alg اعاضيه ما نجحت اجراءاتها أو صار في خطأ و انعطى grent ل request ودى ال system للـ unsafe

ايش رح نعمل ؟؟ وكيف رح نكتشف ال deadlock

Detection عن طريق resource allocation graph (الاصدار الأول)

وبشوف اذا في دائرة أو لا

زي سلايد 36 شكل (a) لاحظ في أكثر من deadlock

اما داخل المثلث القائم أو حتى ع الشكل ع

* من شكل (a) رح يطلع عنا اشي اسمه (wait-for graph) للتسهيل على القارئ فقط

بشيل ال resource وبحلي الاسهم بين ال process على طول

ولكن بشرط :

① وجود one instante (single copy) بكل resource

* اذا وانت تعمل ال safe sequence قدرت تخلص كل ال process معناته آمن وفش deadlock أما اذا وقفت بالنص ومش عارف تكمل

وجربت كل الخيارات معناته في deadlock

الممسوحة ضوئيا بـ CamScanner

Date Page

* قلنا انه أي رسمة ال OS ما بضل يعملها tracking وأصلاً هي extensive انك تعملها وهي رسمة قابلة لتعديل كل شوي ومش ثابتة لذلك ال OS ما بخزنها بـ data structure وعمل هذا ال graph بضيع من وقت ال cpn

- ما في اجابة معينة على كل كم ال OS بعمل construction (بناء) لهذا ال graph ولكن ممكن تكون الاجابة حسب ال system وكل كم ال deadlock بظهر

* مثلاً جهازك بالبيت هل هو عرضه لل deadlock كثير؟؟ اذا قديم آه اذا جديد احتمالية صغيرة

* اذا عندك ال many process رح تزيد احتمالية ال deadlock ويتصير اعراقبة أكثر ولازم ترسم الشكل أكثر (resource allocation graph)

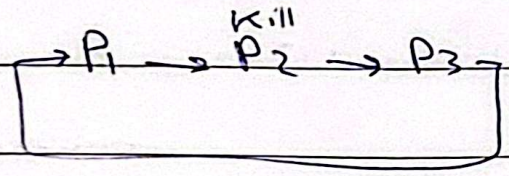
كنا ماشين بـ 2 phases :

① Banker's Alg (avoideance)

② detection تشوف اذا نجح أو لا عن طريق (wait-for) أو

③ اذا في deadlock بتكون ال Alg فشلت

④ اكل ال process termination (Kill)



لو عملت Kill لـ P2 رح تنحل مشكلة P1 لانه بستناه وبعدها P3 بتنحل مشكلته برضه

الممسوحة ضوئيا بـ CamScanner

Date Page

* ال process termination تكلفته أعلى من انك تعمل resource preemption لانه P2 ممكن يكون عندي resources من A B C D ، والمشكلة ممكن تكون قلة B, C فرضاً فانت لما تنهي ال process كلها انت عملت حل شوي extreme فممكن شوي تعمل relaxation لهذا كل وتروح لمنحنى أقل تكلفة بحيث تشوف المشكلة كانت على B, C مثلاً فخذهم هم بس شوي وارجع رجعهم ، هذا ما يضمن حل المشكلة ولكنه بحاول

* لو رجعت لمثالنا السابق لو صارت هاي المشكلة بكون سببها A لانه عليه طلب كثير

* الفرق بين الحلين انه انهاء ال (P) برجعها لنقطة الصفر أما الحل الثاني لا زالت running ولكن فعلياً أخذنا منها بعض ال resources ، فرضاً أخذت B, C انت ما برجعها لنقطة الصفر انت لبرجعها بس بمقدار B, C

الممسوحة ضوئيا بـ CamScanner

Date Page

Ch8: Main Memory

* ال main memory أو ال RAM رح نحكي عنها من منظور ال OS
مش انها HW
يعني ال OS كيف بشوف ال memory وكيف بتعامل معها كيف
بنسق عملها بحيث انه يستغلها بأفضل شكل ممكن

* أي برنامج قبل ما يكون موجود بال memory بكون في منه نسخة
بال harddisk واسمها Application ومجرد ما نضغط عليه مرتين
ويشتغل رح يتحمل منه نسخة بال main memory ويصير اسمه process

* ال OS يدير ال process وال cpu وال mainmem
- أي برنامج بتكتبه من خلال ال addressing mode إلي أخدناهم بال assembly
بتقدر تستخدم 2-typ's من ال storge لهذا البرنامج يا إما
ال main memory ، او حطينا العنوان داخل أقواس [ ] يصبح
يصبح located داخل ال mainmemory
بالمقابل اذا بتستخدم regester زي Ax , SI , DI

اذاً ال 2-storge هم ال ① main memory ② Regesters

* رح يصير لازم تعمل trad-of بين ال regesters المعروف
عنهم السرعة العالية والتكلفة العالية وال main memory المعروف
عنها الاداء الأقل و السعر الأقل

الممسوحة ضوئيا بـ CamScanner

Date Page

عشان يصير عنا Balance بين ال cost و ال speed

بكون عن 2types
of storge

* الواقع جدول مش النوعين الوحيدين في عن secandary storge

زي ال hard disk و ال caches ولكن كمبرمج

ما بتقدر تكتب كود يخزنلك عليهم الا اذا بتستخدم

ال streem على ال harddisk

* ايش بقدر أخزن داخل ال main memory :

① Data ② instrechions

بجميع الحالات انت بتزود ال memory ب address ويتبع

ذلك إما عملية reading أو writing

↓ داتا مجهولة بدي أقراها

↓ داتا معروفة بدي أقراها

* إذاً ال memory تحت الطلب ممكن تعملك 2-oprations

① reading ② writing

* الآن ما يميز ال addressing modes سواء إلي بتعامل مع ال regesters

أو مع ال main memory ، انه أي regester opration بالعادة تتم ب

1 clock cycle (كثير سريع) أما ال main memory لا تتم

ب 1 clock cycle بدها أكثر من cycle عشان تخلص

الممسوحة ضوئيا بـ CamScanner

Date Page

* لما نحكي many cycles معناته ال cpu رح تتعطل وتتوقف
وهذا التوقف بنحكي عنه stall
stall يعني انه ال cpu قاعد يعمل pose حتى تيجي المعلومة
من ال memory أو حتى تنكتب عليها
* بالتالي ال memory بطيئة مقارنة بال cpu وبالتالي ال cpu
اذا احتاج معلومة منها رح يتعطل

* الفكرة انه ال os من وجهة نظرة الأولى لازم بالنسبة
لل memory يعمل اشي بخص السرعة لازم يعمل
speedup لل memory opration حتى يصير في combatibility
بين سرعة ال cpu وسرعة ال memory

* طبعًا هناك حلول أخرى انه ال cpu يتعامل مع ال cache
هو فعليًا انت بتكون بتستخدم ال cache بدون ما تعرف
انت لما تعطي أحواس زي هيك [ ] انت ما بتجبر ال os
يستخدم ال cache هو بستخدمها زي ما بده

* اذًا ال os لازم يعمل اشي بخصوص ال speed وكمان
بخصوص ال protection ، لانه حجم ال memory محدود
وما بقدر ال process ياخد زي ما بده مساحة وتخزين بدون
تدخل ال os لمساحة ال memory قليلة باعتبارنه مع عدد ال process
اللي بحاولوا يستفيدوا منها

الممسوحة ضوئيا بـ CamScanner

* ال OS لازم يعمل التالي لل memory :

① speed up ② protection ③ area utilization

↓

بين ال processes

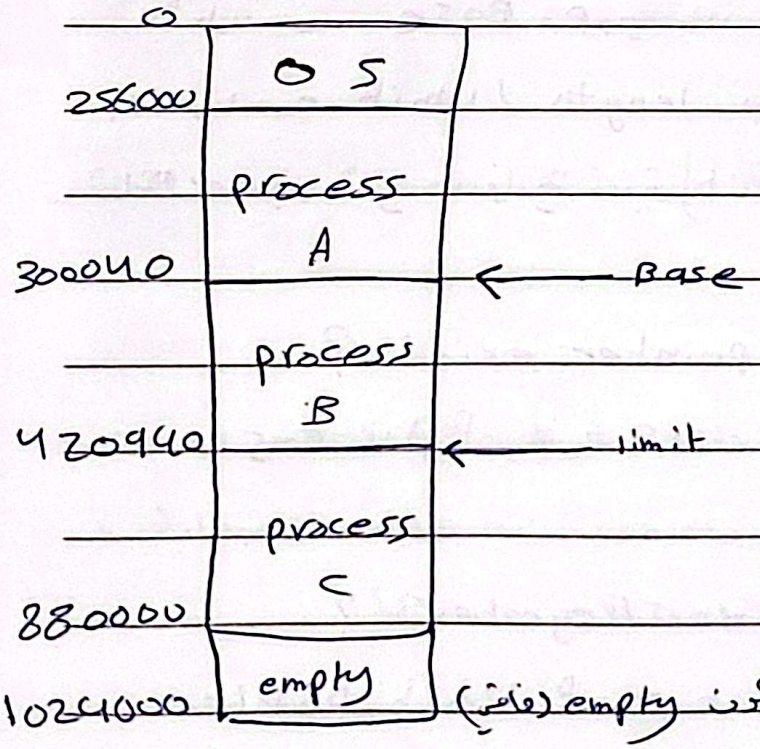
أو ال threads , ال memory ك backup

بتستخدمها جميع ال execution unit

Base and limit Registers :

هون بنحكي عن ال protection وكيف ال OS بضمن انه process معينة

ما تتجاوز المساحة الخاصة فيها وتدخل على مساحة process ثانية



التشكل هذا بوصل معلومتين

① ال address الصغار بالرقم

بكونوا محجوزين لل OS

وانت ما بتقدر تستخدمها

او تخزن عليها

② ما تبقى من ال memory قد يكون empty (فاضي)

او مليان بـ processes

الممسوحة ضوئيا بـ CamScanner

Date Page

لاحظ بالرسم وجود 3 - process لانه زي ما حكينا بال multiprograming

ما بكون في process وحدة بال memory

مهام ال OS :

① يمنع جميع ال process يتعدوا على مساحة ال OS (0 - 256000)

② A process لازم ال OS يضمن انه اي address بتحاول

تقرأ منه أو تكتب عليه محصور بين اعطيها العما (256000 - 300040)

وممنوع هاي ال PA تحاول تتجاوز هاي المساحة لانه بصير violation

③ ال OS يضمن هذا الاشي عن طريق استخدام 2 pointers

الأول اسمه Base وهو يحدد ال starting point ك address

الثاني اسمه limit / length بعرفك حجم ال process

لذلك بتقدر تعرف النهاية وبث (Base + limit)

* How many Base pointer exist ?

