



Operating Systems



Call Us :

WhatsApp: <https://wa.me/qr/7KNUQ7ZI3KO2N1>

LinkedIn: https://www.linkedin.com/in/ahmed-hany-899a9a321?utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=android_app

Facebook: <https://www.facebook.com/profile.php?id=100041435445946&mibextid=JRoKGi>



Topics

1. Introduction
2. Operating Systems Structures
3. Process
4. Threads
5. Process Synchronization
6. Semaphores
7. CPU Scheduling
8. Deadlock
9. Memory Management
10. Virtual memory

Chapter1 (Introduction)



1. Computer System Components
2. Computer System Organization
3. Computer System Architecture
4. Operating System Operations
5. Computing Environment

Computer System Components



1. Hardware
2. Applications
3. Operating System
4. User

تخيل معايا كدا انت عاوز تروح الجامعة وحيث وقفت قدام عربتك وقللها ودينى الجامعة هل دا يعقل او يتنفس؟
طبعاً تحتاج انك تسوق وتحتاج تبقي عارف رايح فين تعالى بقا نفهم
العربية هي الـ H.W و وجهتك هي الـ Application وانت الـ user

طب دلوقتي لو نفس العربية عاوز تروح بيهما مكان ثاني فبالتالي هتعوز Application تاني
يعني عشان استخدم الـ H.W في كل مرة للاداء عمليات مختلفة هتضطر تكون عاملو Application جديد
وهذا مكان يحدث قبل وجود الـ Operating System
الآن مع وجود الـ OS بقى حضرتك تقدر تستخدم نفس الـ H.W للاداء اكتر من عملية
يعني هيبيقي عندك برنامج واحد تنفذ عليه عمليات مختلفة عن طريق الـ OS

ذلينا نشهي الـ OS بالسوق بتاعك هتقلاه ودينى اي مكان هو حافظ كل الوجهات هيوديك
يبقى نطلع من دا كلو ان OS هو برنامج يعمل كوسيط للادارة الـ H.W بمرونة

Operating System Views



Operating System is a Recourse Allocator
تخيل معايا كدا انك بتطبع pdf من 100 صفحة
وجبت قبل مينتهى من الطباعة بتطبع ملف اخر طبيعى
هتلaci كل ملف مطبوع لوحده مفيش تداخل
ودا حصل لأن الي بيدير امر الطباعة هنا هو الـ OS

Operating System is a Control program
اوقدات كدا وانت شغال علي جهازك تلاقي برنامج هنج منك او فيه مشكلة فسعتها بتقفلو
وطبعاً المسئول عن التحكم في دا هو الـ OS

Karnal and System Programs

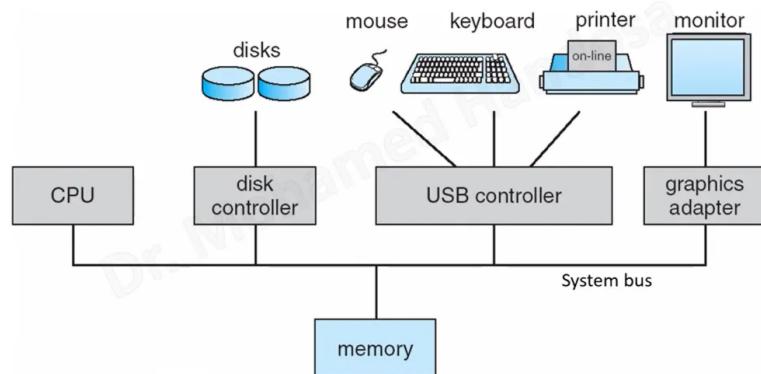
Operating System = Karnal + System Programs
دا قلب نظام التشغيل بدونو لن يعمل: Karnal
التي تكون مدملة مع نظام التشغيل يعني انت مش بتعملها Download
System Programs : Application Programs : Download لها تقوم بعمل لها

Computer System Organization

Interrupts

Storage Structure

I/O Structure



الصورة توضح الـW.H

وحدة المعالجة ودي المسؤولة عن معالجة البرامج المختلفة: CPU:

هنا المقصود بيهها الـRam: وهنا اي برنامج عشان يستغل لازم يحصلو

هو المتحكم في الـDisk Controller في Disks طبعاً الـMother board دا حته جديدة لازم حاجة تشغلهها:

طبعاً لا يخلو جهاز كمبيوتر من مخارج الـUSB وهنا بنوصل سواء الكيبورد او

Graphics adapter : mouse او鍵盤 وغیره على الشاشة: ودا كرت الشاشة المسئول عن عرض الرسومات والنصوص

حتى الان كل حاجة سليمة بس لوحدها

هتتصل ببعضها عن طريق مجموعة اسلام هنسميها System Bus

عن طريقها تترك البيانات بين الـW.H المختلفة

طبعاً تسمع عن الـMother board وهي موجود فيها الـSystem Bus وهي تختلف سعرها علي حسب سرعة

Mother Board وبردو الـUSP Controllers مرفرفة بالـSystem Bus

ومعظم الاحيان بيقي فيها الـGraphics adapter وممكن يكون لوحده او كيوب في مكان معين فيها

وطبعاً عشان تتحكم في مين يقرأ ويكتب

في الـMemory Controller وهو الـMemory عشان ميحصلش تداخل

لان زي ما واضح في الصورة عندي واحد الـUSP Controller علي الـmouse & Keyboard & printer

Computer Startup



سألت نفسك قبل كدا لما بتضغط عن زر ال Power ازاي جهازك بيشتغل ؟

يتم توصيل كهرباء الي ال Mother board

وتفضل ماشية فيها لحد متوصل الي ال Processor سعتها هو عارف

ان اول حاجة بروح عليها هي ال Rom in Mother Board

بداخل ال Rom يوجد برنامج هو ال Bootstrap فيتم تنفيذه

يقوم وبالتالي: Bootstrap Program

يفحصل ال H.w ما اذا كان متاح ام لا.

يبعد عن ال Os يدور مثلا في C or D partition ولما يلاقيه

يعمل Load to Kernel in Ram وادا تم عن طريق الي قال تعمل كدا هو ال Bootstrap والمنفذ

يبدأ ال CPU يقول لل Kernel امر من ال Bootstrap وشد طرف الخيط وكذا مهمتي انتهت.

ال Bootstrap مش برنامج عادي لانو موجود علي الذاكرة الخاصة بالقراءة فقط

وعشان كدا مبقاش اسمو software

Bootstrap is a Firmware

Software: برنامج اقدر اغير في الاوامر بتعتنو

Firmware: برنامج مقدرش اغير في الاوامر بتعتنو

Hardware: مكون صلب لا يمكن تغييره حتى جديدة

Interrupts



Operating System is a Event Driven

نظام التشغيل قائم علي فكرة ال Interrupts

يعني هو قائم علي انك تطلب منو ينفذ حاجة زي تضغط علي تطبيق عشان بفتح

هو اشارة ترسل الي المعالج زي مثلا ضغطة الموس Interrupt

نفترض مثلا انت مشغل فيديو علي جهازك

وقمت مدرك الموس في اللحظة دي هيتم توقف معالجة ال Process

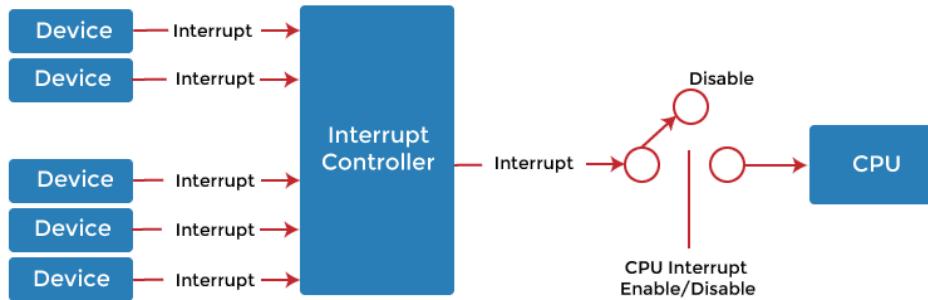
لحد مينفذ ال Trap هو اي الي هيتنفذ بالضبط

هيتنفذ كود هنسميه Interrupt service routine موجود في الذاكرة

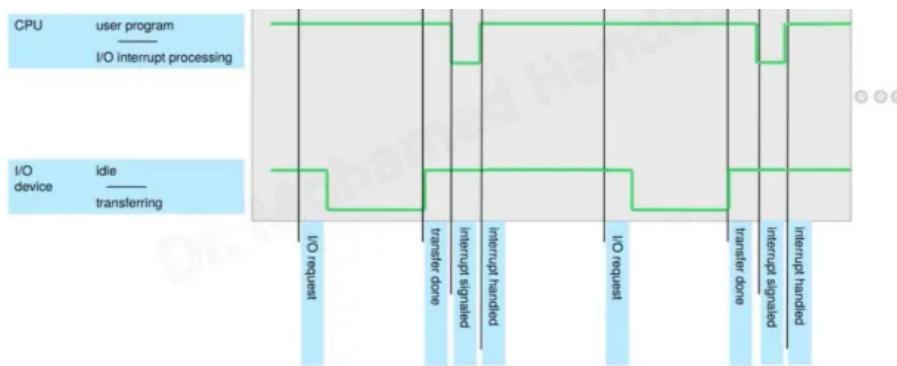
Types of Interrupt

H.W Interrupt : هو المسؤول عن اصداره هو ال H.W

Software Interrupt: زي مثلا ال Trap or Error من برنامج معين



لو مثلاً مشغل برنامج وفجأة البرنامج طلب تدخل حاجة من الكيبورد فالي هيتم ان هيروح يخلص الطلب الاول ثم يرجع يكمل البرنامج من مكان وهو واقف



Storage Structure



Rom : ثابتة للقرأة فقط

EEPROM: نضع بها الـ Firmware ولكن يمكن تدديث محتوياتها هيجي فبالتالي حالاً طب لدام بتتعدد Ram؟ ليه منقلش انها ؟

لان بيبقا ليها حد اقصى لعدد التحديثات التي يمكن عملها عليها وبعدها بتترمي

طب دا هيفدني في اي ؟

الحقيقة ان على اللاب بتاعك الي فاتح منو دالا جوو Mother board ففي لانها بتمكنك تدتها

لان ساعات تستري جهاز وبعد مدة تلاقي نازل تحديثات الـ Firmware بما فيهم الـ H.W.

Main memory (Ram) خاصة بالتخزين المؤقت واي حاجة عشان تشتعل لازم يحصلها load ابا داخلها:



Long-Term Scheduler: يتم نقل الـ processes من الـ disk to Ram طريقة:

Medium-Term Scheduler: Ram be Full لما الـ العكس

Short-Term Scheduler: processes from ready queue in CPU

Instructure Execution Cycle



Fetch: Instruction Register نودية على الـ CPU وي تخزن في الـ Instruction Register

نفترض مثلاً ان دا الكود الي عوزين ننفذ $Z = X + Y$

Decode: نفخ تشفير الكود عن طريق نمسك X ونروح علي عنوانه في الـ Ram

Data Register في another Register مثلاً 1 ونخزنها في

Data Register مع 2 ونضع قيمته في

Execute: يقوم جامع ومذزن الناتج في مثلاً Result Register

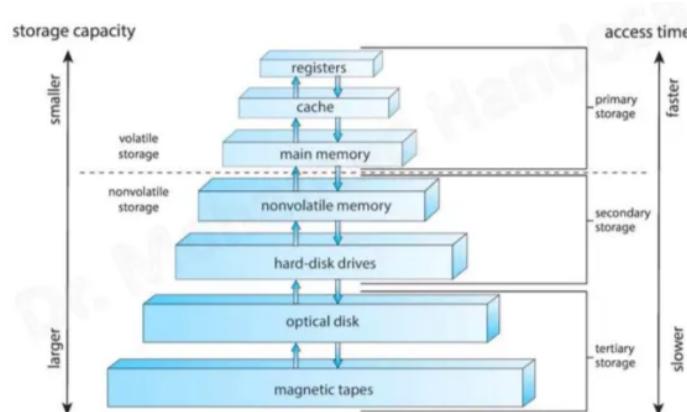
نرجع للكود نلاقيه بيقول خزن في Z التي هي عنوان في الـ Ram

Secondary Storage

نظر لان الـ Ram هو جهاز ذاكرة صغير فتحتاج ذاكرة اكبر وتفضل محتواها عند انقطاع التيار

Hard Disk وهو

Storage Hierarchy



و دي انواع الذاكرة المختلفة كلما ارتفعنا كلما زادت السرعة والسعر وقلت المساحة

وكل ما تنزل كل مالمساحة تكبر والسرعة تقل والسعر يقل

Caching



هي الذاكرة التي ننسخ بها البيانات المكررة بدل مكل شوية اقرأها من Ram هقرأها من الـCache يعني البيانات هتبقي لسه في الـRam بس وخددين نسخها منها في الـcache عشان هي اسرع

هي قطعة الـW.H الي بننسخ بها البيانات التي ساستخدمها مارا: Cache: ظليها في بالك كدا اول متسمع عن نسخ بيانات من مكان الي : Caching اخر في اي نوع ذاكرة يبقى Caching طول مايل معمليات مثل حصل تحديثات على بيانات Cache ميهمناش كدا كدا بنأخذ البيانات من الـمشكلة بتظهر لما بتتعملي ععاوز افضي منها عناصر عشان اخزن غيرها لازم اي بيانات اتحدثت في الـRam نزوج على الـcache

Cache Coherency:



بساطة لو عندك مثل اثنين منprocessors cache كل واحد ليه نفس الـRam وعندك شغلين على $x=8$ وحصل تحدث على قيمة الـ x الي في احدهما هل لوحظ انه استخدمت قيمة x القديمه هيكون صحيحة؟ طبعا لا يبقى لازم اي تحدث يتم في اي من caches ينعكس في الباقي الموضوع دا مهم جدا لو حابب تفهم الموضوع من جهة التعامل مع الـDB الـUSB دا وافي https://www.linkedin.com/posts/eslammohs7n_dotnet-cash-database-activity-7267231821631737857-F_ux?utm_source=share&utm_medium=member_android

I/O Structure



مش احنا قلنا قبل كدا ان كل device (H.W) has a Controller
الجديد بقى ان كل Controller has a local Buffer طب ليه ؟
لو انت مثلا بتكتب على الكيبورد وفي اللحظة دي كانت
الـ memory مشغولة مع الـ CPU فكدا الي بتكتبو هيضيع
عشان كدا every controller has local buffer
يذن فيها مؤقتا لحد ما تجهز ويعملها الي فيه

Device Driver

احيانا تشتري طابعة وتحتاجي توصلها بالكمبيوتر متشتغلش ليه ؟
لان هي محتاجة تعرف يعني اي ؟
يعني محتاجة كود فاهم الطابعة شغاله ازاى فلما نظام التشغيل ينفذ الكود دا يقدر يطبع سعتها
الـ Device Driver دا او التعريف اسمو

I/O Operation



1. Controller Registers ويطلع الـ values بـ executes the Device Driver
2. Controller registers في الـ يقرر هي عمل اي :
3. Controller local buffer لـ و هطبع حاجة ينقلها الي الـ يقوم بنقل البيانات مثلا
4. Controller generates an interrupt to inform the device driver

يعني بعد منخلص على الطابعة
مثلا هنبعث اشارة الي الـ CPU عشان توقف تنفيذ الكود الي كان شغال عليها
الي بيعرفنا الطابعة مثلا شغاله ازاى الي هو اي ؟

Device Driver

يعني لما نقول بعثنا Interrupt to CPU دا حصل عشان نقطع تنفيذ الـ Device Driver

5. Device Driver return control to Operating System and data

ازاي هيرجع له

ممكن اثناء معاجة الـ Device Driver دخلنا حاجة من الكيبورد فهنا التحكم كان مع الـ Device Driver
فلازم ترجع البيانات دي لنظام التشغيل عشان يتولاها وكمان هيرجع لها الحالة الي وصلناها مثلا تمت الطباعة

هنا في حالتنا الي هيمنقل مثلا البيانات الي بتكتبها عالـ keyboard مثلـا
CPU هو From local buffer to memory

لو كنت بتسمع فيديو وكتبت درف عالـ keyboard الي بيعملو وبروح ينفذ الـ Interrupt
دا بالنسبة للبيانات الصغيرة لكن لو البيانات كبرت الموضوع هيختلف سعتها هنعمل التالي

Direct Memory Access(DMA)



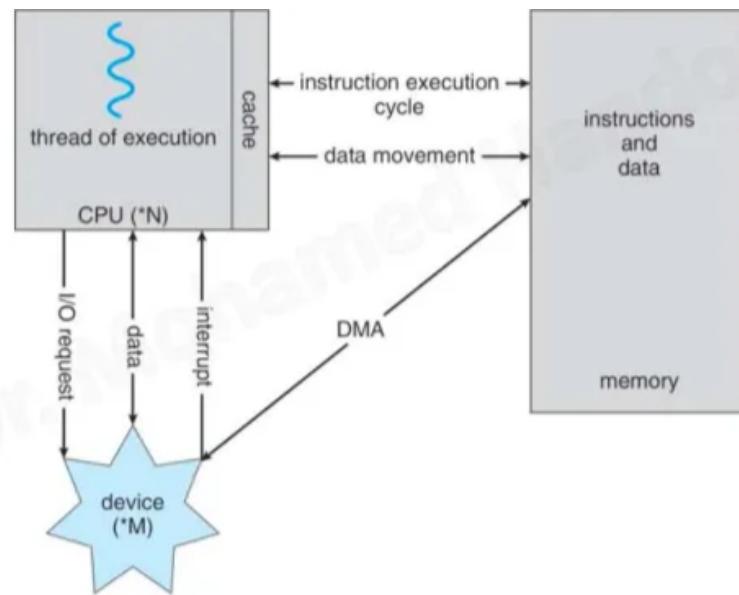
يعني CPU يقول للDevice Controller انقل البيانات الي انت عوزها بنفسك ودا مع البيانات الكبيرة بس هنا CPU مش هيعرف اي حاجة عن البيانات الي اتنقلت دي طب هيعرف منين ان هو خلص ام لا؟ يقوم الـ interrupt per block of data Controller باعتال CPU وهو نقل قد اي