# of Base pointers = # of processes

* How many limit pointers/regester ?

# of process = # of limit pointer

الممسوحة ضوئيا بـ CamScanner

Date Page

read writ

* كل عملية request لازم الـ cpu يتأكد انها من ضمن الـ Base والـ (limit + Base)

* شكل سلايد 6: يوضح كيف الـ Base مع الـ limit بتعاونوا بحيث يضمنوا أي address بطلع من الـ cpu بالـ user mode (يستخدمه اعبرمج) بيكون صح.

① أي address بطلع من جهازك لازم يكون أكبر من أو يساوي الـ Base (اذا لا كأنك بترجع رجوع لورى بالـ address وبتتعدى على مساحة غيرك) فبصير exeption / trap / error

② اذا كان الـ address ≤ Base رح نشيك انه لا يتجاوز الحد الأعلى Base+limit ,

* اذا كان الـ address أصغر من الـ Base أو أكبر من الـ Base+limit رح يصير عنا (error) يحتوي على: انه الـ process بتحاول تحمل access لـ address مش من ضمن حدودها

* سلايد 6,5: حكى كيف تحمي process من process ثانية وكيف تحمي الـ OS من الـ process

الممسوحة ضوئيا بـ CamScanner

Date Page

Binding : معمعة يقوم بعمل الـ OS (تحويل الـ address إلي بطلع من البرنامج من address غير حقيقي إلى address حقيقي يمكن استخدامه مع الـ main memory

* زي ما بنعرف الـ address نوعين :

① physical
↓
وقت التنفيذ يتحول الـ logical إلى physical لأنه الـ memory ما بتفهم غير physical

② Virtual / logical
أي address إلي بصدرة اعبرمج وهو غير حقيقي عشان نعمل هي اعبرمج زي لما نحكي int x ; احجز مكان لـ (x) واعرضة & x هذا العنوان ما بساعد الـ cpu توصل على اعكان الصحيح بالـ memory

* لذلك معمعة الـ OS انه يعمل Binding (ربط) يربطلك الـ address إلي صادر من البرنامج (Virtual) مع الـ physical address إلي الـ memory رح تخدمك فيه

* عملية الـ Binding لا يمكن الاستغناء عنها والعملية حمالي بتصير لأي اشي بدك تجيبه من الـ memory مثل الـ instrections اعد جددين بالـ code segment والـ Data اعد جددة بالـ memory

الممسوحة ضوئيا بـ CamScanner

Date Page

أي logical address (Base ; offset) كنا نعمله Binding

عن طريق Base 0 + offset ← physical

↓

data/codesegment

* كل address حسب شكله وصيغته بصير له Binding مختلف

① compile time binding : عملية التحويل بتصير قبل ما البرنامج يعمل run

② load time binding : لما يتحمل البرنامج بال memory يتم تحويل ال addresses

③ execution time binding : ما بنعرف ال logical من ال physical الا وقت التنفيذ

* مش ال os اي بتبع برنامجك بقدر يحول ال address بتعك انت كمستخدم من logical لـ physical بناء على الحالات الثلاث

① اذا كنت عارف ال address اي بدك تستخدمه يعني مستخدم constant address بقدر يحولك ياه at compile time

زي لما كنا نكتب ORG 1000

absolute code ← كأنك بتحكي لـ os انه بدك تبلش من عند موقع 1000

لو تحمل run أكثر من مرة ال address ما رح يتغيروا بسبب ال ORG

اذا مش موجودة بتغير مكان الفارين

الممسوحة ضوئيا بـ CamScanner

Date Page

② مرات كنا نحكي ADD Ax , [Bx]

العنوان إلي هون ما بتقدر تعرفه الا اذا أخدت

ال data segment (Base) و تجمعلها ال Bx (offset)

* العنوان مش رقم / constant

فرح يبين ال data segment كـ pointer لما نعمل load

لذلك تؤجل عملية تحويل ال address إلى physical إلى

ال (load time)

* العنوان هون رح نسميه Relocatable code ؛ يعني حسب ال location

إلي انعمل فيها loading للبرنامج رح ياخد ال address قيمته من

هذا الشيء وكل مرة بتعمل run رح يتغير الموقع حسب حالة ال memory

③ execution time binding : ما بقدر الـ OS يحول العنوان الا وقت

التنفيذ لانه ممكن العنوان يعتمد على عملية

حسابية لازم تنحسب مسبقاً أو ممكن البرنامج تبعك بنحط

بال memory وبنشال بخاصية ال swapping إلي شفناها

بال mid term schuding (شيل ال process وبوديها على الناتج hard disk)

فالبرنامج بروح وبيجي بين ال hard disk و ال memory

وهذا يؤدي إلى لزوم حساب ال address أكثر من مرة وكل

مرة بنحصل على ناتج مختلف

(MMU)

* بالحالة هاي احنا بحاجة إلى HW معين اسمه memory management Unit

الممسوحة ضوئيا بـ CamScanner

Date Page

* بال compile/load time ما بنحتاج مساعدة من ال HW

عشان يحول العنوان

أما بال execution time رح نحتاج ال (MMU)

شكل سلايد 9 : بفرجيك مراحل تنفيذ البرنامج

① compile time : البرنامج بتحول ل Binary

② load time : تحميل للبرنامج تبعك (inkage editor)

وتحميل للمكتبات بي مستخدمهم (system library)

زي ال i/o streem / mac / manipulation

③ execution time : مرحلة التنفيذ

* physical/ logical بعد عملية التحويل رح تعطي دايماً نفس النتائج

اذا كان compile / load لانهم ما بنشالوا من ال memory

على عكس ال execution time بتغير لانه ال process كل شوي بتروح وترجع

على ال harddisk

الممسوحة ضوئيا بـ CamScanner

Date Page

Memory Management Unit (MMU)

* هاي بتدخل على الخط لما ال process بدها تتحرك كتير بين ال harddisk وال memory لانه رح نضطر نحول العناوين أكثر من مرة

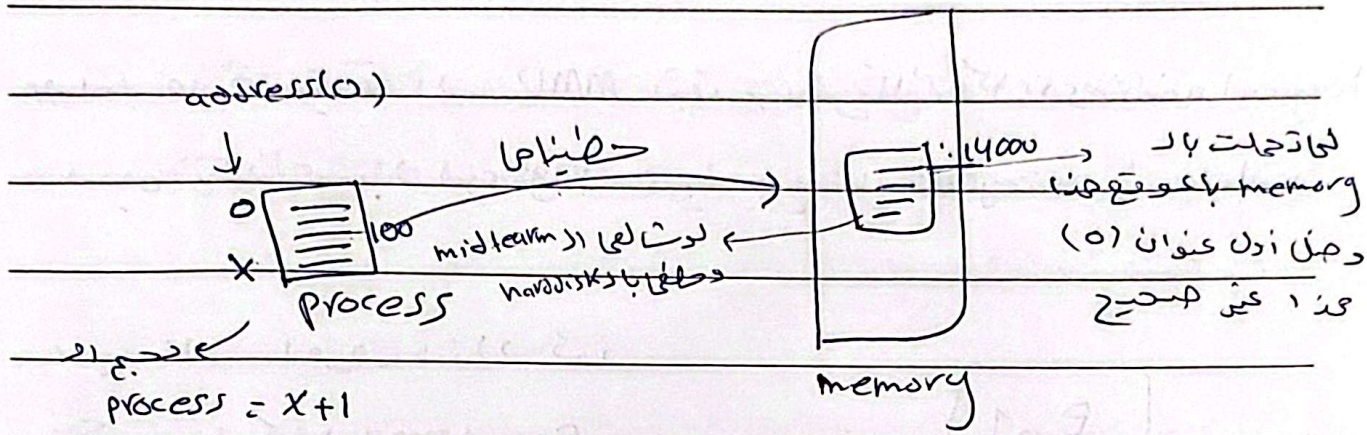
MMU : هي HW وبتشتغل أكثر من مرة بتحاول تعمل binding بين ال virtual وال physical

* التحويل اله أكثر من طريقة واختلاف الطرق راجع لاختلاف طريقة التخزين على ال memory

* فأول ما تشتغل ال process بتتخزن على ال memory ب mechanism معينة وبناء على هاي ال آلية عمليه ال binding تتم داخل MMU

① أبسط طريقة استخدام ال relocation register (بذكرنا بال load time binding)

الممسوحة ضوئيا بـ CamScanner



* كيف بدها تصير عملية ال binding ؟

ال memory مساحتها كبيرة وبرنامج ممكن يرمى بشكل عشوائي
على أي مكان في ال memory بالتالي ال line(0) بال logical
لا يكافئه line(0) بال physical لذلك هون بنخزن اشي اسمه
(relocation regester , starting point , Base)

هو لما يحطلك انه ال starting point = 14000 معناته انت
الصفر قبلك = 14000 و النهاية تكون 14000+X

* أهم اشي نعرف ال starting point وبعديها بنطلع طلوع بالرقم
* يعني موقع 100 بال process رح يكون موجود على الموقع 100
من بعد ال 14000 يعني 14100

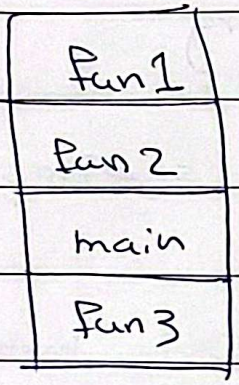
* لعدما شالها ال midtearm رجعت مرة ثانية على ال memory
ولكن بموقع ثاني ويبدأ بال 16000 (أي موقع قديم زيد له 16000 بطلع
الموقع الجديد)

الممسوحة ضوئيا بـ CamScanner

Date Page

* أبسط طريقة الـ MMU بتقدر تعرفلك كل الـ logical address
ايش يكافئة بالـ physical هي جمع الـ relocation regeiter

سلايد 12: افرض عندك كود
بتكون من كذا function
هل من الضروري أمسكهم كلهم وأحطهم
داخل الـ memory وأحجز مكان لـ
3-functions ،



fun1
fun2
main
fun3

ممكن يكون الـ OS مش smart ويحكي نعم
وممكن يكون smart ويحمل الـ function إلي انت مستخدمهم
وبالتالي ممكن نعمل اشي اسمه dynamic louding (تحميل عند الطلب)

اذا اكتشف الـ OS انه الـ main بستدعي fun1 و fun2
وما بستدعي fun3 لما يجي يحمل الكود بالـ memory
اح يحملك الـ main ويصير يعمل dynamic louding بس إلي
بتطلبه بجمله

* حتى لو كانت الـ process بتستخدم أو ما بتستخدم الـ
dynamic louding أساس العمل هو الـ relocation register

الممسوحة ضوئيا بـ CamScanner

Date Page

* فاذا بدك ال old style بتحمل البرنامج كله ابتداءً من ال starting point واسمه static loading لانه حملت الكود كله سواء بحاجة أو مش بحاجة

* الذكاء انك تحمل ال main فقط عند ال starting point وال function بيجوا بعدين عند الطلب وال fun إلي مارح تستخدمها مارح تيجي على ال memory

مقارنة : dynamic loading	static loading
بتحمل فقط ال main عند ال relocation register والباقي بعدين	بتحمل كل الكود عند ال relocation regester
* أفضل من ناحية توفير المساحات Butter memory utilization/saving space	

relocation register لا تختلف وظيفته في الحالتين هو starting point للتخزين

* متى بنستخدم ال dynamic ؟ اذا كودك كبير وفي عديد من ال functions بعضهم غير مستخدم

إلي بميز هاي الطريقة انه عابدها special support من ال OS

* نفس الطريقة ممكن نطبقها على ال library

الممسوحة ضوئيا بـ CamScanner

Date Page

* زي ما حكينا أي برنامج هو مجموعة من ال functions ومجموعة من ال library

* واذا كنت عامل include لـ library وما بتستخدمها وسويت static load رح تخسر مساحة

* خلينا نعمل linking لـ library إلي بستخدم منها أجزاء وأضيفها

Dynamic linking:

دمج ال library مع الكود

اذا static : رح تضيف كل المكتبات سواء مستخدمها أو لا

بالتالي حجم الكود بصير كبير وخسرت مساحة كبيرة من ال memory

بالتالي ال os بكون غير راضي عن هذا الشيء

* اضافة لمهمة ال os بال protection عنده مهمة اختصار مساحات

لانه ال memory عليها high compatition فبحاول قد ما يقدر ما

يحمل فيها الا كل ما هو ضروري عن طريق ال dynamic linking

(إلي ما بستخدمه ما بجيبه ، إلي خلصت منه طلعه من ال memory)

* ال library إلي بلزموك جيبلهم pointer اسمه stub

وإلي ما بتستخدمهم ما تتطلع عليهم أصلاً وكأنهم غير مكتوبين

وال Dynamic linking تحدث في execution time

الممسوحة ضوئيا بـ CamScanner

الهدف من ال dynamic loading وال dynamic linking توفير المساحات

↓ الوقت يصبح أسود

↓ ماعندها مشكلة

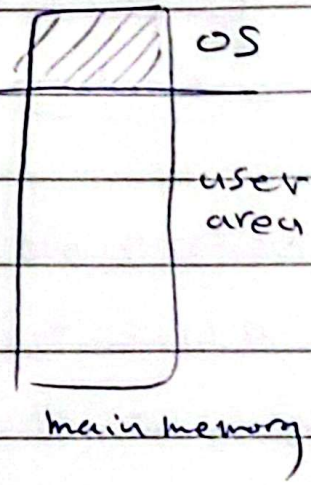
لانه لما تحتاج اشي بدك تروح تجيبه فالبرنامج واقف وال Kernel رح يشتغل شان يجيبلك ياه

ببطئ أو سرعة لانه انت ما بتجيب ال library كلها انت بتحط مؤشر على اي بتلزمك فقط

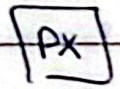
Contiguous Allocation:

رح نحكي الآن كيف نخزن ونتطرق إلى ٣ معايير:

① Contiguous allocation: يعني مسكت process حجمها 1K لازم أحطها بال memory بمساحة حجمها 1K وبشكل متتالي (ورى بعضها) زي ال array



• خلي ببالك انه ال memory ساحتها محدودة جداً لذلك بدك تكون عادل و efficient



• اذا هاي بدها مساحة 1K ال Contiguous بعطيها 1K بالزبط ومتتالين وهذا ال 1K إلي بدي أخزن فيه اسمه partition

الممسوحة ضوئيا بـ CamScanner

Date Page

وهون بصير عن 2-types of partions :

① Paritition الهم نفس الحجم

② Partitions الهم Variable size (متغير)

* ال key idea إلي رح نشتغل عليها :

① Resident OS (يعني ال Kernel code داخل ال main memory)

بالعادة بكون بال low area في ال memory (OS

② ال user process بيجوا بال higher area

تحت ال low part اعطيها ال OS

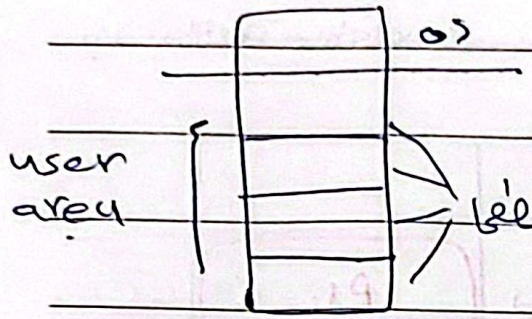
③ each process contained in single contiguous section of memory.

section واحد حجمه بحجم ال process ومتتالي

الممسوحة ضوئيا بـ CamScanner

Date Page

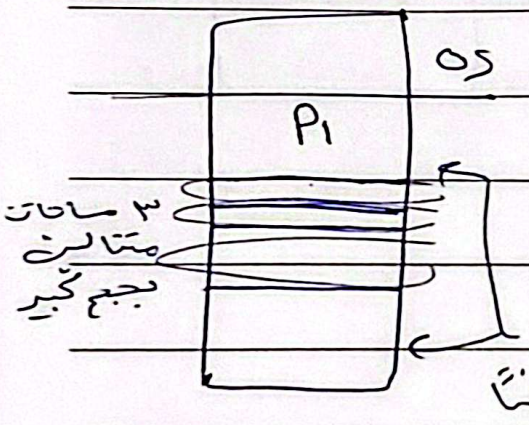
① Partetion اللمم نفس الحجم



بالتالي اذا احبت $\boxed{P_1}$ ما بفرق وين تحطها كله نفس الاشي

* هل من الممكن حجم ال Partition ما يكفي ال process ؟
هون لازم يكفي ، لانه حجم ال Partition بكون بحجم أكبر process موجودة

② Partitions اللمم أحجام متغيرة



بالتالي اذا اجبت process مش كل الخيارات ، ح تؤدي لنفس النتيجة $\boxed{P_1}$ فرضاً بتوسع بأماكن محددة فرضاً

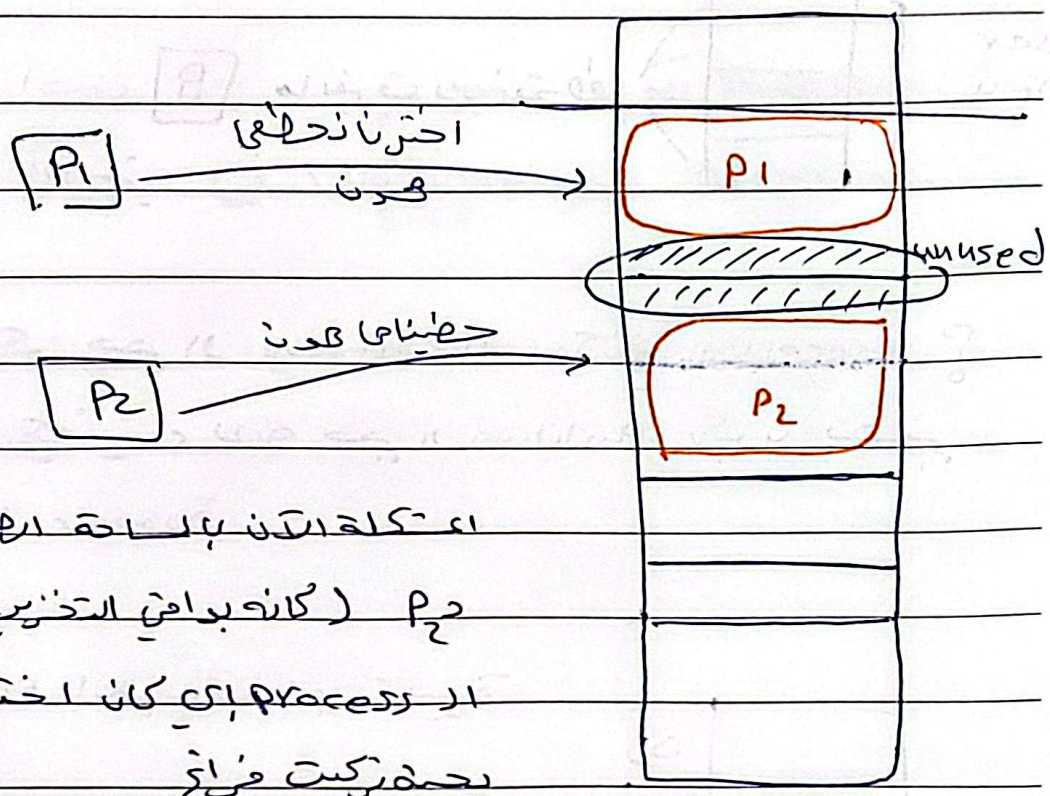
Fixed Partition	Variable partion
بما انه مساحته بمساحة أكبر process فاذا اجا Processes صغار ما ح تحتاج كل المساحة بالتالي تضيع مساحات وهذا اسمه internal Fregmentation (مساحة غير مستغلة)	* P_1 لما تتخزن في أي مكان بيوسعها أي اشي زيادة ما بظل محجوز باسم P_1 بل بتصير Free location خلي الباقي يستفيد منها ، هاي مش دائماً منيحة ، منيحة من ناحية عدم حدوث internal Fregmentation
Bad memory utilization	