هتشوف الكلام دا لو مثلاً بتنزل فيلم فبتلاقي ان بيفصلت تم تنزيل مثلاً 70%

وشوبة بيقى 75%

الي حصل ان الـ interrupt هبيعدت للكل كل من البيانات مش كل byte عشان يعرف الـ CPU وسعتها يبدأ يحدذلك الي بيحصل

بس مين المسئول انو بيعت البيانات ايوه صح :الـ Device Controller



يبقى نطلع من هنا ان و بيانات صغيرة هتنقل عن طريق الـ CPU
اما لو بيانات كبيرة هتنقل DMA يعني عن طريق الـ Device Controller

Computer System Architecture

Single Processor System



يعني كمبيوتر في معالج واحد(CPU)

هل تعلم ان ال Processor هو Disk Controller بس شغلتو ال Processor وال Processor هو Graphics Adaptor وال Processor هو CPU Controller وال Processor هو كيبورد مثلًا لكن ال Processor المسئول عن معالجة البرامج هو ال CPU

Specific-Purpose Processor :

يُعمل عملية وحدة متعددة ليه زي مثلاً ال Disk Controller & Graphics Adaptor

General-Purpose Processor :CPU ال هو ال الذي ينفذ اي اوامر تدهالو

Single Processor System has one General-Purpose Processor

Multiprocessor Systems



Multiprocessor Systems has more General-Purpose Processor

الاكثر من معالج بيقو موجودين علي نفس الجهاز ويشغلوا مع بعض بالتوازي ومشتركين في نفس الذاكرة وعشان كدا بنقول ان

Multiprocessor Systems known as a parallel systems and tightly-coupled systems

Advantages of Multiprocessor Systems:



1. Increase Throughput

يزود الانتاجية لأن لو مثلاً عندك برنامج فيه 100 Instruction

لو اتقسموا على معالجين مثلًا هيكون اسرع من لو اشتغلوا على واحد

بس الامر مثل ورديه للدرجادي يعني لو ال 100 دول بيخدو 100 ثانية وشغلهم على معالجين مثل هيتنفدو في 50 ثانية بالضبط لا هيتنفدو في 50 ثانية وشويه زيادة الشوية الزيادة دول هما الوقت الي جهازك بيخدو عشان ينظم ان المعالجان دول يشتغلوا سوي ويقسموا الشغل ما بنهم

2. Economy of Scale

بدل متروح تشتري كمبيوتر كمان عشان يبقي عندك اكتر من معالج وتقسم الشغل ما بينهم فانت هيبيقي هنا عندك اكتر من معالج في نفس الجهاز وبالتالي التكلفة هتقل كتير

3. Increase Reliability

دلوقتي لو عندك جهاز فيه اكتر من معالج هتعتمد عليه اكتر من لو في معالج واحد لأن لو واحد اتحرق الباقيين شغلين ويبعملو شغلهم

System Reliability



1. Graceful degradation

يعني بيقي في انحدار في الاداء بشكل منطقي يعني لو مثلا عندك جهاز في 4 معالجات المتنقى لو واحد اتفرق يقل الاداء بنسبة 25%

2. Fault tolerant Systems

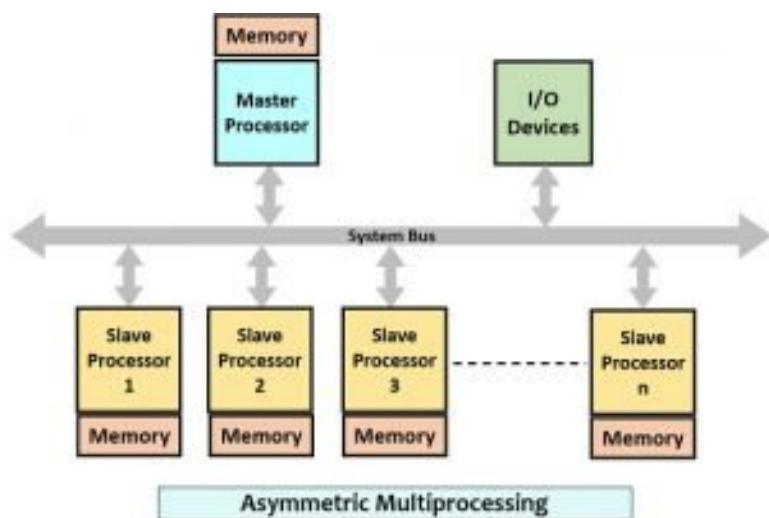
دي الانظمة التي تقاوم المشاكل يعني لو معالج اتفرق مفيش مشكلة عندي غيره لأن بيقي عندنا معالجات زيادة بتشتغل لو حصل مشكلة

Types of Multiprocessor Systems



1. Asymmetric Multiprocessor Systems:

بمعنى المعالجات الي عندي مش كلها متماثلة يعني بعمل معالج منهم هو الرئيس بتعهم وهو يستقبل العمليات ويدأ يوزع على باقي المعالجات الي سعتها هيبقو زي العمال



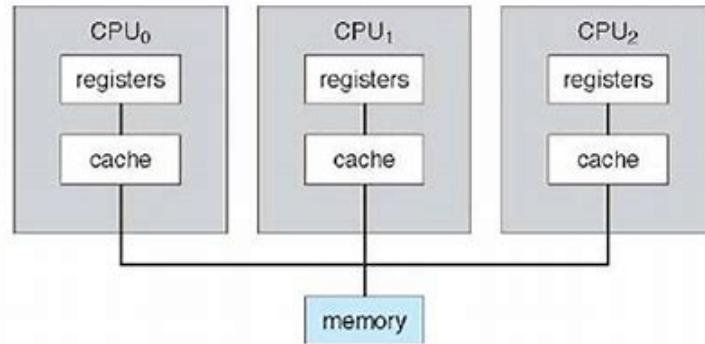
ودا بنستخدمو لو عندنا معالجات كتير وعاوز اتنظم الشغل بنهم

2. Symmetric Multiprocessor Systems (SMP):

هنا مفيش رئيس كل المعالجات زي بعضها الشغل يجي يتعاونو كلهم عشان يخلصوا

ودا بنستخدمو لو عندي عدد معالجات مش كتير لأن لو هما 2 مثلا مش هنستفيد حاجة لما نعمل واحد رئيس علي الثاني

Symmetric Multiprocessing Architecture

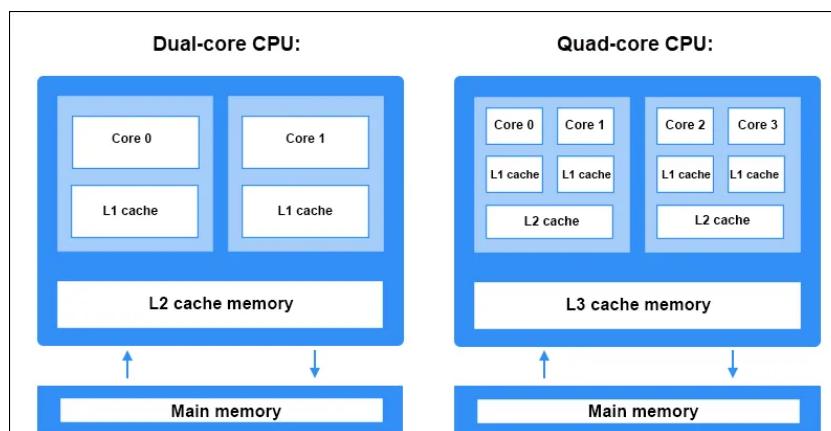


Multicore System



لو بصيت في الصورة السابقة ان عندي اكتر من معالج وكل واحد ليه cache الخاصه بي
فكان لما يعوزو يتبدلوا البيانات بيتبدلوا عن طريق الـSystem Bus

فالله ليه مبيقاش عندنا شريحة وحدة فيها اكتر من معالج وهو دا الـMulticore System
مشتركه فمش هيتبدل عن طريق الـSystem bus هيكلاو بعض عن طريق الذاكرة المشتركة
وبعد استهلاك الطاقة هيقل عن لما يبقي عندي كل حاجة لوحدتها وشنقل في الـSystem bus





Clustered Systems

هنا هنجيب اكتر من كمبيوتر

كل جهاز ممكن يكون فيه اكتر من معالج وهنوصلهم كلهم بنفس الـ

LAN (Local Area Network) وهيتبادل البيانات عن طريق

loosely couple systems ودا الي اسمو

Types of Clustered Systems

1. Asymmetric Clustering

هنا هيقي عندها اكتر من

ولكن واحد بس شغال والباقي احتياطي لو حصل مشكلة يدخل واحد تاني بدالو

2. Symmetric Clustering

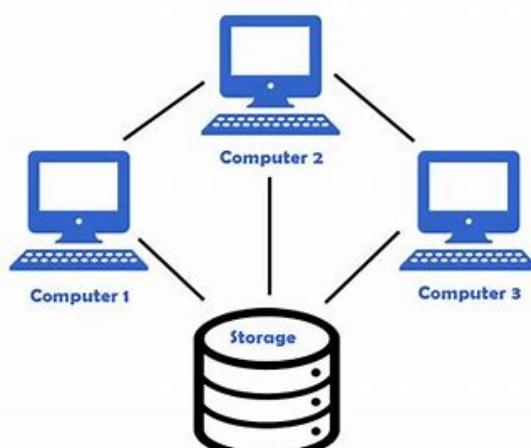
All Servers Working

فكل الشغل هيتقسم عليهم كلهم

ولكن عشان نقسم التطبيق بتاعنا عشان يشتغل على أي نوع من أنواع الـ

Parallelization لازم نراعي في الكود بتاع التطبيق إنه يكون شغال

يعني الكود لازم يكون قابل للتقسيم، مش كلهم معتمد على بعضه ومنش شغال بالتوازي



Clustered System

Operating System Operations



نظام التشغيل شغال بوضعين عشان يحمي الجهاز من الاختراقات

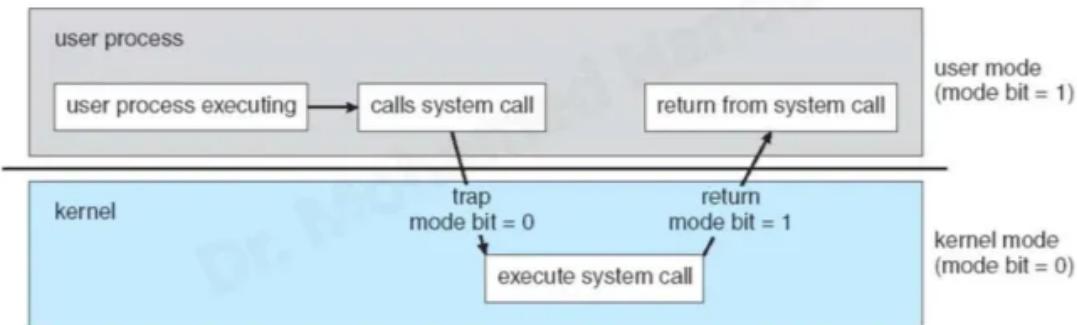
User Mode : لما البرنامج يكون في الوضع دا بيكون ليه صلاحيات محدودة

Kernel Mode : بيكون ليه صلاحيات كاملة والي بيفعل هنا هو نظام التشغيل

Karnal mode لو كان 0 بيقي

وسعتها الـ CPU يمنع تنفيذ البرنامج وينادي علي نظم التشغيل هو اللي ينفذ العملية دي لأنها محتاجة صلاحيات

اما لو 1 بيقي في الـ User Mode وهنا تنفذ عادي بس بصلاحيات محدودة



هنا لو مثلا عندك برنامج شغال ومثلا طلب حاجة محتاجة صلاحيات نبدأ ننادي على الـ System Call ويتم تدوير

User Mode ليعتبر الـ Kernel Mode ويشتغل نظام التشغيل وبعد ميخلص يقول تاني لو خرج الـ User Mode

بيقي اللي بيتحكم في تغيير الوضع هو نظام التشغيل



Multi-Mode Operation

معظم الاجهزه بيقيا فيها وضع اضافي وهو وضع يمكنك من انشاء نظام تشغيل كمبيوتر افتراضي وتنزيل عليه برامج بقى وتعيش زي برنامج ال Virtual Box

تنزلو ثم تنزيل عليه نظام تشغيل وتحددل نسب ال W.H. الي عاوز تدهالو
والوضع دا اسمه VMM

Timer

عملناه عشان حل مشكلة ال infinite loop

لان ممكن برنامح يدخل ال CPU

ويفضل شغال وماسك عجلة القيادة وباقى البرامج عوزة تدخل
هتقليل طب ما نظام التشغيل يخرجها

هقلنك نظام التشغيل مش هو الي سايق هنا عجلة القيادة الي هي CPU

الي مسكتها هو البرنامج الي جوة وعشان حل المشكلة دي عملنا عدد يدي لكل برنامج وقت محدد وبعد ما وقتو يعني اي يعني يخلص نجعت Interrupt علي نظام التشغيل ولما يجي اما يديلو وقت زيادة او يطردو

وطبعا عشان البرنامج ميغيرش في ال Timer Mode bit = 0 يعني الي يقدر يغيره هو ال OS

Computing Environment

Batch Systems



زمان كان عندنا اجهزة الكمبيوتر خدمة Mainframe

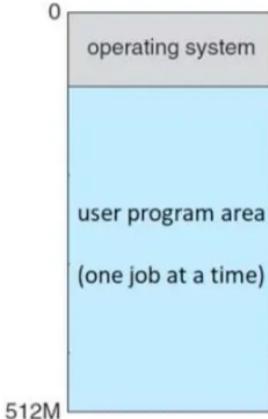
كنت عشان تنفذ حاجة معينه بتحط البرنامج الي عاوز تنفذو على Card وتروح تسافر
لحد متوصل لمكان ال Mainframe

تدخل جوة تلاقي واحد اسمه ال operator يأخذ منه card ويقولك تعال الي بعد مدة عشان تستلم نتائج التنفيذ وبعد كدا ال operator ينظم الترتيب ويحط ال cards جوة الكمبيوتر ويدأ ينفذ
وال operator يروح يشوف وراه اي طب مين هيغير الكروت الي تنفذ ويدخل غيرها ؟

عملو برنامج بسيط يعمل الموضوع دا والبرنامج دا كان بدايات ال OS كان بيطلع مطبوع علي ورق
وال output من الطابعة وكمان ال OS كان بيطبع نسخة من محتوي ال memory

عشان المبرمج الي عامل البرنامج بيصل على النسخة دي الي بتقى مجموعه من 01
يكتشف الاخطاء لو كان في مشكلة فكان الموضوع صعب جدا

المشكلة في النظام دا ان لازم يخلص مهمة تلو الاخر يعني لو برنامج شغال وطلب اى /n
هيوقف كل حاجة لحد مجيب المطلوب ويقى يكمل وبالتالي مستخدمناش من السرعة الكبيرة لل CPU
وعشان كدا ظهر النظام التالى لمحاولة حل بعض المشاكل

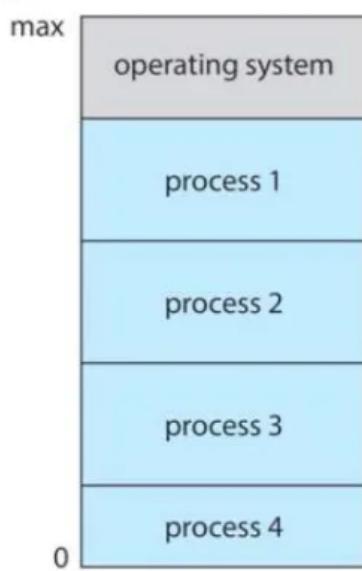


Multiprogramming Systems



قاموا بتطوير نظام التشغيل لمحاولة الاستفادة بشكل اكبر من سرعة الـCpu الكبيرة عن طريق اصبح تنفيذ الـi/o مهمه نظام التشغيل ولما برنامح يطلبـi/o اندخل برنامح اخر علي مهو يخلص 1.2.scheduler يتم اختيار العملية التي ستتلقى CPU عن طريق مكون اساسي في نظام التشغيل وهو الـscheduler وبردو لو عملية طلبـi/o وخلصت وعوزة تنسن بتحط مع باقي العمليات الي موجودة ويتم اختيارها من خالله

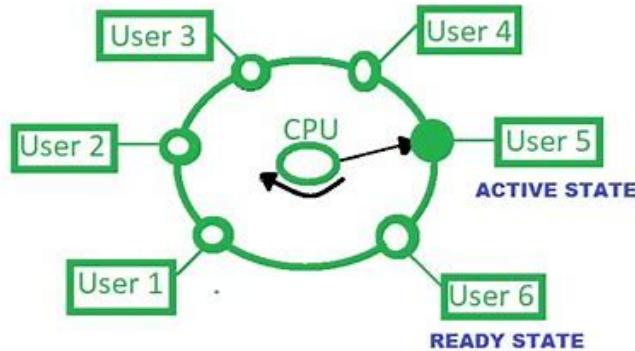
وعشان مفيش process متدخل على اخري في الذاكرة بردو نظام التشغيل كان بينظم بينهم عشان موارد process للتداخل مع اخري زي مثلا في حالات طباعة اكتر من عملية في نفس الوقت 3.4. نظام التشغيل بردو هو الي بيدير الموضوع دا يبقى كدا استخدنا من سرعة المعالج الكبيرة ولكن لسه بردو شغالين بالـcards يبقى بس عشان متنوهش هنا البرنامج هيطلع من الـCPU وتدخل غيره لو طلب i/o or finish وفي مشكلة كمان ان ممكن برنامج يفضل جوة علي طول لو مطلبسـi/o وسعتها هيدخل في infinite loop وعشان يحلو المشاكل دي ظهر النظام التالي