داخل ال Block إي

الممسوحة ضوئيا بـ CamScanner

Date Page

بالـ Variable partition ممكن يصير في wast بالمساحات من نوع آخر



المشكلة الآن بالمساحة الصغيرة بين P_1 و P_2 (كانه بواقي التخزين من P_1 و P_2)

الـ process إي كان اختيارها سيئ بحيث تركت فراغ

* بالتالي هاي الدائرة الزرقاء صارت wasted لانه حجمها صغير بالتالي يعتبر fregmantation (مساحة مش محجوزة باسم حدا ولكنها غير مستغلة لانها صغيرة)

* هون كان في سوء اختيار مكان الـ process وهاي الظاهرة اسمها external fregmantation الـ حل:

① حرك الـ P_2 شوي لتحت حتى يكبر مكان الفاضي

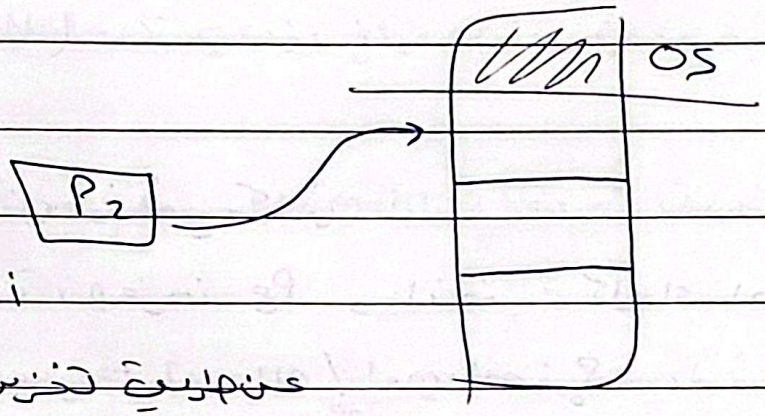
② حرك الـ P_2 لفوق بحيث يروح الفراغ

* ملاحظة الـ internel fregmantation ما الـ حل *

الممسوحة ضوئيا بـ CamScanner

Date Page

سلايد 15 : برجع بحكي عن ال relocation register



لما أحطهما بمكان

كيف بدي أرجع

أعرف وين حطيتهما؟

عن طريق تخزين قيمة ال

Base أو ال relocation

في مكان معين ممكن يساعدنا limit reg

في حالة ال variable partition عشان

أعرف وين بتنتهي ، أما إذا كان

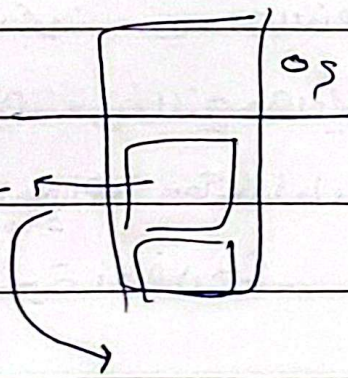
static partition فشا دامي تعرف وين بتنتهي

بس بدنا ال relocation register يعرف وين

مبلشة

* لما نكون نستخدم ال contiguous allocation بكون مرن أكثر

لانه حجم ال kernel (OS) ممكن يكبر في حال صار عندك extra i/o devices



عشان تكبر هذا

شيل هاي لمكان ثاني ونزل خط ال OS

وغيرتلها ال relocation register (نقطة البداية)

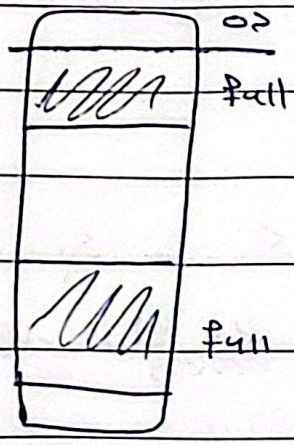
الممسوحة ضوئيا بـ CamScanner

Date Page

سلايد 17 : بفرجينا variable partition وحجم الأماكن
القابلة للتخزين هو الجزء المظلل تتغير مع مرور الوقت
وبنسميها (Hole)

① P8 خلصت شغل وفضي مكانها
② P9 احتجت وكانت أصغر من P8 وما أخذت كل المساحة
هل في enternal / external fregmantation ؟
OR
internal لا يوجد
external ممكن يصير لو صار في فراغ صغير بين الـ processes

سلايد 18 : بشر لـ variable partition يعني لما يكون عندك خيارات
غير متساوية الحجم وتوصلك process 5 حابة تتحمل بالـ memory
وتبدأ تنفيذ وين بدنا نحملها بأي
جزء (علماً انه لما تتحمل بجزء
كبير الباقي رح يترك كمساحة تخزين
محتملة لـ process ثانية اذا بكفي)



عنا ٣ معايير :-
① first-fit : وانت بتوف المساحات الفاضية
أول مساحة تناسب P5 خزنها فيه هاي الطريقة
هدفها الـ speed مش الـ optimal utilization story ممكن أول مساحة
تلاقيها تكون أكبر بكتير بالتالي ليست أفضل خيار

الممسوحة ضوئيا بـ CamScanner

② Best-fit : بدك تلف على كل الأماكن الفاضية وتشوف أحجامهم
واختار المكان الفاضي إلي حجمه قريب كتير على حجم الـ process
بحيث ما يصير عندي فراغ يؤدي إلى extarnal fregment

هون هدفها الـ memorgutilization من أسرعة

③ worst fit : هاي ما في OS بعتمدها ، بدور كل الأماكن الفاضية
وبعطي الـ process أكبر حجم موجود حتى لو ما بدها ياه
(خيار غبي)

* Paging : طريقة يعتمدها الـ OS عشان يقوم بتخزين
الـ processes داخل الـ memory
(Non contiguous allocation)
تخزين غير متتالي (تجزءة الـ process إلى اجزاء وكل
جزء بنحط بمكان)
- هل الاقسام متساوية الحجم ؟
- كيف بدنا نعرف نجيب الـ process وهي كل جزء بمكان ؟

Paging صح انه بغلب ولكنه جاء لحل مشكلة الـ fregmentation

الممسوحة ضوئيا بـ CamScanner

Date Page

* الـ Paging يلغي وجود الـ external fragmentation *

فكرة الـ paging :

① تقسيم مساحات التخزين بالـ memory إلى اجزاء صغيرة

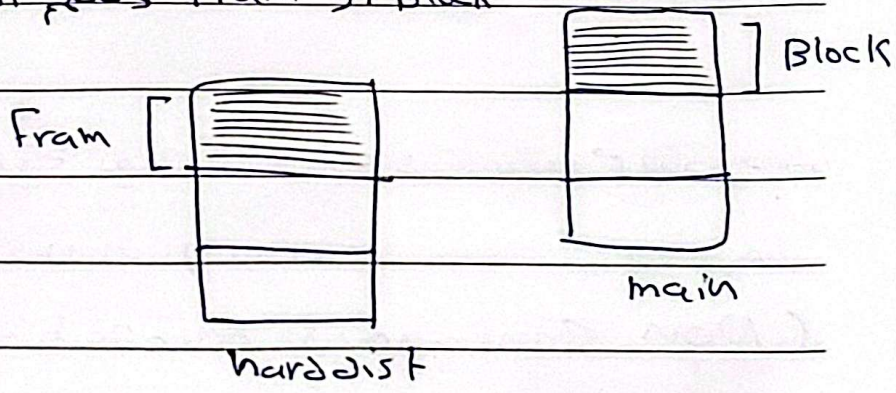
الـ process بداية بتكون موجودة بالـ harddisk وبعدين

تنتقل للـ main memory بالتالي بدنا نقسم الـ harddisk والـ main

إلى اجزاء صغيرة متساوية

فرضاً بدي أقسم الـ main إلى blocks كل واحد فيه 8 rows

Block أو Fram وكلهم الهم نفس الـ size



* لازم الحجم يكون power of 2 يعني 2 , 4 , 8 ...

وهو معطى بين ($512 = 2^9$ — $16M = 2^{24}$)

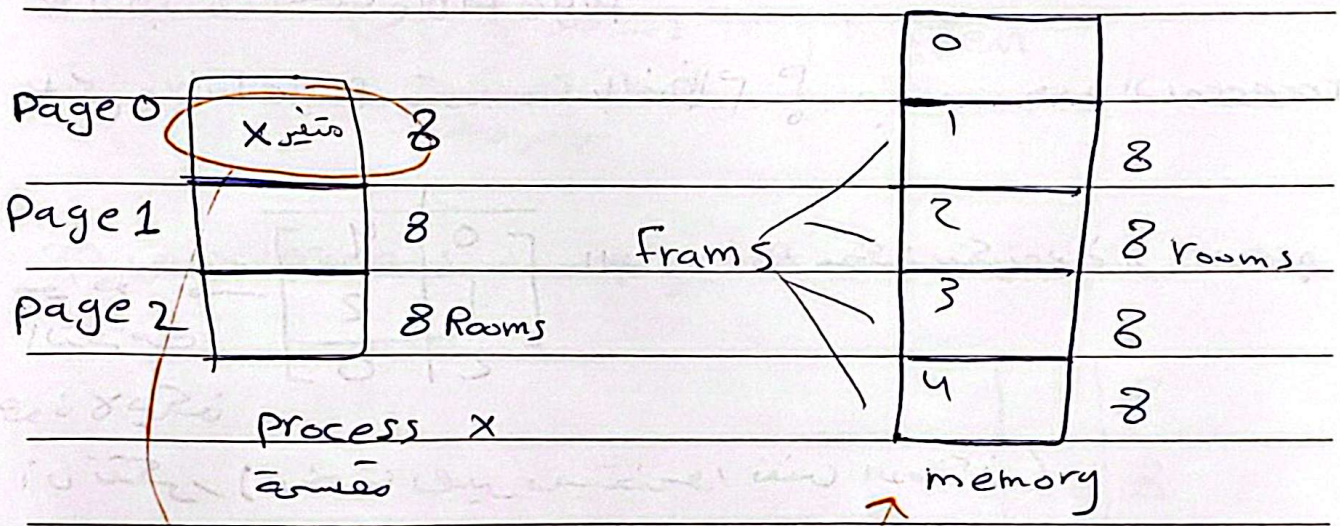
وعند التخزين رح نقسم الـ process بنفس حجم تقسيمة الـ

main/harddisk

- كله لازم يكون نفس الحجم

الممسوحة ضوئيا بـ CamScanner

بالتالي ال Paging : انك تمسك ال process اي حاجة
تتخزن بال memory تقسمها اي rooms بحجم معين
من (512 - 16 M) ويتروح بتقسم ال memory نفس الحجم