Time Sharing Systems

هنا النظم تفاعلي لانك بتحس ان اكتر من برنامج شغلين مع بعض طب ازاي ؟
عن طريق ان هو بيستخدم Timer بمعنى يدي لك كل برنامج وقت معين ولما وقتو يخلص يدخل غيره
يبقى هنا عشان البرنامج يطلع من المعالج لازم يحصلو التالي
وقتو يخلص او طلب I/O او خلص

Time Shared Operating System



Desktop Computers

جهاز الكمبيوتر الي انت بتسخدمو يعني ودا بينزل عليه اما
نظام تشغيل windows / mac / linux وغيرها

Mobile Computing

هنا اهم حاجة في نظام التشغيل دا المحافظة على طاقة البطارية لوقت اطول
أنظمة التشغيل
android / Apple

Distributed Computing

هي مجموعة من أجهزة الكمبيوتر متوصلاً بعضها عن طريق شبكة وعندنا أنواع الشبكات

على نطاق واسع مثل شبكة الانترنت: WAN

مساحة اصغر مثل مخطية جامعة: MAN

مساحة اصغر مخطية مثل مبني: LAN

شبكة شخصية زي موصل تليفونك بيلوتوث دد: PAN

Types of Distributed Computing

Client-Server Computing

Peer-To-Peer

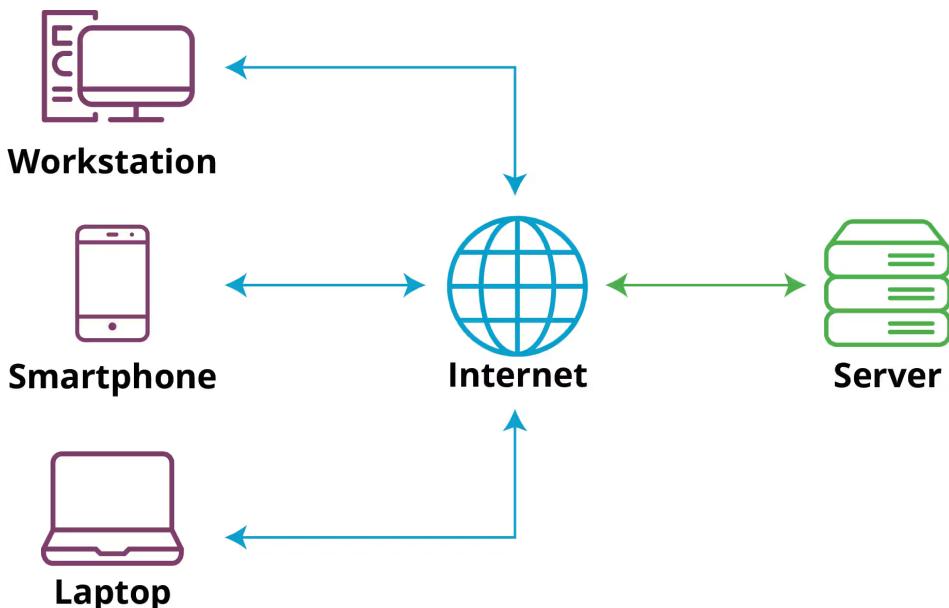
Client-Server Computing

انت كـ Client لو بتبحث عن حاجة في جوجل Request لـ Server

وهو يعمل مجموعة حسابات عشان يرجعلك بالرد وسعتها بنسميه Compute-Server

عندك بـ File-Server بترفع عليه ملفات وتنزل من عليه ودا بنسميه ondrive

وطبعاً الاتصال بين الـ client and server طريق الـ Network



Peer-To-Peer

هنا مفيسش servers مخصوص لأن فعلياً كل جهاز جوة الشبكة يلعب دورين client and server

ودا عن طريق ان بنوصل مجموعة من الأجهزة في شبكة ولو مثل واحد بينزل ملف هناخد من كل جهاز جزء ويبيقي ودا مميز في ان لو جهاز شغال كا server كا مش هيسكب مشكلة هكملي بياناتي من جهاز اخر

اما في نظم الـ Client-Server وقع على توقف العمل



Cloud Computing

انت مثلا لو بتصمم برنامج لبيئة عمل مش هيزيد عدد الناس الي هيستخدموها فبتستخدمها على server
لان هو بيبقى ثابت لعدد معين من الطلبات الي يقدر يعالجها وبعد العدد دا مبيقدرش يتعامل مع حد زيادة
طلب لو انت بتعمل برنامج تجاري مثلاً فطبيعي ان في عدد عملاء مش هعرف احدهو علي المدى البعيد
وهنا يجي دور الـ Cloud Computing

فكترو مبنية على مبدأ الـ Virtualization بحيث اي زيادة في عدد الطلبات
يبدأ يزولك تلقائي من غير متلاقي السيرفر وقع وفي اخر الشهر مثلاً يقالك استخدمت قد كدا تدفع كذا

Types of Cloud Computing

ودا الي انت بتروح تأجره من شركة معينة: public

ودا بيبقى خاص بالشركة نفسها لبياناتها الخاصة: private

مزيج بين الاثنين مثلاً بيانات العملاء تبقى private وحجات تنيه public

Levels of Cloud Computing

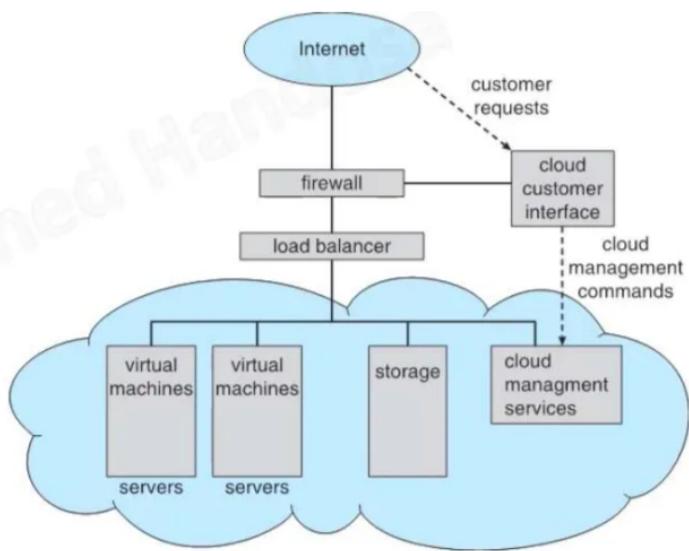
وهنا تلاقي شركة بتقالك هأجلرك كمبيوتر بامكانيات عالية تستخدمو عن طريق جهازك الي بيقى امكانياتو مش كبيرة يعني بتتأجرلك H.W

هنا نفس الكلام بيتأجرلك H.W Platform as a service:

وبيزولك ان بينزلك مجموعة من الـ softwares اللي معك

تستخدمها زي VS,SQL SERVER ,NET

هنا يقالك احنا عملين برماج تستخدمنا : Software as a service online



دا الي يحمي البيانات من الهجمات عن طريق فحص الـ Firewall:requests

Load balancer : بيعتكم للسرفر المناسب ليك :

Real-Time Embedded Systems

ودا الكمبيوتر المدمج في مثلا غسالة او موجة صواريخ او السيارة

للزيم الحاجة تتم في معاد محدد لأن اي تأخير هيسبب عواقب كبيرة زي توجيه الصواريخ: real time systems

ودا بنسميه sold - real time system

في نوع كمان الي التأخير مش هبيقي مشكلة جامدة يعني زي لو بتلعب العاب فيديو

ودا بنسميه soft-real time systems



Chapter2(Operating Systems Structures)



1. Operating Systems Services
2. System Calls
3. Types of System Calls
4. operating system structure

Operating Systems Services



اي الخدمات الي يقدمها نظام التشغيل لمستخدم الكمبيوتر ؟

1. User Interface

ببوفرك طريقتين للتعامل

ودي الايقونات الي انت بتضغط عليها يفتحك البرنامج: (GUI)

Command Line Interface مثل: CMD in Windows

وهنا بتكتب اوامر عشان تنفذها

2. Program Execution

عن طريق ان نظام التشغيل بيعمل it

البرامج الي بتشتغل ممكن لما اعوز اقفلها ودا طبيعي وممكن بردو يتوقف بشكل غير طبيعي

زي مثلاً لما برنامج يهنج منك تفتح ال Task manger وتعملو End task

3. I/O Operations

نظام التشغيل هو بس ليه صلاحية ينفذ هذه العملية

4. File system manipulation

بيمكنك من قرأ وكتابة الملفات المختلفة

5. Communications

ودا لما تكون process عوزة تكلم اخرى

6. Error Detection

نظام التشغيل هو المسؤول عن التعامل مع الأخطاء ويبداً يطلعك رسالة بالخطأ او يحلها بنفسه من غير مان تحس دوا لو مثلاً في محاولة اختراف عليك نظام التشغيل بيمعن ان دا يحصل وانت متبلاش حاسس

اما الأخطاء الي بتحتاجك تعالجها بيطلك رسالة زي مثلاً كنت بتطبع ورق وفي ورقة علقت في الطابعة

هيطلعك رسالة عشان تشيل الورقة

Additional Operating System Functions

في بعض الحالات الاضافية الي نظام التشغيل يقوم بيها زي

1. Resource Allocation

عملية حجز الموارد لكل process

2. Logging

عن طريق عمل recording لكل process بتحصل المعلومات دي ممكن احتجها مثلاً لو بتتبع خطأ

وعاوز اعرف مين البرامج الي اشتغلت في وقت معين وبالتالي هكتشف الخطأ اسرع

3. Protection and Security

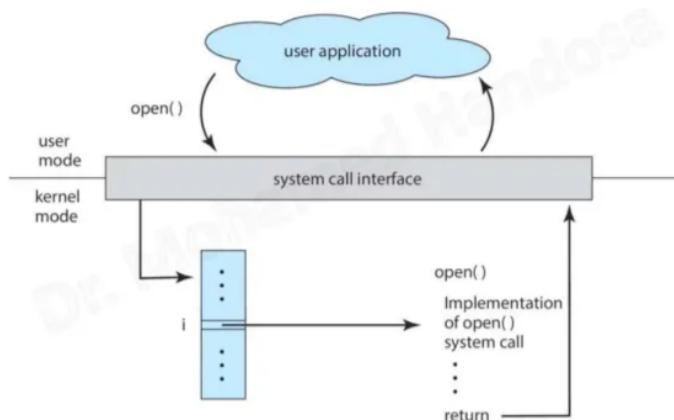
عن طريق التأكد ان اي وصول للموارد يتم بشكل نظام التشغيل علي درايه بييه ومتدكم بييه:

يكون نظام التشغيل قادر علي حماية الجهاز من اي هجمات خارجية:

System Calls

بساطة هنوفرواجهة مكونة من مجموعة دوال تسهيلات على المبرمج فيقدر يستخدمها ويطلب الـ O/A المختلفة

او تفتح ملف او تكتب علي ملف وهكذا بمعنى الاوامر الي يحتاج نظام التشغيل ينفذها هي هنادي الدالة الي بتعمل كدا ونظام التشغيل ينفذها



هنا عندك برنامج عاوز يفتح ملف فهندول من الـ user mode to kernel mode

عن طريق هنغير الـ mode bit from 0 to 1

ويبدأ يروح علي الـ interrupt vector to find Interrupt Service Routine

ثم ينفذ العملية المطلوبة وبعد كدا يحول للـ user mode

طب مش احنا قلنا ان دي دوال طب ما معنكم ابعطلهها ودا ازاي ؟

عن طريق 3 طرق

1. passing parameters in CPU Registers

زي مثلاً بنفتح ملف فالملف دا ليهها فھيتخزن في Register

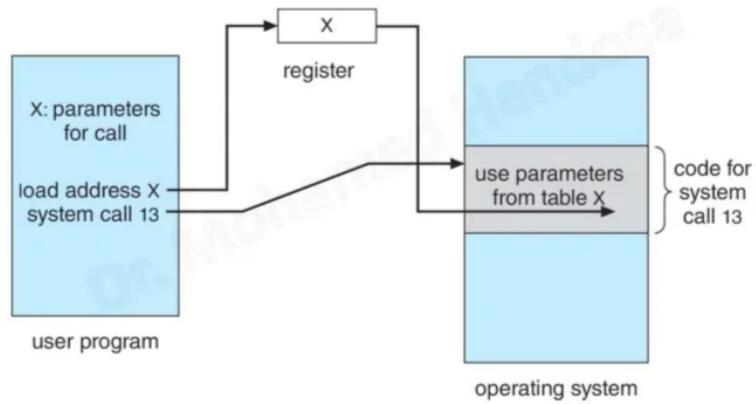
ودا مع البيانات الصغيرة لأن زي ما عرفين الـ register مساحتو صغيرة

2. Block in Memory

لو الـ parameter دا كبير فهندحتاج نخزمو في الـ memory ونخاند عنوانو في الـ

3. Stack

ودا افضل حل بنخزن الـ parameters فيها ونظراً لفكرة عمله أنها شغاللة Last in - First Out



وذا مثاً للاستخدام طريقة الـ **Block in Memory**

Types of System Calls



Process control

create process, terminate process
lend, abort
load, execute
get process attributes, set process attributes
wait for time
wait event, signal event
allocate and free memory
Dump (Clear) memory if error

Debugger for determining **bugs**, **single step** execution

Locks for managing access to shared data between processes

File management

create file, delete file
open, close file
read, write, reposition
get and set file attributes

Device management

request device, release device
read, write, reposition
get device attributes, set device attributes
logically attach or detach devices

information maintenance

get time or date, set time or date
get system data, set system data
get and set process, file, or device attributes

Communications

create, delete communication connection
send, receive messages if **message passing model** to **host name** or **process name**

From **client** to **server**

Shared-memory model create and gain access to memory regions
transfer status information
attach and detach remote devices

Protection

Control access to resources
Get and set permissions
Allow and deny user access

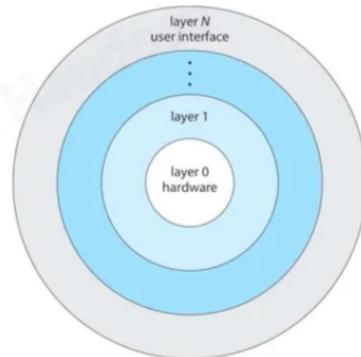
operating system structure

Monolithic Approach

في هذه الحالة نظام التشغيل كتلة واحدة مفיש تنظيم ونظراً لذلك عملية الاصابة والتعديل تكون صعبة جداً الميزة ان التعامل جوة ال Kernel يكون سريع لأن هنا اي حد عاوز يعمل حاجة بيعملها على طول مفيش تنظيم والنظام دا بيقي **Tightly Coupled** لأن نظراً للارتباط القوي اي تعديل في اي حته هيسبب مشاكل

Layered Approach

في هذا النظام بنقسم نظام التشغيل الي طبقات وبالتالي عمليات التعديل والاضافة اسهل بكثير ولكن التعامل هيكون ابطأ لأن بدل مكنت مع ال kernel تفروج تنفذ علي طول لا هتنقل بين اكتر من طبقة فوق عشان تنفذ وكمان لزم المبرمج ينظم ترتيب الطبقات بحيث مفيش طبقة تحت تنادي على طبقة فوق لازم يكون منظم والنظام دا **loosely coupled** لأن اي تعديل في اي جزء مش هيسبب مشكلة للجزاء الاخر بيقي كدا طاعنا نظام تشغيل منظم ولكن بطى



- Bottom layer (layer 0), hardware.
- Highest layer (layer N), user interface.

Micro-Kernel Approach



النظام هنا ان بنحاول نصغر حجم ال Kernel عن طريق تحويل الباقي الى **System Programs**

Advantages:

لو هضيف خدمة جديدة مش هاجي جعب ال Kernel هخضفها ك

طب نفترض ان عوزين نعدل في ال kernel

هيبيقي التعديل اسهل لأن حجمو صغير مقارنة بجمة في حالة ال

Monolithic Approach

بردو من مميزات الشكل دا ان لو هضيف نظام التشغيل design. W آخر فهعدل على ال Kernel فقط بدل مكنت هعدل على نظام التشغيل كلو

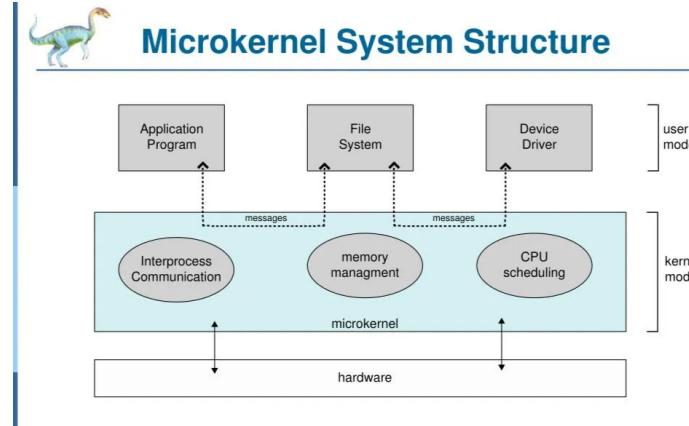
كمان بيوفر security and reliability

لأن نظراً لأن ال kernel أصغر فاحتمال وقوع فيه خطأ هيبيقي أقل من لو كان كبير:

فبقي لو برنامج حصل فيه مشكلة بقفلو لوحدو مش بعيد تشغيل ال kernel كلو من جديد

يتحكم اكتر لأن بقينا نشغل ال system programs في ال user mode

هيبيقي هنا وضعنا الحجات الأساسية في Kernel والباقي حولنا في user space

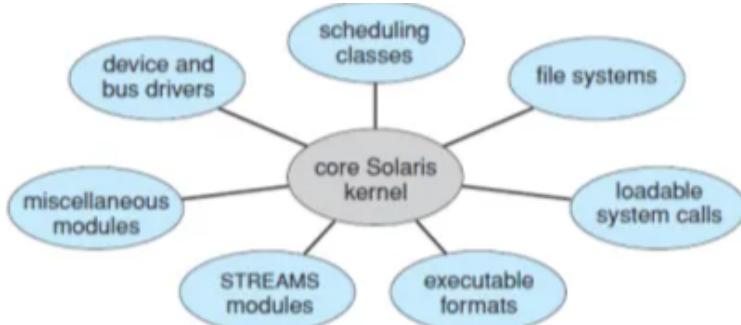


Modules Approach



هنا هنقسم نظام التشغيل الى Modules وكل واحدة هيكون جواها الكود المسؤول عن عملية معينة واثناء التشغيل لما اعوز module معين هعمل load على الكود الي عوزينو يعني كدا كل جزء هيبيقا ليه عمل خاص بيه زي فكرة استخدام ال libraries لابشأء تطبيقات بلغات برمجة مختلفة فلو مثلاً كان في pudge في نظام التشغيل الشركة المصنعة تحلاها وتنزلها كتحديث انت بتنزلو restart كدا لو بتحدث نظام التشغيل يفلك اعمل kernel اكمن قطعت اال kernel الي اجزاء ببدل مبيقي عشان اشغل اي حاجة يتولها اال kernel بقى كل module بيشغل لوحده بس بردو اال kernel ليه وظائف زي

- **Coordination:** التنسيق بين Modules.
- **Shared Services:** الخدمات المشتركة مثل المصادقة أو الإعدادات في Modules.
- **Global Policies:** تطبيق السياسات العالمية الموحدة عبر Modules.
- **Resource Management:** إدارة الموارد المشتركة بين Modules.



Hybrid Approach



وهنا انا هعمل مزيج بين اكتر من طريقة في التقسيم علي حسب احتياجاتنا في نظام التشغيل المصمم
لان بنحاول نستفيد اكتر استفادة من التقسيمات المختلفة

المزايا:

- تنظيم جيد مع مرونة عالية.
- سهولة في الصيانة والتطوير.
- قابلية إعادة الاستخدام.

العيوب:

- قد يكون معقداً في التخطيط.
- يحتاج إلى تنظيم دقيق.

Chapter3(Process)



1. Process Content
2. process states
3. process control block
4. process scheduling
5. operations on process
6. inter-process communications
7. Client-Server Communications



Process :is a program in Execution

في الانظمة القديمة كانت تسمى job

تنفيذ الـ processes يكون في شكل sequential

طلب دلوقتي هنعرف منين الحالة الحالية process current status عن طريق حاجتين

Program Counter:

موجود في الـ CPU شايل عنوان ال instruction الى عليه الدور في التنفيذ register

هتبقي محتوية على باقى معلومات الـ Process : contents of the processor's registers

Process Content

1. Text-Section: Executable Code of Process
2. Data-Section: Global Variable
3. Heap-Section: dynamically allocator variables
4. Stack-Section: Local Variables

واعشان تفهم بعض عالكود دا مكتوب بلغة C#

```
namespace Process_Content
{
    internal class Program
    {

        static void Main(string[] args)
        {
            Process p = new Process();
            // P : store at stack
            // new Process() : store at heap
        }
    }

    class Process
    {
        static int N; // Global Variable will store in Data Section

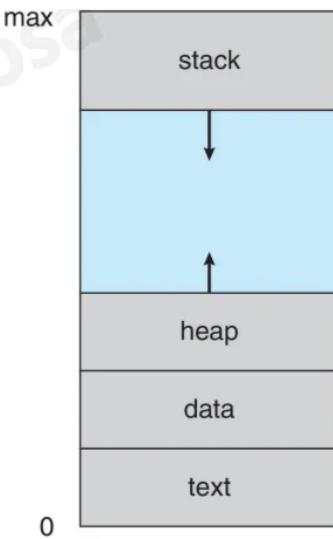
        int[] Numbers=new int[10];
        //Numbers store in stack
        //Heap : اماكن 10 هيرجع

        public void Sum(int x,int y)
        {
            int n1 = x;
            int n2=y;

            Console.WriteLine($"sum = {n1+n2}");

            // x,y,n1,n2 : will store temporary at stack
        }
    }

    // all code of process store in Text-Section
}
```



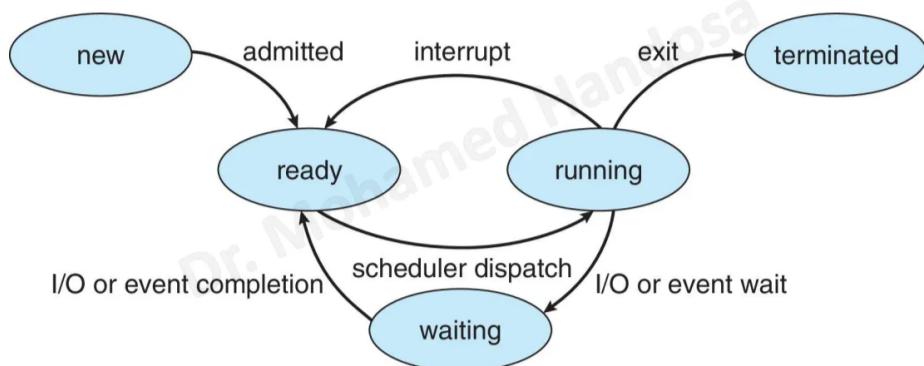
لاحظ ان جمیعهم ثابت ماعدا الـ

لان البيانات بتبقى فيه بشكل مؤقت

process states

الـ process يتمر بحالات مختلفة زي

1. **New:** لسه جديدة يدوب دخلة
2. **Running:** Executing in CPU
3. **Waiting:** حدث معين ان يحصل زي مستويه تدخلها حاجة من الكيبورد
4. **Ready :** CPU بس مستويه يجي دورها هي بردو
5. **Terminated:** خلصت



Scheduler: هو الي هيختار الـ process عشان تتنفذ

Dispatcher: هو الي هينفذ الاختيار

process control block(PCB)



PCB :is a Data Structure store information about Process

شبہ کدا بالبطاقة بناء كل Process

طب اي المعلومات الي بنحتاج نعرفها عن كل process

1. Process State

2. Process Number

3. Program Counter

المقصود هنا القيم الموجودة بها الخاصة بالprocess

عنوان اعرف هي موجودة في الـ memory من اول فین لحد فین: 5. Memory Limits

يبيقي فيها اسمى الملفات الي هي فتحتها: 6. List of Open Files

الاولوية بتأثیرها في الاختیار للمعالجة : 7. CPU-Scheduling Information

هي بتأثیرها وقت قد اي: 8. Accounting-Information

9. I/O status information : i/o device

process scheduling

CPU Scheduler: ready processes from memory

Degree of Multiprogramming :processes in memory

Types of Process

I/O-Bound Process: CPU او أقل استهلاكاً للـ devices

CPU-Bound Process: I/O Devices

والمعروفة دي مهمة جداً بالنسبة ليك كمبرمج وبالنسبة لنظام التشغيل برد طب ليه

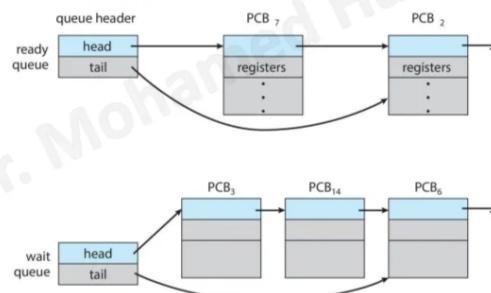
بالنسبة لك مهمه في اختيار لغات البرمجة او بيئه التطوير للبرنامجه او الموقعي بتأثیر

ودي امثلة للتوضيح

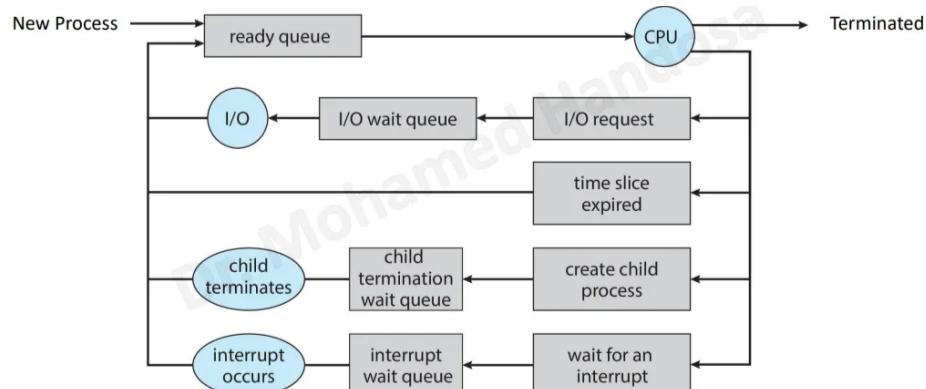
النوع	اللغة / بيئه العمل	الاستخدام المثالى
I/O Bound	Node.js (I/O Bound)	تطبيقات الويب التي تعتمد على الشبكة، التعامل مع قواعد البيانات APIs
	Python (مع asyncio g threading) (I/O Bound)	القراءة والكتابة في الملفات بشكل غير متزامن APIs، الأتمتة، التعامل مع
	JavaScript (مع AJAX, Fetch) (I/O Bound)	تطبيقات الويب التي تتفاعل مع الخادم وقواعد البيانات بشكل متزامن
	C# (مع ASP.NET Core) (I/O Bound)	تطبيقات الويب والخدمات التي تعتمد على الوصول لقواعد البيانات والتفاعل مع الواجهات
	Java (مع Spring Boot) (I/O Bound)	بناء تطبيقات الويب المعتمدة على قواعد البيانات وخدمات الشبكة
	Ruby on Rails (I/O Bound)	تطبيقات الويب المعتمدة على التعامل مع بيانات من قواعد بيانات أو APIs
CPU Bound	Go (Golang) (I/O Bound)	I/O خدمات الشبكة وتطبيقات الويب التي تعتمد على عمليات
	C/C++ (CPU Bound)	التطبيقات التي تحتاج إلى معالجة كثيفة للبيانات، الحوسنة العلمية، الألعاب
	Fortran (CPU Bound)	الحوسبة العلمية وحسابات رياضية معقدة (التطبيقات الهندسية والعلمية)
	Python (مع NumPy g TensorFlow) (CPU Bound)	الحوسبة العلمية، التحليل البياني، الذكاء الاصطناعي والتعلم الآلي

النوع	اللغة / بيئة العمل	الاستخدام المثالي
	Java (CPU Bound)	تطبيقات الدوسبة المعقودة، معالجة البيانات، الألعاب الكبيرة
	C# (مع LINQ) (CPU Bound)	معالجة بيانات كثيفة الحسابات وتحليل البيانات المعقودة في بيئات .NET.
	Rust (CPU Bound)	التطبيقات التي تتطلب أداء عالي مثل الأنظمة المدمجة والبرمجيات عالية الأداء
	Julia (CPU Bound)	الحوسبة العلمية والتحليل الرياضي، الذكاء الاصطناعي
أفضل الأدوات لتحسين الأداء	Asynchronous I/O, Threading, Caching, Load Balancing	/ لـ I/O لتقليل الانتظار وتحسين التعامل مع العمليات التي تعتمد على
أفضل الأدوات لتحسين الأداء	Parallel Processing, Multi-threading, GPU Acceleration	تحسين أداء العمليات الحاسوبية الثقيلة باستخدام المعالجة المتوازية والـ GPU

- **Ready queue** contains processes waiting to execute on a CPU.
- **Wait queue** contains processes waiting for a certain event to occur.



وبعدو من الصورة التالية هتفهم المسارات المختلفة الي ممكن تقطعها الـ Process



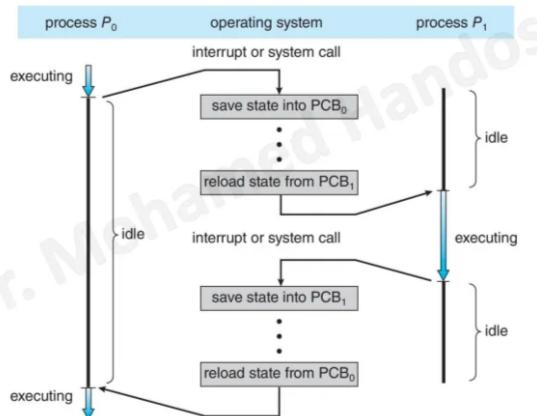
Context Switch

المقصود هنا ان الـ CPU يبدل بين الـ Process الي شغاله عليه ويبدل مكانها اخري من الـ CPU والعملية دي هتتم علي مرحلتين

1. Save the PCB of Current Process
2. Load PCB of Process that scheduler selected from ready queue

طبعا عملية التبديل هتأخذ وقت ودا بنسمية

وفي الوقت دا الـ CPU مبيكنش بيعمل اي حاجة غير التبديل فهنا بنتحتاج نقل الوقت دا للقصي دا فالتبديل دا شغل تنظيمي فمثلا لو عندك process1 بيتاخذ نص ساعه عشان تخلص process2 وبتحتاج نص ساعه فلتلاقي ان الشغل خلص في ساعه وربع طب ليه ؟ عشان وقت التبديل الي بقى حمل عليا بس مضطر اعملو

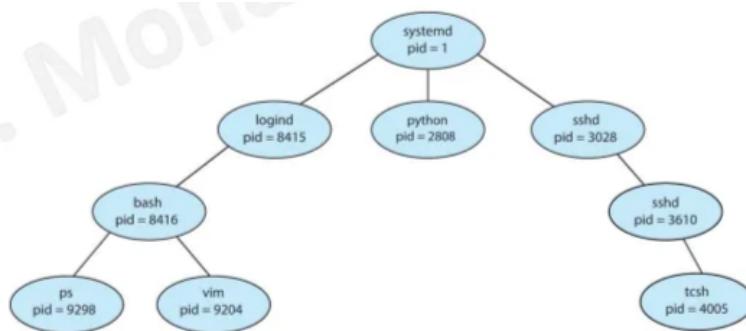


operations on process

Process Creation

Parent Process can create Child Process and child can create child

يُعمَلُ زِي شجرة



طبع اي العلاقات التي تظهر بين الـ processes parent and child

1. About Execution

1.parent and child works Concurrently

يعني الاثنين هشتغون مع بعض وهيتنافسون على الـ CPU

2.Parent wait child to completed

لان ممكن يكون محتاج الناتج بتاع تنفيذ الـ child يقدر الـ parent يكمل شغلو

2. Address-Space

ممكن يكون الـ child و برنامج تاني في التالى متخزن في مكان اخر في الـ memory

ممكن يكون الـ child موجود في نفس مكان الـ parent زي لما تفتح tab على متصفح مثلاً new

3. Resource sharing

- Parent and child share no resources.
- Parent and children share all resources.
- Children share a subset of parent's resources.



Process Termination

Process after end execution closed

ودا عن طريق ان نظام التشغيل يبزفها من الـ memory عن طريق exit system call

ودا الـ Compiler بيضيفو تلقائي بمجرد متنتهي العمليه

وكمان نظام التشغيل يمسح الـ resources كانت تستخدموها

زي فتحة ملفات معينه وغيره

بس لازم الـ resources توقف الاول ثم الـ process

ودا ممكن تكون لاحظتو لو كنت فاتح ملف word وحيث تم سحوه وهو مفتوح هيعرف

Parent Process can Terminate Child Processes

ودا هيحصل بسبب من الـ

1.parent process unwanted a child process again

2.parent is ended then all Childs ended

3.child تجاوز الموارد المتاحة ليه

parent killed it وسعتها الـ

في بعض الانظمة مبتسعدش ان الـ child process تفضل موجودة بعد ما يوقف parent

ودا يسمى Cascading Termination

inter-process communications(IPC)

Relations Between Two Processes

1.Independent Processes : يعني كل واحدة مستقلة عن الاخر وملهمش علاقة ببعض

2.Cooperating Processes : يتعاونو مع بعض ويتبادلو البيانات مع بعض

Processes Cooperating طب ليه اصلا نخلي الـ

1. Sharing Information(Copy & past)

نسرع الشغل بحيث نقدر نستفيد من اكتر من processor

cooperating processes بمعنى لو عندنا large process وقطعتها الى اجزاء ويقي في عندي

processors فسعتها هقدر استفاد من تعدد الـ

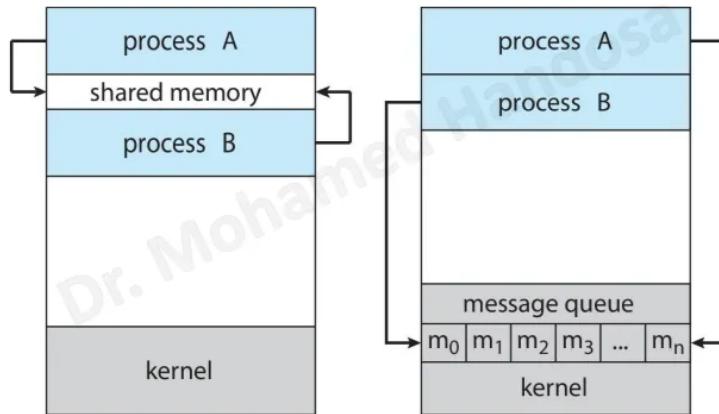
3. Modularity نفس الكلام هقىم الـ process الى اجزاء بس هنا هدفنا سهولة الصيانه فيما بعد

Inter process Communication طب لادم قسمتهم لازم يكونو بي التواصل مع بعض يعني يبقى في IPC

ودا بطريقتين:

1. Shared Memory

2. Passing Messages



Shared Memory

هنا ال Processes عملي جزء مشترك مابنهم يتبادلو فيه البيانات بس نظام التشغيل ملوش دعوة بيهم اي غلط او خلل هما الي يحلوه طب اي المشاكل الي معكן يواجهوها

مينفعش واحدة تكون بتقرأ قبل مالتانية تكون كتبتها الي تقرأو 1.

مينفعش اتنين يكتبوا مع بعض في نفس المكان لأن كدا هبيحصل تداخل للبيانات. 2.