لما بدك تخزنه يتروح
ويتدرله على Fram فاضي وهي عملية عشوائية

* الأماكن اي كانت مرتبة بال process تصبح مبعثرة في memory
* لانه كل ال Frams جنب بعض عشان هيك تلغي ال external frag-

* لو بدي أعمل عمليات على متغير X كيف ال computer بده يرجعلك
ياه ودين تخزن ؟
أي عملية تخزين عشوائية لابد يكون مرافق الها table
وفيه كل مكان دين راح بال memory

0	4
1	2
2	0

Page 0 راح موقع Fram 4
Page 1 راحت Fram 2
Page 2 راحت Fram 0

الممسوحة ضوئيا بـ CamScanner

Date Page

بدونه لـ Table) لا يمكن تعرف وين الـ page راحت

على أي fram بالتالي الكود ما بعمل run ولا بعمل fetch

للـ instraction

* الجدول اسمه page table

* كم page table بكون عنا بالنظام؟ بنفس عدد الـ processes

العمود هذا ممكن يتكرر بين الـ process

0	4
1	2
2	0

→ مواقع التخزين

هون لا يمكن

أن تتكرر (لانه ما بصير يستخدموا نفس الاماكن)

logical memory	physical ~~logical~~ address space /
logical address space	physical memory
المقصود فيها الـ	المقصود فيها:
Process space يعني ($page_0$ ... $page_2$)	RAM / main memory إي فيها الـ
	frams

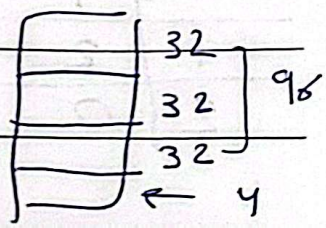
* كيف بدي اضمن انه البرنامج إي يكتبه حجمه من مضاعفات الـ pages؟

يعني لو كل page حجمها 32 = 2^5 وكودك 100 سطر

رح يضل ٤ شو اعمل فيهم؟ رح نحجز لهما 32

وهذا يؤدي إلى internal fregmantation

ولكن حصراً في آخر page



الممسوحة ضوئيا بـ CamScanner

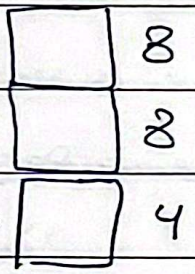
* ان حصل نقص نتيجة انه حجم ال process أقل من مضاعفات ال pages بكون هذا في ال page الاخيرة و هذا يؤدي بال internal fragmentation

↓

may happen (مش دايماً بصير)

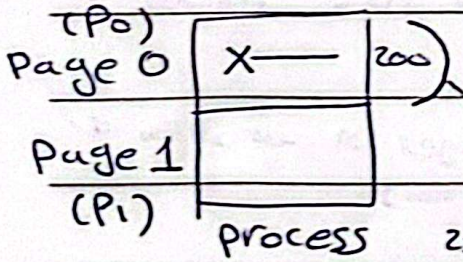
واذا صار بكون بآخر وحدة

* لو انت فارض حجم ال page = 8 واجتك process حجمي 20



من أصل 8 فحصل internal Freg.. حجمة ٤ مواقع

* Address Translation scheme



يعني هذا ال Block ممكن يكون حجمة 256 أو 512 أو أكبر

عشان ال memory تقدر ترجع قيمة X بدك تعمل التالي :

P0 : 200

لازم ال os يكون قادر يترجم متغير X يقع على سطر معين من Page 0

حاي كموقع على ال memory

لازم أرجع على ال Page table أشوف P0 عند أي Fram فرضاً كانت عند Fram 12 بالتالي بصير بدك F12 : 200

* اسطر 200 من F12 الذي بخزن P0 كاملاً

الممسوحة ضوئيا بـ CamScanner

Date Page

① Page number : يعني P_0 و P_1 و P_2

$P_0 \to 0$ ، $P_1 \to 1$ ، $P_2 \to 2$

بس هيك ما بكون حدث اي offset/ line

② Page offset (line) : اسطر اي جوة ال page

Page number = 0 ، page offset = 100

يعني بدي سطر 100 من ال page 0

* لو في process حجمها 2^m بدنا نقسمها ل pages حجمها 2^n

- حجم ال page رح يكون معتمد بالنظام كامل

- لو حكيتلك عندك page حجمها 512 ايش بتستنتج ؟

انه جميع ال pages في ال logical address space لجميع ال process وكل frams تبعون ال memory رح يكون حجمهم 512

ما هو عدد ال pages (frams) المطلوب ؟

$$2^{m-2} = \frac{2^m}{2^n}$$

Page number = $(m-n)$ bits

Page offset = n (قوة ال page size)

اذا ال page/frame حجم 512 بكون ال page offset = 9

$512 \to 2^9$

الممسوحة ضوئيا بـ CamScanner

Date Page

سلايد 21 : بفرجيك Page address resolution HW

يعني لما برنامجك يحكي انه بده اسطر كذا من page كذا

شو بصير وراء الكواليس داخل الـ OS حتى يقدر يعرفلك

الـ address المطلوب وين موجود بالربط بالـ memory

سلايد 24 : overview about paging techiques

① تخيل عندك process حجم الـ page الوحدة = 2048B = 2^{11}

② لو اجاك برنامج حجمه = 72.766B كم page رح تحتاج ؟

كم page بدك = $\frac{72.766}{2^{11}}$ = 35 وشوي (1086B)

بجبرهم

بالآخر رح احتاج (36) ولكن فيها internal Fregmantation

كيف نحسب الـ internal frag ؟

حجم الـ page − المتبقي

2048 − 1086

= 962B

* الـ Paging اذا كان اله سلبيات ؟

① internal Fragmentation

② قاعد بعمل overhead بسبب تخزين الـ page table لكل process

الممسوحة ضوئيا بـ CamScanner

Date Page

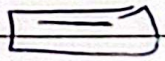
* أكبر كمية لـ internal fragmentation ممكن تصير (worst) :

- لو حاي الـ page = 2,048B اتاخد منها Byte واحد
والباقي ضلة فاضي . 1 Frame (page) — 1 byte

* average fragmentation = 1/2 Frame size
فاضي

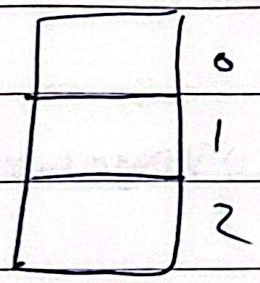
* بما انه الـ page الاخيرة ممكن ماتتعبي ليش ما نصغر حجم
الـ pages كلهم وقتها نسبة الـ wast بتكون قليلة

* لو بدلا تعمل page كبيرة وتاخد منها شوي الباقي wast (فاضي)

اما لو page صغيرة  حتى لو اخدت منها شوي الباقي بكون قليل

* هو فعلاً أحد الحلول ولكنه غير مستخدم لانه له تبعات أكبر
من انه يصير internal fragmentation

قسمنا الـ pages كبار بالتالي احتجت 3 pages
معناته الـ page table رح يكون فيه ٣ فقط



Process

* ممكن يصير في internal frag عالي بالـ page الاخيرة

0	Fx
1	Fy
2	Fz

الممسوحة ضوئيا بـ CamScanner

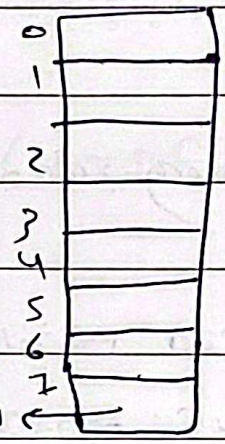
Date Page

لو تسمنا ال process السابقة لاجزاء أصغر:

P_0	F_x
P_1	F_y
P_2	F_z
P_3	⋮
P_4	
P_5	
P_6	
P_7	

حجم ال Page table

اح يصير كبير



ال internal freq اح يصير هون ومارح يكون عالي لانه

حجم ال fram صغير

* لا يوجد حل مثالي كل واحد اله مساوءه

* الحل الذي يستخدمة ال OS مدموج بين الطريقتين

سلايد 25:

Free-fram List : data structer بخزنها ال OS عشان يعرف ال Frames الفاضيين

* بالواقع ال OS بخزن معلومات عن ال Frames الفاضيين والمليانين

ومين صاحب المليانين

الممسوحة ضوئيا بـ CamScanner

Date Page

Implementation of page Table

Fram14 على page 0 ←

0	14
1	13
2	18
3	20

* هذا الجدول لازم أخزنه بالتالي لما بدي أحطه بـ memory أو cache أو register

* الـ process size مستحيل يتولد منه 4-pages فقط لأنه الـ process بكونه كبار وقد يتولد منهم عشرات بل مئات الـ pages بالتالي حجمه كبير فمش رح نقدر نخزن هذا الجدول بـ register ولا بـ cache

page table عددهم كبار وحجمهم كبير لذلك الخيار الأمثل هو تخزينهم في الـ main memory

الآن لو مسكنا هاي الـ page tables وخزناه بالـ memory رح نرجع لمشكلة ثانية (لما بدك تشوف الترجمة وين هي الـ page table)

* بالتالي عملية قراءة محتوى معين تتطلب زيارة الـ page table بالـ memory وزيارة الـ memory بعد اكتشاف مكان الـ page tables ومعرفة على أي Fram

* لذلك عنا 2-pointers عشان أشوف الـ page table

① page table - Base register (PTBR)

② page table - length register (PTLR)

الممسوحة ضوئيا بـ CamScanner

Date Page

* ال (PTBR) بدلك على بداية ال page table تبع أي process
(هو إي بلا فيها بال memory)

* (PTLR) تقلك انه هاي ال process كم فيها pages

* ال OS بشتغل انه كل ال processes الها نفس العدد من ال pages
ولكن ال process الصغيرة تحتاج (PTLR) صغير فاذا
كان ال PTLR صغير هذا يدل على انه ال process آخذه 2 pages
مثلاً 2

واذا كان PTLR=200 يدل انه هاي ال process أخذه 200 pages / Frams
200-

PTBR : بخبرك الجدول وين ببلش وال PTLR بخبرك
كم حجمه وبالتالي بتقدر تعرف وين ينتهي

* زيارة أي offset من ال page table يترتب عليه زيارتين
لل memory .