طب لدام في مشاكل كدا في الطريقة دي ليه ممكن نستخدمها؟

لان هي اسرع من ان لسه هجهز رسالة وابعتها



Passing Messages

دي اكيد هتسخدم لو ال shared memory or passing messages

وبعدو لو مشتركين في نفس ال Memory ممكن استخدم الطريقتين operating system allow to ways for passing messages

1. send(message)
2. receive(message)

طب الرسالة الي تتبع دي هيكون حجمها ثابت او متغير

وفي الحالة دي لو بيعت رسالة حجمها اصغر من الحجم المحدد هنضطر نحط الباقى فاضي:

ولو كان حجمها اكبر من الحجم المحدد هنضطر نقسمها لاجزاء عشان نراعي ثبات الحجم
بس هنا العمل زاد على المبرمج

هنا نظام التشغيل قرر يريح المبرمج ويأخذ الرسالة باي حجم ويكسرها هو:

طب ليه يكسرها لان كدا البيانات بتتبع في ال Network بحجم packets

طب عشان ال processes يكلمو بعض لازم يكون بينهم رابط(Link)

Types of Link

1. Physical Link : يعني موصل بينهم كل

لو علي نفس الجهاز هيبيقي H.W Bus

ولو جهاز اخر هيبيقي Network

2. Logical Link: ودا ليه اكتر من نوع

1. Direct and Indirect Communication

2. Synchronous and Asynchronous Communication

3. Automatic or Explicit buffering



Direct Communication

1. Symmetric Addressing

Send(P2,message) : P2 يبعث رسالة لـ

Receive(P1,message) : P1 يُعنى **هستقبل رسالة من**

هنا الاتنين زى بعض وبيقولو اسامي بعض

2. Asymmetric Addressing

Send(P2,message) : P2 أرسل رسالة message.

هنا ال P2P تستقبل من اي حد بس لما تستقبل عوزة تعرف ال ID : Receive(ID,message)

عشان لما تيجي ترد تبقي عرفة هترد على مين

طب ای مواصفات ال

1. automatic بشكل خلقية
 2. Link between two processes only
 3. only one link between two processes



Indirect Communication

هذا هو عنوان البريد الإلكتروني الذي تم إرساله إلى صندوق البريد.

1. mailbox user: الرسالة هي بعثت الى اي

دا الي هيستقبل الرسالة: mailbox owner

هنا بقت العملية ابعت لصندوق البريد وهات من صندوق البريد

Send(mailbox, message)

Receive(mailbox, message)

طب ای هی مواصفات الـLink

1. Link created after creating mailbox and share it

للازم تعامل كود يخلق الـ `mailbox` وتعرفو للـ `processes`

2. process has one mailbox that more processes can send to it

3. Process can have more Link(More Mailbox)



Synchronous and Asynchronous Communication

- ## 1. Blocking (Synchronous)Send

الـprocess تتبع الرسالة ومش هتكلل شغلها غير لما يتم استقبال إسالتها

- ## 2. Nonblocking (Asynchronous) Send

الـ process، لا هتكمـل شـغلـها

- ### 3. Blocking (Synchronous)Receive

هنا له حالو ام يستقبل اسالله ومثلا مكتتش لسه حت هيفضل عدو مستنى لعد متبني

- #### 4. Nonblocking (Asynchronous) Receive

و ها ها ام بستقا، رسالة و مکانتش، ایس جت و های طبعات مارکوپالا، شغله علم، و تجاه.



buffering

1. No Buffering (Zero Capacity) هنا مفهوم مكان تدفق فيه الى عاوز تعتله اديله المسالة في ايدك:

فطبيعي، تكون الـ Sender blocking للـ LAN هو لازم يضمن ان الم رسالة وصلت

- ## 2. Bounded Capacity بعدي، هكذا فـ buffer محدود

هنا في الحالة دي اول ما بتعمي سعتها || buffer must be blocking

- هنا في، مساحة مفتوحة وهنا معاً بنقها، المساحة مالازهارة

sender nonblocking all time , || |
הענעה

Client-Server Communications



1. Sockets

ازی مثلًا لما تعوز تكلم حد لازم تبقى معاك رقم تليفونه

Socket = IP + Port

ای، جهاز متناظر، شبکه لازم باخد IP عشان اکلمه عليه

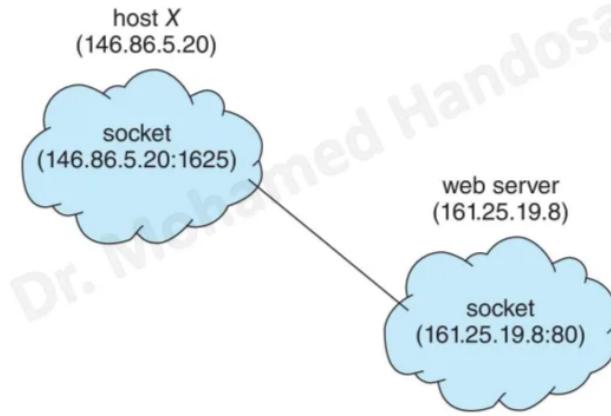
Port: **process**, **اللّه عاصي** اكلمه

فڪدا له عندي اتنين processes بيكلمو بعض لازم يكون عندي 2 sockets

فدا سطر عن طبقه، اون الـ server listening at specific port.

لما بعده request بـ Client soket

للان الـ **request** سيبقى شايل المعلومات اللي متجهها عشان يرد عليك



Remote Procedure Call (RPC)

ودا الي انت بتعملو ديمما بتعمل methods
بس الاختلاف هنا ان ال ه تكون في another process method
يعني اي ؟ يعني هتنادي علي method
كمثال علي دا ال Webservice ودا بيبقى عباره عن دوال علي Server بيندهلها وترد عليا بتبيه
يبيقي دلوقتي هنبقى عندنا Server عليه مجموعة دوال وهبدأ استدعها عند client
طب هنتعامل معها ازاي ؟ دا عن طريق حاجة اسمها stub: client-side
ودا نفس شكل ال لكن بيبقى عند ال methods
بس دي مش ال stub
الحقيقة لان الحقيقة عند ال server لكن هنستخدم ال stub اقدر استخدمهم

تعالي ناخذ مثال نفترض ان عندي دالة بتجلبلي مجموع رقمين عند ال server اسمها Sum
RPC Daemon listening at port of the remote system فحاليا ال
يعني ال request قاعد مستنيك تبعث له
فانت تيجي تجيب الدالة ال virtual من ال Stub وتبعت لها الرقمين
Sum(3,7)
RPC Daemon get request and send a response يبدأ ال
response > 10

Chapter4(Threads)



1. Benefits
2. Multi-Core Programming Challenge
3. Types Of Parallelism
4. Multithreading Models
5. Implicit Threading
6. Thread Cancellation



Threads

simple unit of process

Single sequence of Execution

يعني ال thread هو ابسط ما يمكن عمل ليه معالجة على cpu نطلع من هنا ان

Thread light Wight cooperation by Process

Process can has only one Thread (Main Thread)

دي ال process التقليدية الي هي بتبقى في البرامج البسيطة

وبتقدر تنفذ one task at a time

Process can be Multi-Threaded :do multi tasks at a time يعني قادر على

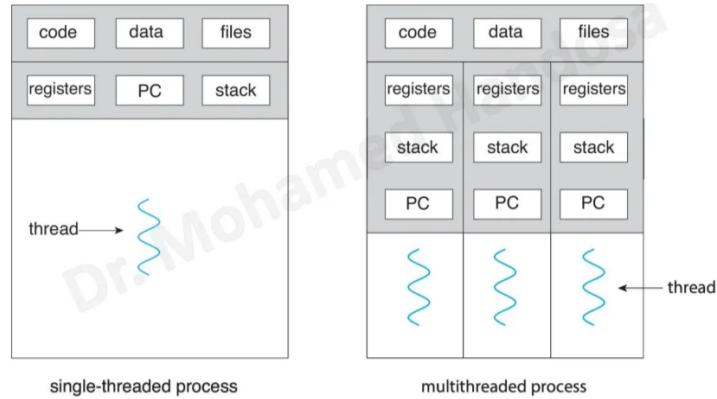
threads works parallel وسعتها التنفيذ هيقيا بالتوالي يعني ال

Thread Contain:

1. ID
2. PC(Program Counter) عشان نعرف مين هيتنفذ بعده
3. Register Set :عشان لما يعمل تبديل يفظل محتفظ بالقيم بتاعو
4. Stack :to store Local Variables

Process Threads Shared :

1. Code Section : الكود كل و بتاع ال process مشتركين فيه
2. data Section :(global variables)
3. Resources: devices or files open



Example Code use single thread:

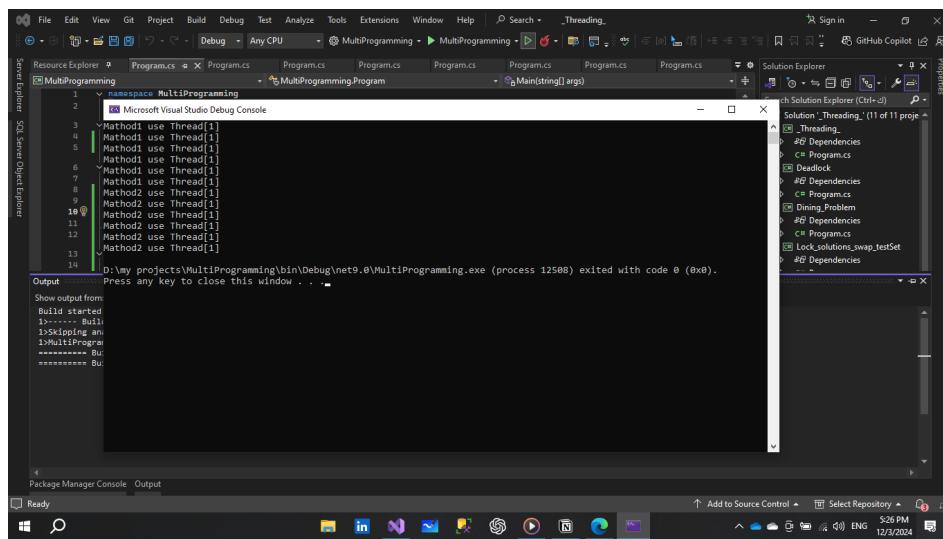
```

namespace MultiProgramming
{
    internal class Program
    {
        //Here Process Works with Single Thread
        static void Main(string[] args)
        {
            Method1();
            Method2();

        }

        static void Method1()
        {
            for (int i = 0; i <=5; i++)
            {
                Console.WriteLine($"Mathod1 use Thread[{Thread.CurrentThread.ManagedThreadId}]")
            }
        }
        static void Method2()
        {
            for (int i = 0; i <= 5; i++)
            {
                Console.WriteLine($"Mathod2 use Thread[{Thread.CurrentThread.ManagedThreadId}]")
            }
        }
    }
}

```



2.Example2 use many threads

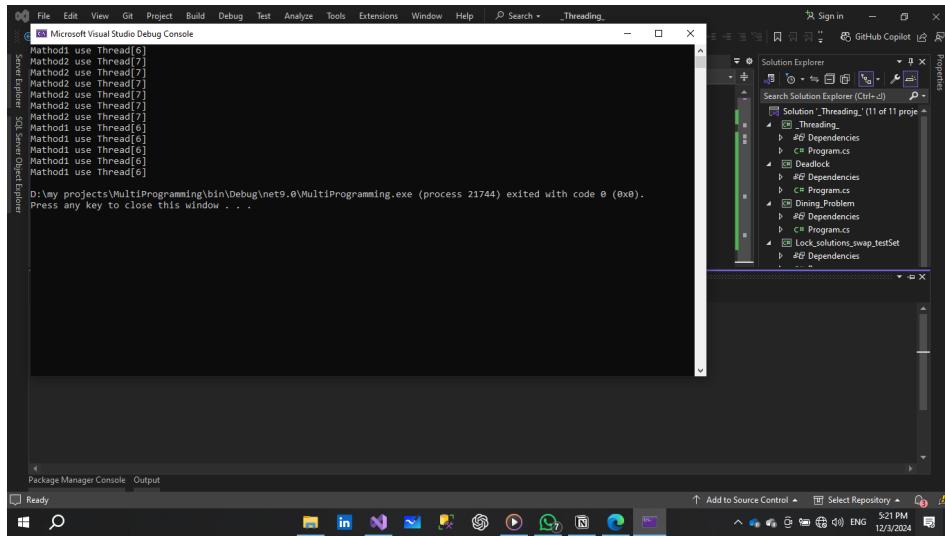
```

namespace MultiProgramming
{
    internal class Program
    {
        //Here Process Works with Single Thread
        static void Main(string[] args)
        {
            Thread T1 = new Thread(new ThreadStart(Method1));
            Thread T2 = new Thread(new ThreadStart(Method2));

            T1.Start();
            T2.Start();
            //Here Two threads works in parallel
        }

        static void Method1()
        {
            for (int i = 0; i <=5; i++)
            {
                Console.WriteLine($"Mathod1 use Thread[{Thread.CurrentThread.ManagedThreadId}]");
            }
        }
        static void Method2()
        {
            for (int i = 0; i <= 5; i++)
            {
                Console.WriteLine($"Mathod2 use Thread[{Thread.CurrentThread.ManagedThreadId}]");
            }
        }
    }
}

```



Benefits of Multithreaded Programs

1. **Responsiveness:** tasks يعنى سرعة الاستجابة ودا بيتم عن طريق توزيع الـ input قبل متلخص تحمل زى مثلا لو فاتح صفحة وكانت بتحمل حاجة معينة فممكنا انك تدخلها اولى thread واستقابال المدخل كان بآخر
2. **Resource Sharing:** memory ودا هييمكنا من استخدام اماكن اقل من الـ memory
3. **Economy:** Thread creation has less overhead than process creation.(Light Weight)
4. **Scalability:** Threads can run in parallel on a multiprocessor machine.



Multi-core Programming Challenges

Identifying tasks : parallel عشان نشغلها tasks يعنى هنقسم البرنامج بنعنا الي

Balance: لازم كل الـ tasks يبيقو متزجين معنی كل وحدة تأخذ وقت تنفيذ يساوي تقريبا غيرها

Data splitting:core هنقسام البيانات بين الـ threads الي هتبقا شغالة على اكتر من

Data dependency: thread معتمد على غيره

لازم وانت بتصمم البرنامج تراعي حتى ان لو في thread يخلص الطلب الي هعوزو منو عشان ابدا منو واكمال

Testing and debugging:

والموضوع هنا بيقدا صعب لانك مش هتبقا متتحكم في ترتيب تنفيذ الـ threads

لانهم شغلين توالي فلما تيجي تتبع الـ error مش هتبقي عارف اي thread اللي سبب المشكلة

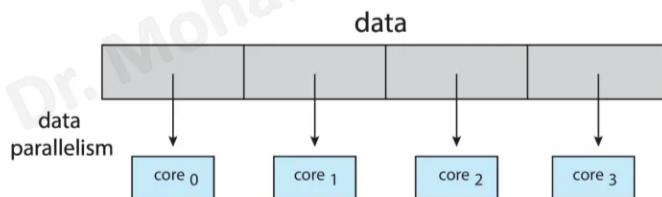
Types of Parallelism



Data Parallelism

تخيل معايا كدا بنعمل برنامج تصحيح الكتروني للامتحانات وكان الامتحان فيه 100 سؤال لو ادنا شغلين بـ (main thread) فهناك هنا هنأخذ وقت اكتر من لو عندنا اكتر من واحد threads طب لو جينا نشتغل multithreading فهناك main thread هيخلق مثلًا 4 threads وكل واحد منهم هيأخذ 25 سؤال يصدهم ولما يخلصوا الـ main thread يجمع منهم النتائج فكدا وفرنا وقت اكتر طب الي حصل هنا اي؟
تقسيم البيانات الي هي في حالتنا الاسئلة بين اكتر من thread هو الـ Data Parallelism وطبعا كل thread run at core

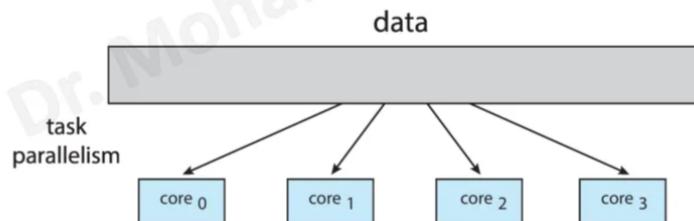
- Distribution of data across multiple cores.



Task Parallelism

نفس المثال بتاع تصحيح الامتحانات بس افترض ان الورق بيتدبر مش معدول
فمحتاج الاول اعدلو ثم اصدرو
هيأخذ وقت اكتر
خلو الموضوع دا تم بـ task parallelism من عاملش واحد يعدل الورقة واحد يتصفح
فسمعتها في البداية هيبيقو اللتين فاضيين هيشتغل واحد يعدل الورقة
ويسلامها للثاني يتصدّها وعلي ميتصفح يكون الاول عدل الورقة الثانية ويدهالو فباتالي الوقت هيقل
ودا الي بنسمييه الـ Task Parallelism

- Distribution of tasks across multiple cores.



Multithreading Models

1. Many-to-one model
2. Many-to-many model
3. One-to-one model
4. Two-level model



User Threads versus Kernel Threads

User Threads: بعملها عن طريق مكتبات بحملها عندي زي مثلاً **Pthread** ومن خلله نقدر:

نشيئ threads بدون تدخل نظم التشغيل

ودا لما يكون نظام التشغيل ميعرفش يعني اي thread

Kernel Thread: يتم انشاءة من خلال نظام التشغيل

Most modern OSs support kernel threads (e.g., Windows, Linux)



Many-to-One Model

ود بيبقى لما يكون نظام التشغيل ميعرفش يعني اي threads

many user threads mapping with one kernel thread

بس بردو هششغل only one use thread in one kernel thread

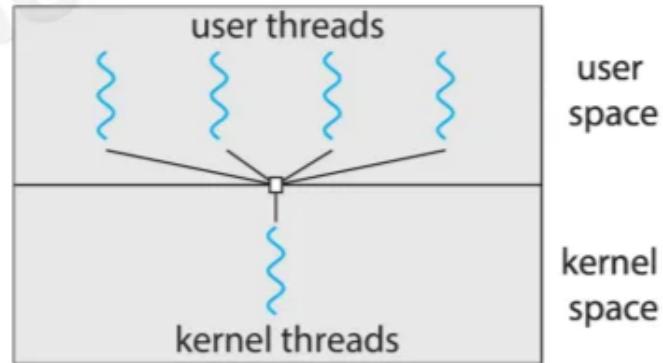
وبالتالي مش هقدر اشغل ال user threads in parallel

في مشكلة ممكن تحصل ان ممكن user thread يسيستولي على ال kernel thread

يعني ممكن يكون عامل blocking حاجه مستعينها فبالتالي ال kernel thread

هييفضل مستنى ومعطل باقى ال user threads

Thread management is done in user space by a thread library not OS



One-to-One Model

وهنا بنتكلم عن نظام التشغيل الي فاهم يعني اي thread كل مهطلب منو هيخلق واحد

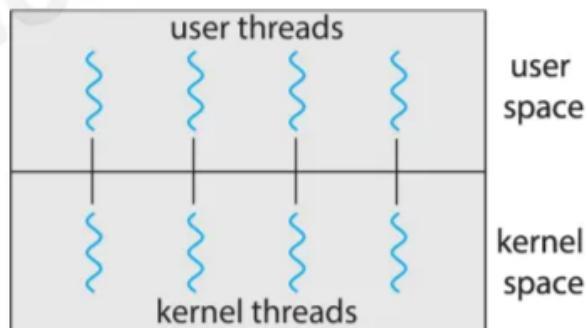
One user thread mapping with one kernel thread

threads in parallel قادر يشغل كل ال

ولما user thread يعمل يعطيه blocking فقط ال kernel thread الي بيشغلو والباقي شغلين عادي

بس هنا في مشكلة كبيرة علي ال Performance هيدمره

ال CPU مش هيبيقا ملائق





Many-to-Many Model

هنا بردو نظام التشغيل فاهم يعني اي thread ينبدأ ويبقدر ينشأ عادي

Many user threads mapping with many kernel threads

بس مش شرط يكون لكل user thread : kernel thread طب هي عمل اي ؟

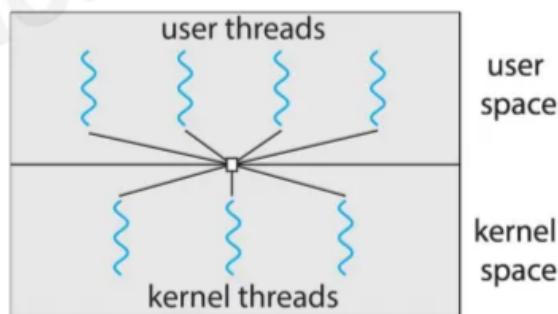
نظام التشغيل هيقول انا اخري اخلق كذا kernel thread لاحافظ على ال performance

OS creates a set of kernel threads (less than or equal to user threads).

ولو عدد ال kernel threads يبدأ ويبلو على عدد ال user threads المماثلين

ولو واحد عمل blocking غيره هو مشتغل

فكدا انا حلية معظم المشاكل



Two-Level Model

دا مزيج بين ال one to one model and Many to many model

الحجات المهمة والضرورية الي لازم تشتبغ حالا هتبقي بنظام ال one to one

والباقي يشتغل many to many

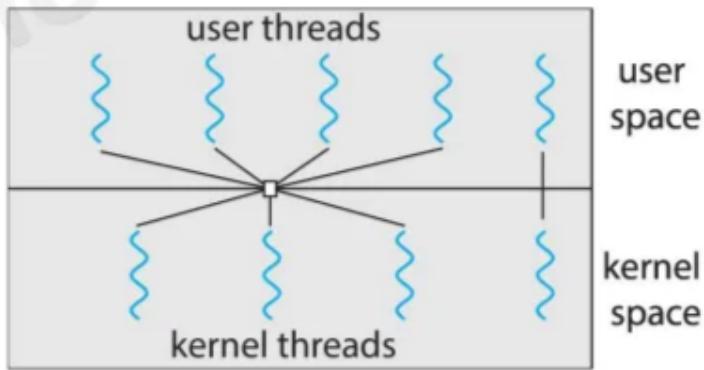
وكمان بالطريقة دي هيبيكي عندنا مجموعة مكتبات هي الي هتنظم عدد ال threads وتحدها

many to many model وبتطبق ال one to one

ودا يفيدك كمبرمج انك تعمل اي عدد من ال user threads زي مانت عاوز وتبقي متطمئن ان

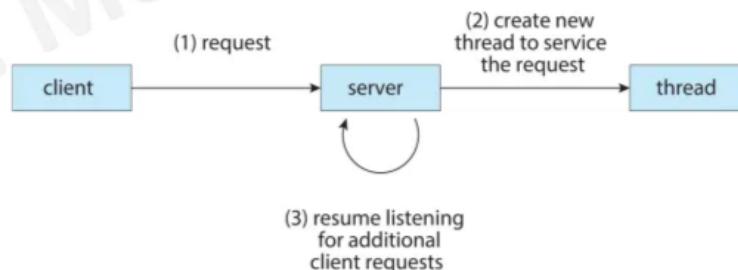
ال kernel threads المقابلة ليها هتبقي محدودة

one to one ونظام التشغيل هيوفر حتى ال



Explicit Threading

نفترض مثلاً ان عندك Server وكل ميديل او request هيفهم عامل one to one model ولما يخلص عمليه يموت و وهكذا يعني هنا شغال server فتلباقي فحصة الـ create threads قادر يرد عليك طلب ليه لأن بقى عامل اكتير اكتر مما يتحمله الـ cpu وكمان عمال تعمل create and remove threads وتخبيط في الوقت



Implicit Threading

هنا هنريح المبرمج خالص بدل ميوجع دماغو ويعمل threads بنفسه الي يتولى الامر هي الـ tasks وهي تشنفهم على الـ Run-time libraries عندها فهقراعي الـ performance وكيبر هيدلوا على الـ threads many to many model هنا احنا اشتغلنا اشغالنا

Two implicit threading approaches:

- Thread pool. works well for asynchronous (independent) tasks.
- Fork-join. works well for tasks that are synchronous (cooperating).



Thread Pool

فكرو ان هيقيا عندي مجموعة من الـ threads جاهزين مش هتضرط تنسأهم لسه وكمان لو حصل request ممكن اعيد استخدام المزايا هنا:

1. التنفيذ اسرع لأن وفرت وقت بدل مكل شوية انشئ واحد لا هستخدمو علي طول
 2. threads هنحسن الاداء لأن هيقي عندي عدد محدد من الـ threads يعني هي لوحدها لو لقت الاداء مش احسن حاجة بتقلل عدد الـ threads ولو كان قادر يستحمل اكتر بتزود عددهم
- A dynamic thread pool can adjust the number of threads in the pool.

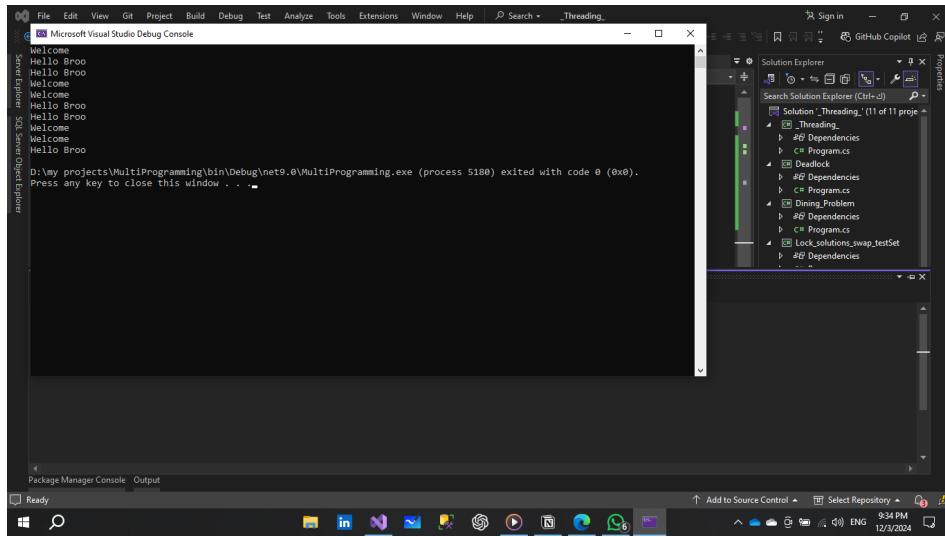
```
namespace MultiProgramming
{
    internal class Program
    {

        static void Main(string[] args)
        {

            ThreadPool.QueueUserWorkItem(print, "Welcome");
            ThreadPool.QueueUserWorkItem(print, "Hello Broo");

            Console.ReadKey();
        }

        static void print(object obj)
        {
            for (int i = 0; i <=4; i++)
            {
                Console.WriteLine(obj);
                Thread.Sleep(500);
            }
        }
    }
}
```



Fork-Join

هنا انا عاوجستني غيرها عشان هتاخذ منها بيانات عشان تقدر تكمل شغلها

ودا هيتم على 3 مراحل :

1. create child thread
2. wait child thread to finish his task
3. Join all child threads and combine all results

ودا مفيد لو عندي كثيرة ومهامها لاجزاي وفي الآخر هجمع النتائج بتعتمد كلها

Example1 Explicate Fork-Join

```
namespace MultiProgramming
{
    internal class Program
    {
        static int Count1;
        static int Count2;
        static void Main(string[] args)
        {
            Count1 = 0;
            Count2 = 0;

            Thread T1 = new Thread(new ThreadStart(print1));
            Thread T2 = new Thread(new ThreadStart(print2));

            T1.Start();
            T2.Start();
        }

        static void print1()
        {
            for (int i = 0; i < 1000000000; i++)
                Count1++;
        }

        static void print2()
        {
            for (int i = 0; i < 1000000000; i++)
                Count2++;
        }
    }
}
```

```

        T1.Join();
        T2.Join();

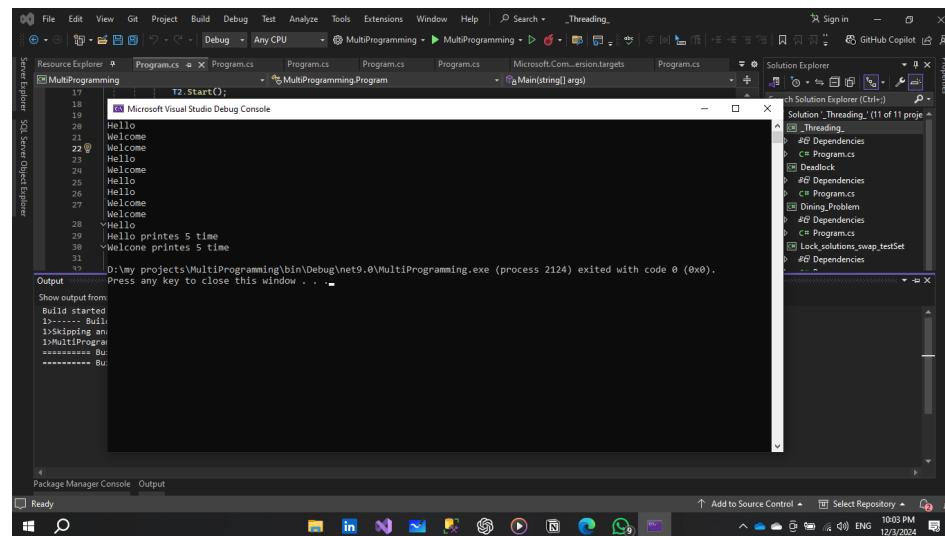
        Console.WriteLine($"Hello printes {Count1} time");
        Console.WriteLine($"Welcone printes {Count2} time");

        Console.ReadKey();
    }

    static void print1()
    {
        for (int i = 0; i <=4; i++)
        {
            Console.WriteLine("Hello");
            Count1++;
            Thread.Sleep(500);
        }
    }

    static void print2()
    {
        for (int i = 0; i <= 4; i++)
        {
            Console.WriteLine("Welcome");
            Count2++;
            Thread.Sleep(500);
        }
    }
}

```



Example2 Implicate Fork-Join

```

namespace MultiProgramming
{
    internal class Program
    {
        static int Count1;
        static int Count2;
        static void Main(string[] args)
        {
            Count1 = 0;
            Count2 = 0;

            Task T1 = new Task((print1));
            Task T2 = new Task((print2));

            T1.Start();
            T2.Start();

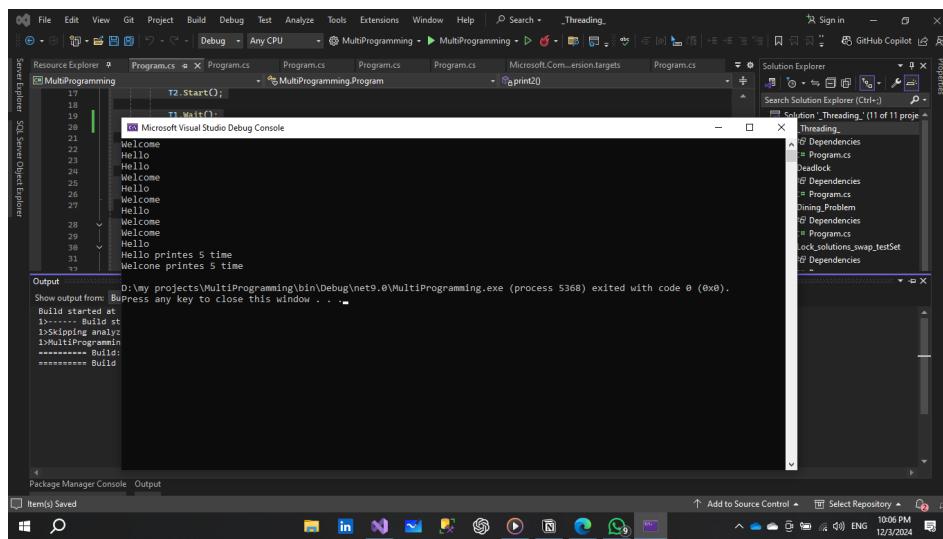
            T1.Wait();
            T2.Wait();

            Console.WriteLine($"Hello printes {Count1} time");
            Console.WriteLine($"Welcone printes {Count2} time");

            Console.ReadKey();
        }

        static void print1()
        {
            for (int i = 0; i <=4; i++)
            {
                Console.WriteLine("Hello");
                Count1++;
                Thread.Sleep(500);
            }
        }
        static void print2()
        {
            for (int i = 0; i <= 4; i++)
            {
                Console.WriteLine("Welcome");
                Count2++;
                Thread.Sleep(500);
            }
        }
    }
}

```



Thread Cancellation

بمعنى أنه قبل ما يخلص شغلو اجباري وفي طريقتين لكدا

1. Asynchronous cancellation

بدون ترتيب أو تنظيم هيبيقي على فجأة

2. Deferred cancellation

هنا هيبيقي الموضوع بشكل منظم عن طريق جعل الـ target thread كل فترة يعمل

هل المفروض يوقف ام لا عشان مييقاش الالغاز مفاجأً وكان مثلاً في نص شغلو فدا يسبب اي مشاكل

Example1 (Asynchronous cancellation)

```
namespace MultiProgramming
{
    internal class Program
    {

        static void Main(string[] args)
        {

            Thread t=new Thread(new ThreadStart(CopyFiles));

            t.Start();
            Thread.Sleep(2000);

            t.Abort(); //Asynchronous cancellation

            Console.ReadKey();
        }
    }
}
```

```

        static void CopyFiles()
    {
        for (int i = 10; i <=100; i+=10)
        {
            Console.WriteLine($"Copying Files. {i}% completed");
            Thread.Sleep(500);
        }
    }

}

```

Example2 (Deferred cancellation)

```

namespace MultiProgramming
{
    internal class Program
    {
        static bool cancle;
        static void Main(string[] args)
        {
            cancle = false;
            Thread t=new Thread(new ThreadStart(CopyFiles));

            t.Start();
            Thread.Sleep(2000);
            cancle = true;

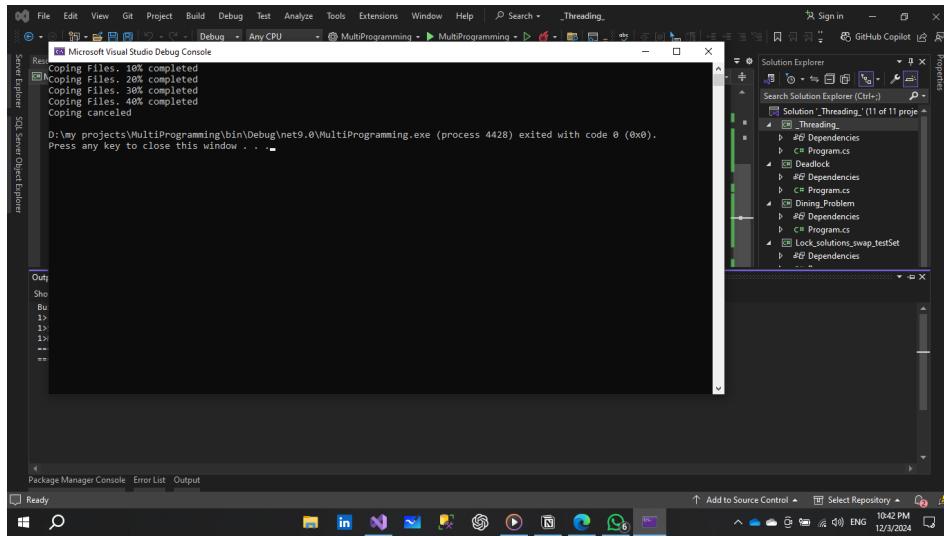
            Console.ReadKey();
        }

        static void CopyFiles()
        {
            for (int i = 10; i <=100; i+=10)
            {
                if (cancle)
                {

                    Console.WriteLine($"Copying canceled ");
                    return;
                }

                Console.WriteLine($"Copying Files. {i}% completed");
                Thread.Sleep(500);
            }
        }
    }
}