لانه لما اقلك بدي offset من ال page الأولى أول اشي بدك
تشوف ال page table إي هو أصلاً بال memory وتقلك راح على أي Fram
وهاي الزيارة الأولى ، بعد ما تعرف أي Frame راح تروح عليه
وهي الزيارة الثانية

الممسوحة ضوئيا بـ CamScanner

Date Page

* هذا الاشي مش كثير منطقي لانه كل مرة بدك تعرف اشي بدك تزور ال main mem مرتين واحنا عارفين انه ال main بطيئة لذلك أيضاً يجعل وقت التنفيذ أطول

* لحل مشكلة زيارة ال memory مرتين اعمل مكان بال computer خزن عليه ال page table يعني فرضاً حطه بال cache بالتالي أعرف وين ال Fram وبعدين زور ال main

بالتالي صار زيارة الموقع بال memory تكلفته :

زيارة ال cache + زيارة ال memory

سلايد 27 : شكل ال page table ما تغير ولكن إي تغير انه رح استخدم Associative memory أو cache memory أو transilation lookaside buffer (TLB) عشان أخزن ال Page table

* عدد ال Page إي بتستخدمهم ال Process كثير كبير مثلاً (50,000 Page) وهذا يعني انه الجدول رح يكون عملاق لذلك تخيل بدك تحط جداول كل ال Process اعوضوع بظل واقعي لانه ال cache أو TLB حجمها صغير

* ايش رح نعمل ...

الممسوحة ضوئيا بـ CamScanner

Date Page

* رح نعمل الجدول العملاق ولكن إلي بستخدمهم بكثرة خليهم بال cache أو TLB عشان السرعة

- ال TLB لا يحتوي على كل ال pages ولكن بعضهم الأكثر استخداماً

- اما ال pages إلي استخدامهم قليل خليهم بال memory

* اذا لقيت ال page بال TLB رح نقول hit
واذا لا رح نقول miss وندور بال main
سلايد 28

- التدوير بال TLB بكون parallel يعني كل المواقع مع بعض

* if the translation is done in TLB :
the time is low , fast
* if you did not find in TLB go to main
slow so the time is high

لذلك رح نحكي عن ال miss/ hit ratio
↓
كم نسبة تلاقي طلبك
بال (TLB)
لو حكينا 20% معناه لما
يطلب address وتيجي تترجمة في 20% من المرات ال TLB بتفيدك
80% لا , رح تروح main

الممسوحة ضوئيا بـ CamScanner

Date Page

بصير ال Access time (عملية زيارة ال memory)

hit ratio × hit cost + miss ratio + miss cost | hit ratio = α

ε = ابسيلون

(وقت ال TLB)

$$EAT = (1+\varepsilon)\alpha + (2+\varepsilon)(1-\alpha)$$

وقت زيارة ال memory | TLB time | hit | TLB time + 1 + 1 | miss

1: accesse page table in memory

1: accesse memory itself

Memory protection :

- كيف ال OS بضمن انه ال process ما تتعدى على جارتها

يعني process فيها 6 pages والمبرمج طلع عنوان لل page 10

هذا غير صحيح

عشان نضمن انه ما في violation من ناحية ال accesse

① ال Pagetable بكون فيه valid-invalid bit

valid لل pages اسموح لل process تستخدمها لانه من ضمن عناوينها

زي سلايد 31

الممسوحة ضوئيا بـ CamScanner

Date Page

- بالعادة ال page tables بكون حجمهم كبير حتى ولو مش كل

ال pages translation موجودين فيها

* لما تشوف مواصفات جهازك بعرف انه 32 bit أو 64 bit

هذا الرقم يدل على حجم أي process بالحد الأعلى

يعني P_1 حدها الاعلى هو 2^{32} أو 2^{64} حسب الجهاز

لما تشتغل هاي ال process بدنا نقسمها لـ pages فرضاً

حجم كل page = $2^9 = 512$

بالتالي $\frac{2^{32}}{2^9} = 2^{23}$ بعطيك عدد ال pages إي رح تحتاجهم

رقم كتير كبير

بالتالي من غير اعتطاقي استخدام ال TLB (لانه للجداول الصغار)

* ولانه حجم الجداول جداً ضخم لازم ال OS يتدخل ويقلل

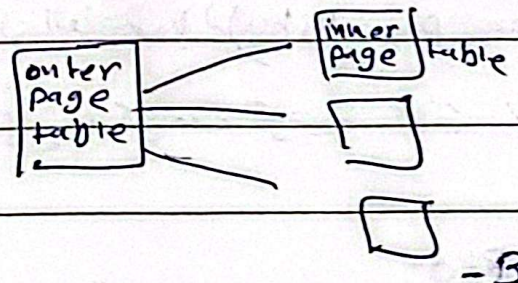
مساحة التخزين لانه هو بالنهاية Secondary data structure

بتساعدك تلاقي ال Frames مش اشي اساسي.

عن ٣ طرق لحل هاي مشكلة :

① hierarchical page tables : اذا كانت ال page table التابع لـ Px

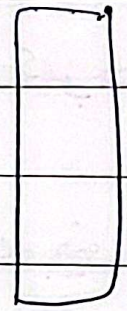
كتير كبيرة ، اقسمها levels



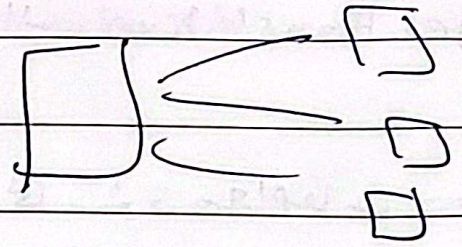
الممسوحة ضوئيا بـ CamScanner

Date Page

الاستفادة : الجدول بالشكل هذا بشرط على ال 05 يلاقيله مكان ورى بعض ويضع مساحة



أما التقسيم



ممكن تروح تخزن كل وحدة بمكان حسب المتوافر بدون شروط زي سلايد 34

ال Pagenum رح يتقسم بعدد ال levels الموجودين

مثلاً لو كان Pagenumber = 22 لو بدك تخزنهم بالطريقة التقليدية بدك 22bit حتى تخزنه

بالطريقة المعدلة رح تقسمهم لـ inner / outer

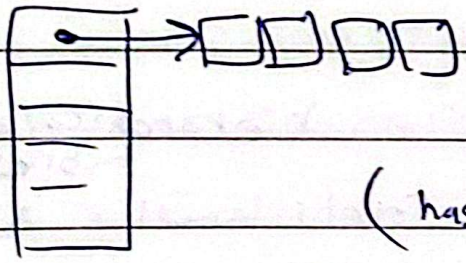
(التقسيم غير مطالبين فيه دفعة الصيفي)

② Hashed Page Tables :

لما يكون عندك page table هائل الحجم ممكن تعمل smaller page table

بحيث المدخل فيه بشرايي أكثر من entry

من ال page table



(المدخل الواحد بال hashed page table)

بعطيني أكثر من ترجمة زي سلايد 38

الممسوحة ضوئيا بـ CamScanner

Date Page

③ inverted page table :

تبنى بشكل معكوس

لو بدي أبني كذا page table

بالطريقة القديمة رح يصير المساحة

المحدودة كبيرة لذلك رح نبنيه

بالعكس :

F0
F1
F2
F3

① شوف Frame 0 حمل في أي process

حاطة اشي داخلة اذا لا انتقل على اي بعدة

② Frame1 حمل في أي process مخزنة

داخلة أي اشي اذا آه بصير التالي

Frame1 داخل page7 حاطة process x

Px	P7	F1
Py	P20	F3

③ Frame2 حمل حدا مستخدمة فرضاً لا

على اي بعدة

④ Frame3 حمل حدا مخزن فيه فرضاً آه

process y مخزنة page20 داخل Frame3

وهكذا

أول فارق : رح يكون بالنظام Single inverted page table

(تذكر بالبداية لما قلنا عدد ال page table = عدد ال process)

أما هون رح يكون جدول واحد فقط مبني بشكل عكسي ولازم

يتم ارفات ال process بالجدول

الممسوحة ضوئيا بـ CamScanner

Date Page

* يزداد الوقت اي مطلوب للبحث

P2	P30

P2 طلبت page30 رح

تنجبر تبحث بكل ال

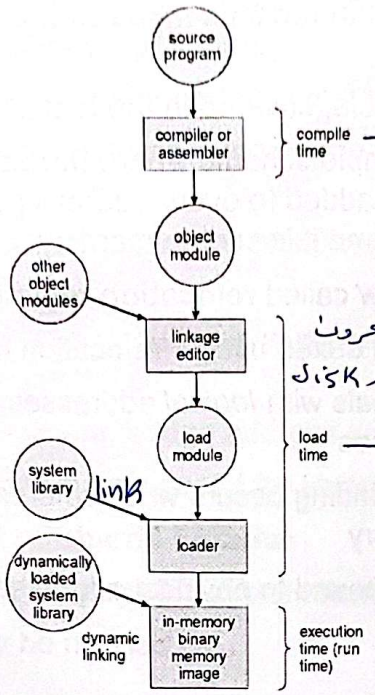
inverted page table حتى توافق أو ترفض

ولكنها بكل الأحوال بتضل أصغر من الجداول العادية

ولكن الجداول العادية أسرع لانه كل process الها جدول لحال

الممسوحة ضوئيا بـ CamScanner

Multistep Processing of a User Program



تحويل العنوان بالمرحلة هاي اذا
كان absolute او constant
وهي مرحلة مبكرة من التحويل