```



Chapter5(Process Synchronization)



Processes can be

1. Independent Process: معني ملهاش علاقة بالباقي مستقلة بنفسها
 2. Cooperating Process: ي تكون بينها وبين اخري حجات مشتركة فالتالي يتأثر عليهم processes

Process can Executed in :

1. يعني اشغل العمليات بشكل متزامن بمعنى: اشغل مثلاً 3 عمليات على معالج بحيث كل عملية مثلاً تستغل ثانية وتدخل غيرها وهكذا

في مشكلة اسمها **data inconsistency** ودى يتصل لو شغال concurrently at shared memory

يعني لو كل العمليات اشتراك في نفس مكان تخزين هيكل تضارب

فلازم نحافظ على الـ consistency ونراعي ترتيب التنفيذ ونكون في اكتر من عملية بتدل في البيانات المشتركة

- يعني هبّي عندي أكثر من معالج وهبّي أشغل العمليات عليهم بالتوالي: Parallel**

Process Synchronization:

is a task for coordinating execution of processes to avoid two processes access to shared memory

يُقع، هي عملية التنظيم بين العمليات عشان تتجنب ان مفيش عمليتين يدخلو على السيات المشتركة مع بعض

عندنا مشكلة شهيرة وهي :

Race Condition

بمعنى ان بيقا عندي اكتر من process دخلين سباق علي مين فيهم هيعمل access in shared memory وعشان نفهم اكتر ازاى دا يحصل هنأخذ مثلاً على حاجة شهادة وفديو

zz

Producer Consumer Problem

نفس فكرة و عندك مصنع فهو يعتبر ال producer
و العملاء الي هيسهلكو المنتجات هما ال consumers

producer: generate data and put it in buffer

consumer: consume data from buffer

بس خلي بالك ان المنتج مش هينتج تاني لو المخزن اتملي

ونفس الكلام لو المخزن فاضي المستهلك مش هيلاقني حاجة يستهلكها

يعني مخزن بحجم ثابت (Bounded Buffer)

فعشان نحل مشكلة ان المخزن اتملي ولی لسه فاضي هنعمل التالي:

يعدلی عدد العناصر في المخزن واول ميوصل لانو انتلی نبدأ نوقف الانتاج: Counter

ونفس الكلام لو كان بـ 0 نمنع الاستهلاك

Producer

```
while (true) {
    /* produce an item in next produced */

    while (counter == BUFFER_SIZE) ;
        /* do nothing */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```

Consumer

```
while (true) {
    while (counter == 0)
        ; /* do nothing */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
    /* consume the item in next consumed */
}
```

Race Condition

- `counter++` could be implemented as

```
register1 = counter
register1 = register1 + 1
counter = register1
```
- `counter--` could be implemented as

```
register2 = counter
register2 = register2 - 1
counter = register2
```
- Consider this execution interleaving with "count = 5" initially:

S0: producer execute <code>register1 = counter</code>	{register1 = 5}
S1: producer execute <code>register1 = register1 + 1</code>	{register1 = 6}
S2: consumer execute <code>register2 = counter</code>	{register2 = 5}
S3: consumer execute <code>register2 = register2 - 1</code>	{register2 = 4}
S4: producer execute <code>counter = register1</code>	{counter = 6}
S5: consumer execute <code>counter = register2</code>	{counter = 4}

Critical Section

- General structure of process p_i is

```
do {
    entry section
    critical section
    exit section
    remainder section
} while (true);
```



Critical Section

Critical هو الجزء من الكود الذي يمكن لأكثر من عملية الدخول عليه والتعديل فبالتالي لازم مفيش اتنين يدخلو مع بعض عشان يحصل مشاكل وخطاء غير متوقعة مفعوزين نحمي الـ Section

فلازم قبل المعملية تدخلوا لازم تسأل الاول هل في عملية اخرى جواة ام لا لو مفيش عملية جوة تقدر تدخل ولو في مينفعش تدخل لحد ما لي جوة يخلصه تقسم المشكلة الي

خطوات الحماية كالتالي:

1. **Entry Section :** دا ما قبل الدخول وفيه بندط شرط هل في حد جوة ام لا ونختار عملية عشان تدخل
2. **Critical Section :** دا الي عوزين نحمية
3. **Exit Section :** دا ما بعد الـ Critical Section وفيه بنعرف باقى العمليات ان مفيش حد جوة
4. **Remainder Section:** باقى الكود



وعشان نفكّر في حل لمشكلة الـ Critical Section لازم يتصنّف بال التالي:

1. **Mutual Exclusion**: لو في عملية شغالة على الـ Critical Section مينفعش عملية اخري تدخل عليه
2. **Progress**: لو مفيش عملية شغالة على الـ Critical Section فلازم نختار واحدة تدخل من المنتظرين
3. **Bounded Waiting**: وكمان مينفعش عملية تفضل مستندة الى مالا نهاية يعني يكون في تقدم ومرونة في الاختيار لازم يكون لكل عملية حد معين لعدد المرات التي مسموحة ليها عشان تدخل عشان تفتح المجال لباقي العمليات عشان تأخذ دورها وميكونش في عملية مسؤولية على الـ Critical Section عدا هيدينا ضمان ان مفيش عملية هتسولولي على الـ Critical Section

تعالي بقى ناخد اول حل لـ Critical Section

Example of Producer Consumer Problem

```
using System;

namespace Producer_Consumer_Problem
{
    internal class Program
    {
        //Ahmed Hany Ahmed Elhady Goma
        static void Main(string[] args)
        {
            var wallet = new Wallet("Ahmed", 5000);
            Random test = new Random();

            while (true)
            {
                Thread t1 = new Thread(() => wallet.Debit(test.Next(20, 800)));
                Thread t2 = new Thread(() => wallet.Debit(test.Next(20, 800)));

                t1.Start();
                t2.Start();

                t1.Join();
                t2.Join();

                Console.WriteLine(wallet);
            }

            Console.ReadKey();
        }
    }

    public class Wallet
    {
        public Wallet(string name, int bitcoins)
        {
```

```

        Name = name;
        Bitcoins = bitcoins;
    }

    private readonly object Locker = new object();
    public string Name { get; private set; }
    public int Bitcoins { get; private set; }

    public void Debit(int amount)
    {
        lock (Locker)
        {
            var bit = Bitcoins;
            if (Bitcoins >= amount)
            {
                Thread.Sleep(1000);
                Bitcoins -= amount;
                Console.WriteLine($"Debit {amount} from {bit}");
            }
            else
            {
                Console.WriteLine($"Cannot Debit, you have only {bit} because {amount} > {bit}");
                Credit(amount);
            }
        }
    }

    public void Credit(int amount)
    {
        lock (Locker)
        {
            Thread.Sleep(1000);
            var bit = Bitcoins;
            Bitcoins += amount;
            Console.WriteLine($"{amount} added to {bit}");
        }
    }

    public override string ToString()
    {
        return $" Wallet of [{Name}] => {Bitcoins} Bitcoins]";
    }
}

```

The screenshot shows a Visual Studio IDE with the following details:

- File Explorer:** Shows the project structure: RaceCondition > RaceCondition.
- Solution Explorer:** Shows files: Microsoft.Com_ersion.targets, Program.cs, Program.cs, Program.cs, and Program.cs.
- Task List:** Shows the error: "lock_solutions_swap_testSet".
- Output Window:** Displays the following text:

```
Debit 699 from 818
Show output from: Build started at [Ahmed] > 120 Bitcoins]
Build started at [Ahmed] > Debit 120 you have only 120 because 146 > 120. Adding credit to continue.
1>----- Build st146 added to 120
1>Skipping analyzerBuild 146 from 266
1>----- my projects\>
1>----- my projects\>
1>Done building p
***** Build:
***** Build
```
- Package Manager Console:** Shows the command "Add To Source Control".
- Status Bar:** Shows the date and time: 12/4/2024 4:42 PM.



Peterson's Solution

يعتبر حل تقليدي وقد تم وتمكّن دلوقتي ميشتغلش على الـ H.W Modern بس هيوفرلنا الأساسيات المهمة

وهو برمجي عبارة عن كود لبرنامج Software يعني حل.

وکمان هو محدود بین عملیتین فقط مش اکتر

طب الحل دا عبارۃ عن ای:

طب خدنا الحل البرمجي ناخذ بقى حل W.H

Algorithm for Process P_i, P_j

The structure of a process P

```
do {  
    flag[i] = true;  
    turn = j;  
    while (flag[j] && turn ==  
           j) {  
        critical section  
        flag[i] = false;  
    } remainder section  
} while (true);
```

The structure of a process P :

```

do {
    flag[j] = true;
    turn = i;
    while (flag[i] && turn == i)
        critical section
    flag[j] = false;
    remainder section
} while (true);

```

```

namespace peterson_Algorithm
{

    class PetersonAlgorithmExample
    {
        static int sharedBox = 100;
        static volatile int turn = 0;
        static volatile bool[] flag = new bool[2];

        static void ProcessP0()
        {
            flag[0] = true;
            turn = 1;

            while (flag[1] && turn == 1)
            {
                Thread.Sleep(1000);
            }

            sharedBox += 10;
            Console.WriteLine($"P0 box=: {sharedBox}");

            flag[0] = false;
        }

        static void ProcessP1()
        {
            flag[1] = true;
            turn = 0;

            while (flag[0] && turn == 0)
            {
                Thread.Sleep(1000);
            }

            sharedBox -= 5;
            Console.WriteLine($"P1 box = {sharedBox}");

            flag[1] = false;
        }

        static void Main()
        {
            Thread t0 = new Thread(ProcessP0);
            Thread t1 = new Thread(ProcessP1);

            t0.Start();
            t1.Start();
        }
    }
}

```

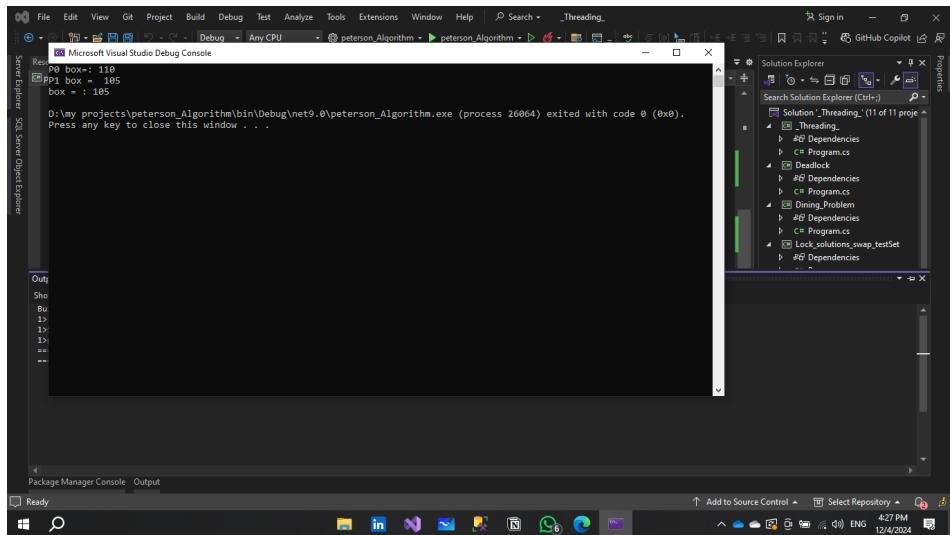
```

        t0.Join();
        t1.Join();

        Console.WriteLine($"box = : {sharedBox}");
    }
}

}

```



Synchronization H.W

H.W Instruction معمظ الانظمة بيعمل فيها جزء H.W يساعدنا على حل المشكلة وهو الـ

Critical Section وكل الحلول مبنية على فكرة الـ locking يعني هتفصل على الـ



If we has Uniprocessor :

فهنا ممكن يكون لحل ان نمنع حدوث الـ Interrupts لأن هي اصلاً اللي سببت المشكلة لأنها لما بتحصل بتضطر الـ CPU انها تطلع العملية الي عملتو وتدخل وحدة تبديل فلو كان عندنا معالج واحد نمنع الـ Interrupts وبالتالي العملية الي دخلت هتفضل تتنفذ لحد متخلص شغلها وتطلع ويدخل غيرها وهكذا بس الحل دا مش هيفي فعال لو عندنا اكتر من معالج



By using H.W Instruction in Modern Machines

معنی ان الكود هنا باللغة المACHINE

يتعقلي مفيشن InterruPT يعني متوقف التنفيذ لازم كلها تنفذ وهي نوعين:

1. Test space in memory and set a value (Test_and_Set_Instruction)
2. or Swap contents of two memory spaces(Compare_and_Swap_Instruction)

فالموضوع بيهمشي كالتالي ان عملية هتدخل تبدأ تقول علي نفسها وهي بس الي معها المفتاح وبعد مهتملاً تفتح القفل عشان تسعح بعملية اخري انها تدخل وبردو هي كمان هتعمل زيها



Test_and_Set_Instruction

فكرتها ببساطة ان هعمل متغير هسمية target من نوع Boolean

if target = false : يبقى مفيشن حد جوة وقافل علي نفسوقدر تدخل:

target = true واول ميدخل يظلي الـ

مش هتقدر تدخل لأن في حد جوة :

Target >> Lock variable

Example of Code C#

```
namespace Lock_solutions_swap_testSet
{
    class Program
    {
        private static volatile bool lockFlag = false;

        static void Main()
        {
            Thread thread1 = new Thread(ExecuteCriticalSection1);
            Thread thread2 = new Thread(ExecuteCriticalSection1);

            thread1.Start();
            thread2.Start();

            Console.ReadKey();
        }

        static void ExecuteCriticalSection1()
        {
            while (true)
            {
                while (TestAndSet(ref lockFlag))
                {
                    Thread.Sleep(1);
                }
            }
        }
    }
}
```

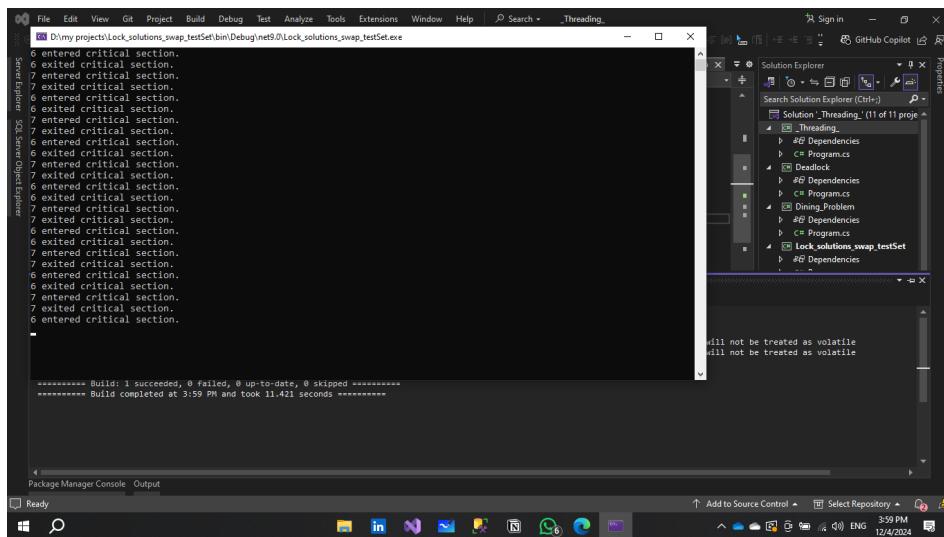
```
Console.WriteLine($"{{Thread.CurrentThread.ManagedThreadId}} entered critical section");
Thread.Sleep(1000);

lockFlag = false;

Console.WriteLine($"{{Thread.CurrentThread.ManagedThreadId}} exited critical section");
Thread.Sleep(1000);
}

}

static bool TestAndSet(ref bool lockVariable)
{
    lock (typeof(Program)) // لضمان أمان العملية
    {
        bool oldValue = lockVariable;
        lockVariable = true;
        return oldValue;
    }
}
}
```





Compare_and_Swap_Instruction

هنا هي بقى عندي 3 متغيرات :

Lock variable

Expected Variable

New Value

الفكرة ان لو كان ال Lock variable == value of expected variable

lock variable = new value

بس الي هيرجع هي قيمة ال Lock variable

يعني ببساطة تيجي عملية عوزة تدخل فلازم تقفل علي نفسها وتأخذ المفتاح معها فالاول

هاد القيمة القديمة لي بتنقول ان القفل مفتوح عشان ادخل الي هي هنا ان ال temp = value

وبعد كدا هنسأل هل انت معك المفتاح عشان تدخل واقفل عليك الي هو value = expected

تعالي نفترض ان القيمة المتوقعة هنا ان المفتاح expected = 0

يعني لو ال value = 0 يعني انت معك المفتاح وهدخل واغير المفتاح الي هو هظلي ال value = new value(1)

فكدا لو عملية اخري جت عوزة تدخل مش هيكون معها المفتاح لانو اتغير ومع العملية الي جوة حالا

وبعد ما الي جوة تخلص هتغير قيمة ال value = 0

وكذا هتقدر تدخل

بس هنا في ال two [H.W Instruction](#) الي قلناهم الموضوع ماشي حظ شوية

لان بعد عملية تفتح القفل الي هتدخل هيدخل بالحظ فمعنكم عملية متعرفش تدخل خاليس

وعشان نحد المشكلة هنخلي كل عملية تدخل عدد محدد من لمرات عشان تفتح المجال لغيرها ان هو يدخل

وكذا لو عندي 5 عمليات عوزين يدخلو العملية الأخيرة هتسندي 4 عمليات

```
namespace Lock_solutions_swap_testSet
{
    class Program
    {
        private static volatile bool lockFlag = false;

        static void Main()
        {
            Thread thread3 = new Thread(ExecuteCriticalSection2);
            Thread thread4 = new Thread(ExecuteCriticalSection2);

            thread3.Start();
            thread4.Start();
        }
    }
}
```

```

        Thread.Sleep(10000);
    }

    static void ExecuteCriticalSection2()
    {
        bool key = true;

        while (true)
        {
            key = true;
            while (key)
            {