اذا العنوان غير محدد ولكن ال process معروف
انها مارح تروح وتيجي بين ال memory وال disk
رح نعمل التحويل بال load ، متى
ما يتحمل بنعرفوا العناوين

اذا ال process بتروح وتيجي رح
ننتظر وقت التنفيذ لانه كل مرة
بتروح فيها ال process وبترجع بصير في
new binding لذلك يؤجل التحويل لوقت
التنفيذ

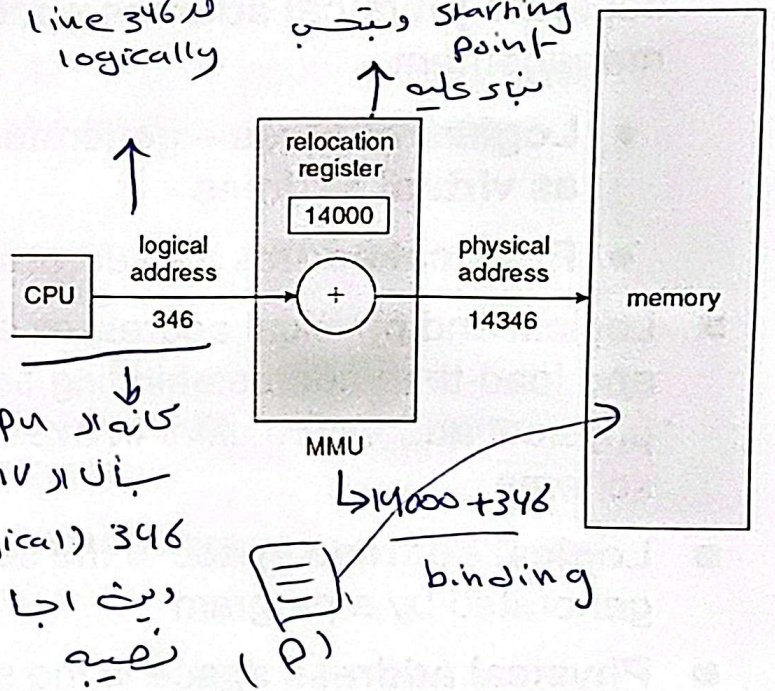
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

الممسوحة ضوئيا بـ CamScanner

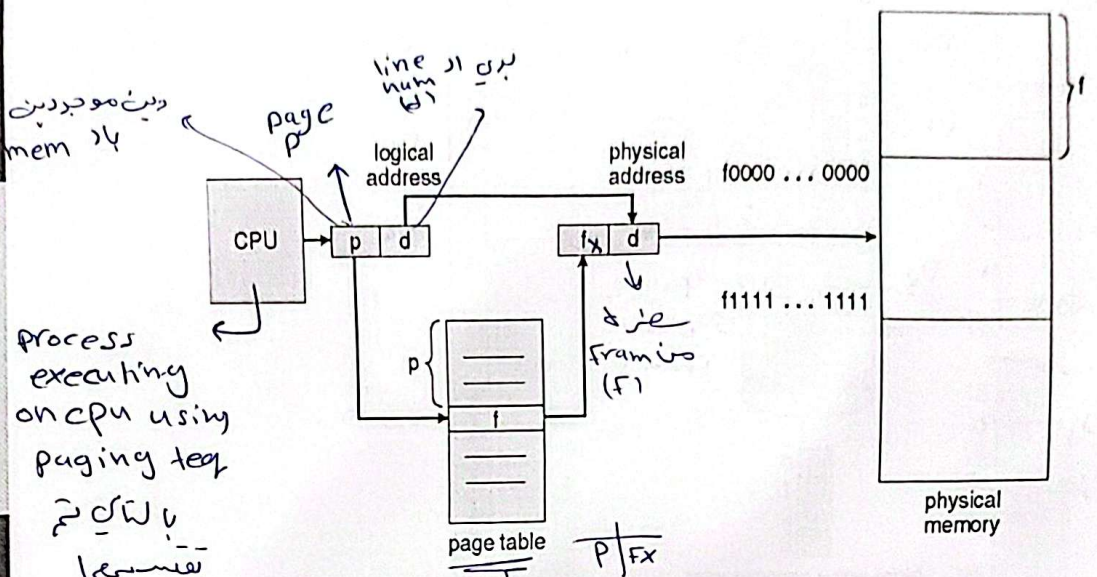
Dynamic relocation using a relocation register

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading

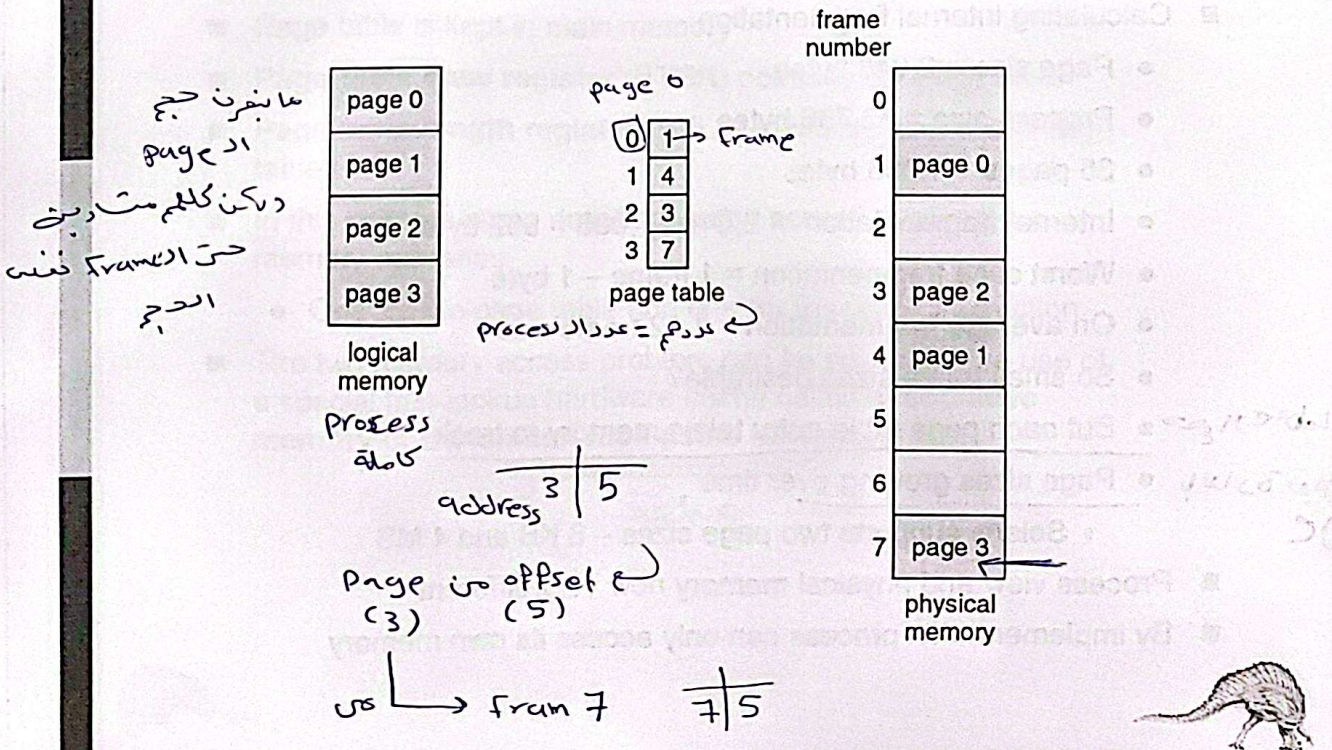


الممسوحة ضوئيا بـ CamScanner

Paging Hardware

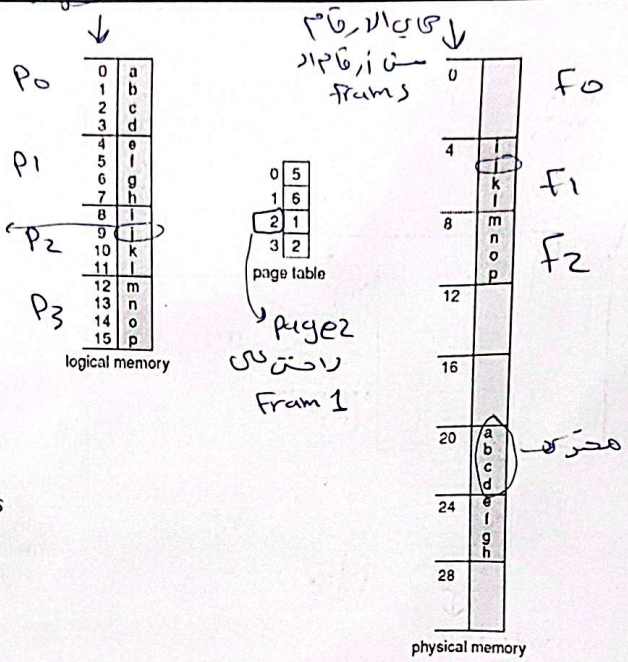


Paging Model of Logical and Physical Memory



الممسوحة ضوئيا بـ CamScanner

Paging Example



n=2 and m=4 32-byte memory and 4-byte pages

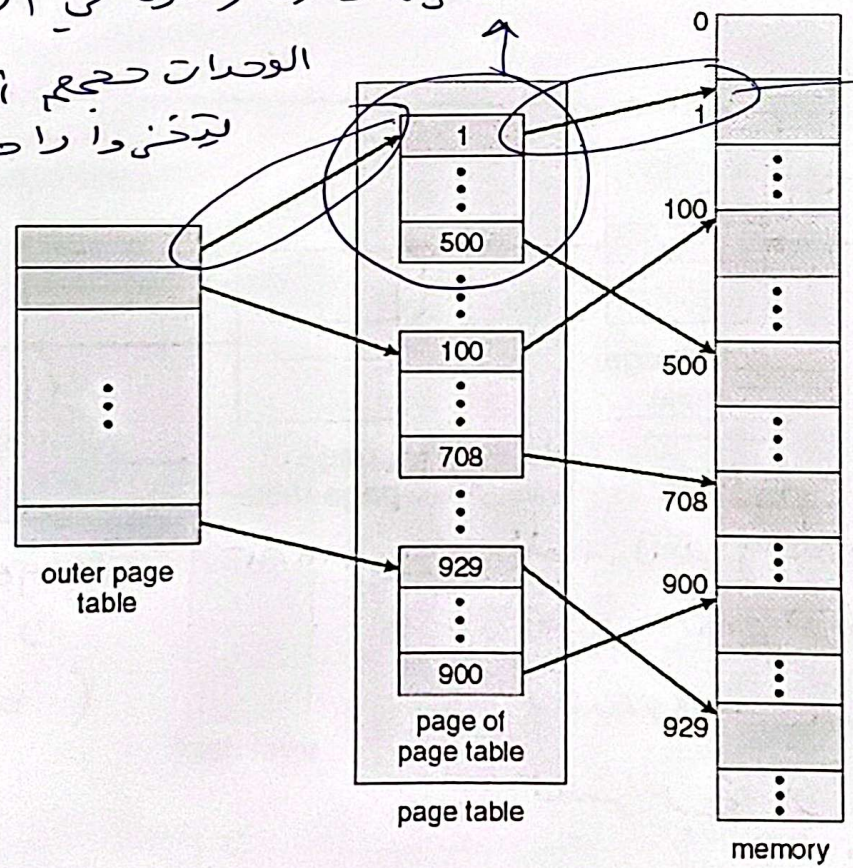
Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
  - So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Solaris supports two page sizes – 8 KB and 4 MB
- Process view and physical memory now very different
- By implementation process can only access its own memory

الممسوحة ضوئيا بـ CamScanner

Two-Level Page-Table Scheme

تخزن كوحدة واحدة في ال memory
الوحدات حجمهم أصغر فأسهل
ليتخزنوا داخل ال memory



Forward mapped

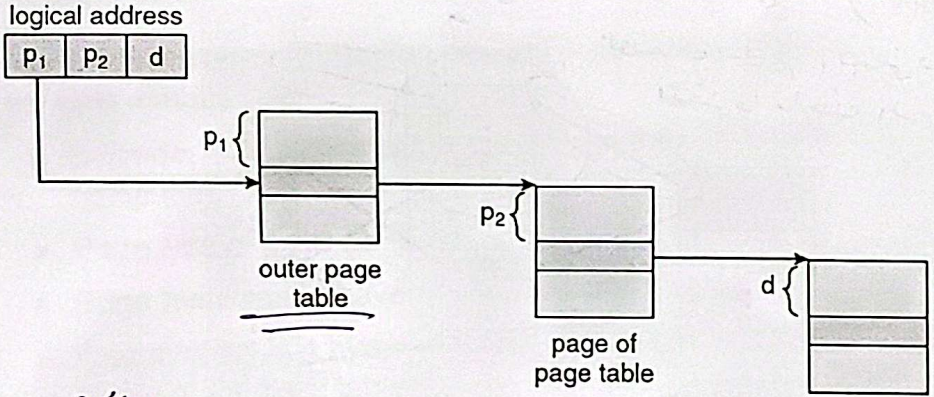
الممسوحة ضوئيا بـ CamScanner

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:

page number		page offset
p_1	p_2	d
12	10	10

- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
- Known as **forward-mapped page table**

Address-Translation Scheme



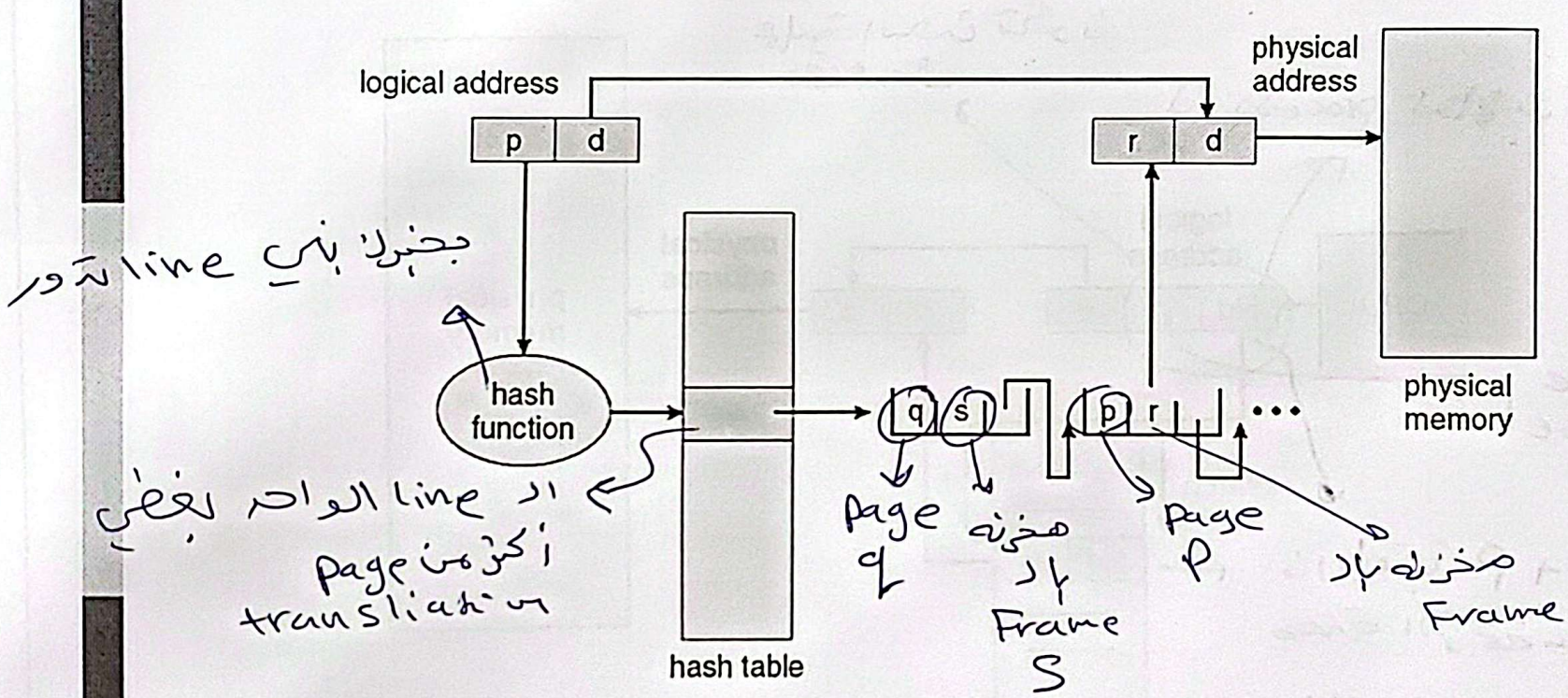
الوصول لل memory بأكثر من مرحلة لذلك
توقع الاجراء يكون أبطأ
وهذا على حساب حل الحجم الهائل
لتخزين ال page table

inner

اقسم ال page
بال frame)
(memory

الممسوحة ضوئيا بـ CamScanner

Hashed Page Table

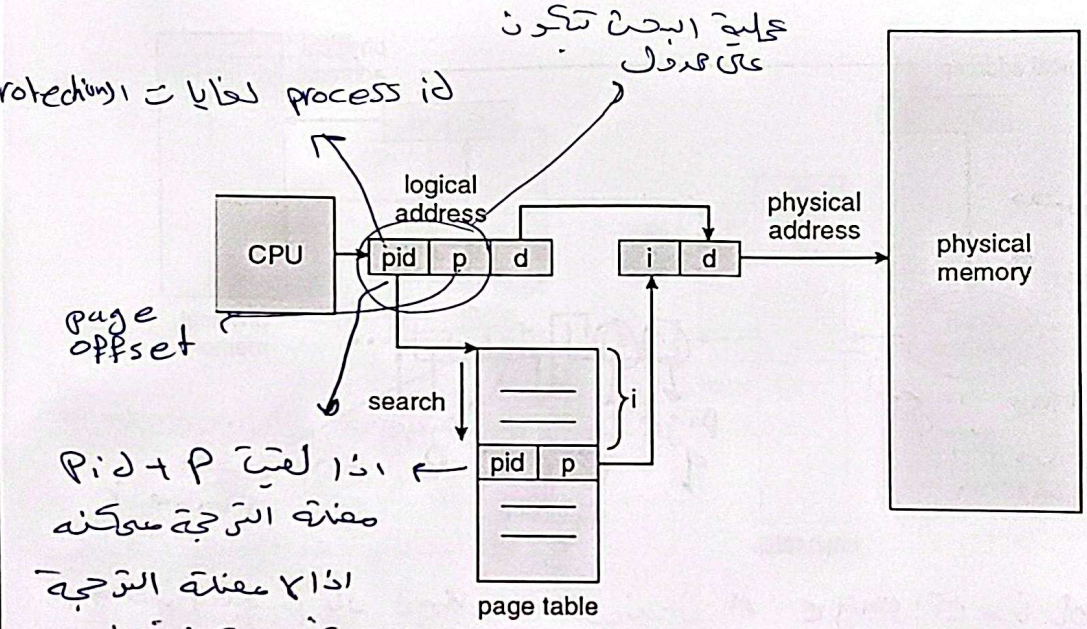


* لازم نبحث بال linked list حتى نلاقي ال page إي بدنا ياه

الممسوحة ضوئيا بـ CamScanner

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

Inverted Page Table Architecture



الممسوحة ضوئيا بـ CamScanner