                Swap(ref lockFlag, ref key);
            }

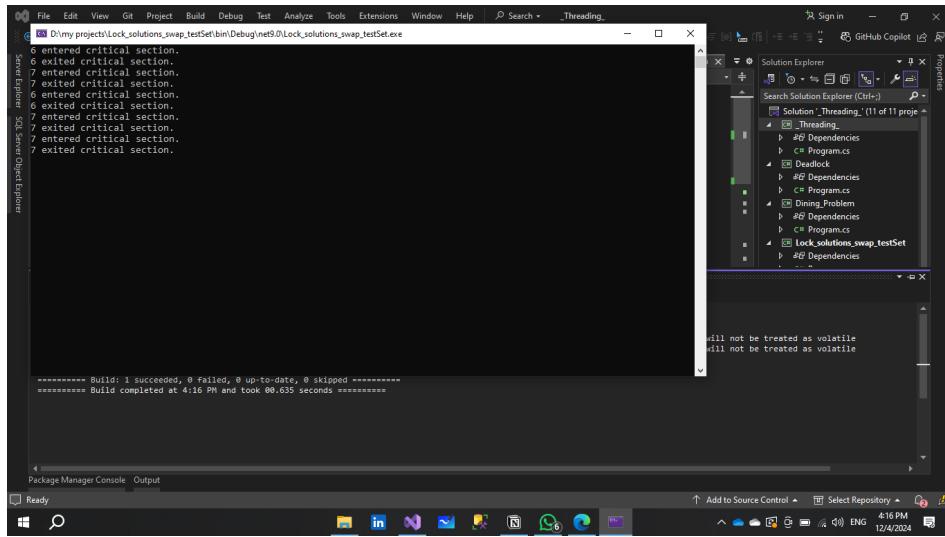
            Console.WriteLine($"{Thread.CurrentThread.ManagedThreadId} entered critical sect
Thread.Sleep(1000);

            Console.WriteLine($"{Thread.CurrentThread.ManagedThreadId} exited critical secti
Thread.Sleep(1000);
            lockFlag = false;

        }
    }

    static void Swap(ref bool lockVariable, ref bool keyVariable)
    {
        bool temp = lockVariable;
        lockVariable = keyVariable;
        keyVariable = temp;
    }
}

```



كدا قلنا حل برمجي وحلين W.H تعالي ناخد حد كمان برمجي وهو



Mutex Locks

نظرا لان الحل البرمجي الاول كان محدود والحلين الـ W.H صعب انك كا مبرمج تعمليم ظاهر الحل التالي ودا عن طريق دالتين

1. acquire() :busy waiting في العمليات باقي

هنا بعد مالعملية الي جوة تخلص هتفتح القفل : release()

ونظرا لان النوع دا بيستخدم الـ busy waiting معها احنا شغلين الـ CPU عالفاضي

لان مبنطلاعش العمليات الي في الانتظار من الـ CPU بيفضل في loop

وعشان كدا بنسمعي النوع دا spin waiting

بس الحل دا مش كوييس لو عندي معالج واحد لان هنضيع وقت عالفاضي وهنحطو في ضغط

بس لو عندنا اكتر من معالج هيككون الحل د احسن بدل منعمل Context Switch

هيبيقي اوفر فالوقت من الحل دا

acquire() and release()

- The availability of a mutex lock is indicated by the value of a boolean variable called available.

```

acquire() {
    while (!available)
        ; /* busy wait */
    available = false;
}

release() {
    available = true;
}

```

```

namespace Mutex_Lock
{
    class Program
    {
        // إنشاء كائن Mutex
        private static Mutex mutex = new Mutex();

        static void Main(string[] args)
        {
            Thread thread1 = new Thread(CriticalSection);
            Thread thread2 = new Thread(CriticalSection);

            thread1.Start();
            thread2.Start();

            thread1.Join();
            thread2.Join();

            Console.ReadKey();
        }

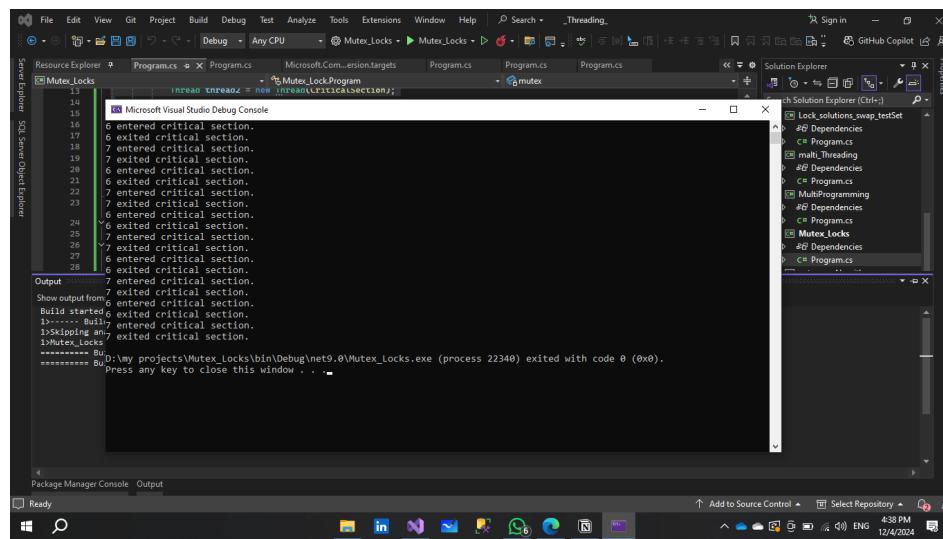
        static void CriticalSection()
        {
            for (int i = 0; i < 5; i++)
            {
                // محاولة الحصول على القفل
                mutex.WaitOne(); //Aquire > put all in busy waiting

                // القسم المربح
                Console.WriteLine($"{Thread.CurrentThread.ManagedThreadId} entered critical sect");
                Thread.Sleep(500);
                Console.WriteLine($"{Thread.CurrentThread.ManagedThreadId} exited critical sect");

                // تحرير القفل
                mutex.ReleaseMutex();

                Thread.Sleep(500);
            }
        }
    }
}

```



Chapter6(Semaphores)

Is a Integer variable

يتغير قيمتو من خلال عمليتين atomic(non interrupt)

`wait()` and `signal()`

```
Wait(S) {  
    while (S ≤ 0);  
    //busy wait  
    S- -;  
}  
signal(S) {  
    S++;  
}
```

Types of Semaphores

1. Counting Semaphores: integer in any range تیکا د تیکا د قیمود لیک
 2. Binary Semaphores : integer value 0 or 1 only the same of mutex lock

ونقدر بردو نستخدم الـ Counting Semaphores لـ التهبيق

Semaphores used for solve various synchronization problems

طب ازای بنشتغل هن؟

٢٩٦ قبل ما تنفذ

A

P1:
S1
Signal(Synch); //1

P2:
Wait(Synch); //1 then waiting
S2



بس هنا في عندنا مشكلة ال busy wait فعشان نستفيد من وقت الانتظار ونشغل المعالج في

حاجة مفيدة فنرجع كدا لحاجة كنا بنعملها وهي ال Context Switch

فبدل منشغل المعالج في الانتظار هنشغلو ودخل عملية تانية تنفذ

عشان نبيقي نرجع العمليات الي عملناها block ومنعناها من الدخول

هندطهم في قائمة

ولما المعالج يفضي معك منشغلا واحدة منههم او نحطها في ال ready queue

يعني كدا اكني بعمل sleep للعمليات الي هتبقى

وهبقي اعملها على المعالج او هحطها في ال ready queue ووهنجلها wakeup

وهنتكلم علي امتيازاتي كدا وامتناعي كدا بعددين في ال CPU Scheduling

هندط الكلام دا ازاى:

هنعمل Queue نحط فيها العمليات الي هتبقى waiting

وهبقي في عمليتين

1. Block :waiting queue في ال وفيها هندط العمليات

2. Wakeup: ready queue من ال ونحطها في ال هنجل العمليات

تعالي بقى نغير الكود بتاع الدلتين الي هما ال wait() and signal()

```
TypeDef struct{
    int value;
    struct process *list;
}semaphore;
```

```

 wait(semaphore *s) {
    s->value--;
    if(s->value < 0) {
        add this process to s->list;
        block();
    }
}
...
 signal(semaphore *s) {
    s->value++;
    if(s->value ≤ 0) {
        remove a process p from s->list;
        wakeup(p);
    }
}

```



أشهر المشاكل التي هي هنا وجهاً وهي

1. Deadlock

بساطة تعالي كدا نفترض ان انت وصبيك بتديلو مسألة في امتحان ومعاكو الة حاسبة وقلم واحد انت معاك الة الحاسبة وهو معاه القلم انت بتحسب ومش عارف تكتب و هو معاه القلم ومش عارف يكتب حاجة احنا الاثنين مستنين بعض لانا بخلاصولي هو بيخلص

يلقى كدا ممكن نعرفو بـ

two or more processes are waiting for event that will fired by another waiting process

no one can finish

2. Starvation

يعني ان في عملية هتفضل الي مالانهاية في الـ

3. Priority Inversion

ودي بتحصل لما يكون في عملية لها اولوية في انها تنفذ اقل من اولوية عملية اخري وقفلة عليها ومش مخليها تنفذ

Classical Problems of Synchronization

Bounded-Buffer Problem(Producer Consumer Problem)

Semaphores 3 هنستخدم

buffer has n of elements ونهفترض ان عندنا

1. Binary Semaphore (Mutex) : critical section عشان يحل مشكلة الـ

2. semaphore full : buffer هيعدد عدد العناصر الي في الـ

3. semaphore empty: buffer هيعد عدد الاماكن الفضية في الـ



Producer

```
do {  
    wait(mutex); //عشان امنع عمليه الـconsume من انها تحصل قبل مخلص  
    wait(empty); // wait يعني هنتأكد ان هل في مكان فاضي ولو مفيش هنسنطي في الـ  
    // Critical section > add item to buffer  
    signal(mutex); //consume هسمح بعملية الـ  
    signal(full); // حجز مكان الي انا ضفت فيه العنصر  
}while(true)
```



Consumer

```
do{  
    wait(full); //هنتأكد ان في عناصر ام لا  
    wait(mutex); //امنع دخول دد وانا جوة شغال  
    //critical section > remove item from buffer  
  
    signal(mutex); //هسمح بدخول غيري  
    signal(empty); //هلغي حجز المكان الي حذفنا منه  
}while(true)
```



Reader Writer problem

العمليات هتقرأ البيانات من ملف مثلاً ومش هتعدل: Reader process

العمليات هتكون قادرة على القراءة والتعديل: Writer Process

فلازم انظم مابنهم عشان ميحصلش data inconsistency

فمهنمصح باكتر من عملية بس تحصل في نفس الوقت

وبردو مش هنمصح بعملية reader تدخل معها

بس هنا هنعوز نعرف الاولوية ه تكون اكتر لـ writer and reader

سواء اديت الاولوية لـ writer or reader

هنجاعاني من مشكلة ال starvation وطبعاً المفترض انت عارف اي هي

تعالي نأخذ حل علي ان هدي الاولوية لـ reader

two binary semaphores and counter هنستخدم

من خلاة اول هيدخل هيقولو علي الـ writers واخر هيفك القفل : 1. rw_mutex :

من خلاه وعدل في المتغير read_counter كل ما عملية reader تيجي جديدة: 2. mutex;



The Structure of Writer Process

```
do{
```

هنا هنخلي عمليات الـ writer تستني لو في عملية writer وبتحصل

وبردو هنخليها تستني لو في عملية reader وبتحصل

Process Writer work.....

```
signal(rw_mutex);
```

```
}while(true);
```

The Structure of Reader Process

```
do{
    wait(mutex);
    read_count++;
    if(read_count==1)
        wait(rw_mutex);
    signal(mutex);
    /*reading work.....*/
    wait(mutex);
    read_count--;
    if(read_count==0)
        signal(rw_mutex);
    signal(mutex);
}while(true);
```

بس لسه هنعاني من مشكلة ال starvation on writer processes

Dining Philosophers Problem

تخيل عندنا 5 فلاسفة قعدين علي طاولة ومبיעملوش حاجة غير انهم يأكلو ويفكررو بس عشان الواحد منهم يعرف يأكل لزم يستخدم شكتين مع بعض بس عالطاولة في 5 شوك بس ازاي بقى نذلهم يكلو ويفكررو ومحدش يتظلم semaphores هنا خمسة هنحتاج

Structure of Philosophers

```
do{
    wait(chopstick[i]);//لما تيجي ياخدها
    wait(chopstick[(i+1)%5]);//ياخد الشوكة الييمين
    Eating.....
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
    thinking.....
}while(true);
```

طب اي المشاكل الي هتزاجة الحل دا

هندخل في deadlock and starvation ودا هيحصل لو كلهم جاعو في نفس الوقت وبالتالي كل واحد هيبيقي معاه شوكة وكل واحد مسنتي الثاني عشان يسيب واحدة ودا مش هيحصل



الحلول كال التالي:

1. Allow only 4 philosophers to be hungry at a time.
2. Allow pickup only if both chopsticks are available.
(Done in critical section)
3. Odd # philosopher always picks up left chopstick 1st,
Even #philosopher always picks up right chopstick 1st.

Allow only 4 philosophers to be hungry at a time.

```
using System;

namespace Dining_Problem
{
    internal class Program
    {
        static void Main(string[] args)
        {
            PhiloFork p = new PhiloFork();
            new Philo(1, 1000, 500, p);
            new Philo(2, 1000, 500, p);
            new Philo(3, 1000, 500, p);
            new Philo(4, 1000, 500, p);
            new Philo(5, 1000, 500, p);
        }
    }

    public class PhiloFork
    {
        bool[] forks = new bool[5];
        private readonly Semaphore semaphore = new Semaphore(1, 1);

        public void GetForks(int left, int right)
        {
            lock (this)
            {
                while (forks[left] || forks[right])
                {
                    Monitor.Wait(this);
                }

                forks[left] = true;
                forks[right] = true;
            }
        }

        public void PutForks(int left, int right)
        {
            lock (this)
```

```

        {
            forks[left] = false;
            forks[right] = false;
            Monitor.PulseAll(this);
        }
    }

    public void WaitToEat()
    {
        semaphore.WaitOne();
    }

    public void DoneEating()
    {
        semaphore.Release();
    }
}

public class Philo
{
    int n, left, right, eatdelay, thinkdelay;
    PhiloFork philofork;

    public Philo(int n, int eatdelay, int thinkdelay, PhiloFork philofork)
    {
        this.n = n;
        this.eatdelay = eatdelay;
        this.thinkdelay = thinkdelay;
        this.philofork = philofork;

        this.left = n == 0 ? 4 : (n - 1) % 5;
        this.right = n % 5;

        new Thread(new ThreadStart(Run)).Start();
    }

    private void Run()
    {
        while (true)
        {
            try
            {

                Thread.Sleep(thinkdelay);
                philofork.WaitToEat();
                philofork.GetForks(left, right);

                Console.WriteLine($"Philo{n} is eating....");

                PrintPhilos(n);
                Thread.Sleep(eatdelay);
            }
        }
    }
}

```

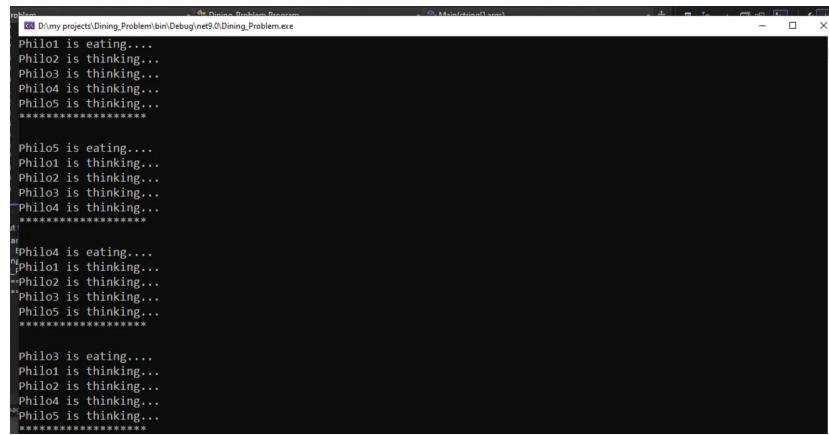
```

        Console.WriteLine("*****");
        Console.ReadLine();
        philofork.PutForks(left, right);
        philofork.DoneEating();

    }
    catch
    {
        return;
    }
}

private void PrintPhilos(int n)
{
    for (int i = 1; i <= 5; i++)
    {
        if (i != n)
        {
            Console.WriteLine($"Philo{i} is thinking...");
        }
    }
}
}

```



**Allow pickup only if both chopsticks are available.
 (Done in critical section)**

```

using System;
using System.Threading;

namespace Dining_Problem
{
    internal class Program
    {
        static void Main(string[] args)
        {
            PhiloFork p = new PhiloFork();
            for (int i = 1; i <= 5; i++)
            {
                new Philo(i, 1000, 500, p);
            }
        }
    }

    public class PhiloFork
    {
        private readonly bool[] forks = new bool[5];

        public void GetForks(int left, int right)
        {
            lock (this)
            {

                while (forks[left] || forks[right])
                {
                    Monitor.Wait(this);
                }

                forks[left] = true;
                forks[right] = true;
            }
        }

        public void PutForks(int left, int right)
        {
            lock (this)
            {

                forks[left] = false;
                forks[right] = false;

                Monitor.PulseAll(this);
            }
        }
    }

    public class Philo
    {

```

```
private readonly int n;
private readonly int left;
private readonly int right;
private readonly int eatDelay;
private readonly int thinkDelay;
private readonly PhiloFork philofork;

public Philo(int n, int eatDelay, int thinkDelay, PhiloFork philofork)
{
    this.n = n;
    this.eatDelay = eatDelay;
    this.thinkDelay = thinkDelay;
    this.philofork = philofork;

    this.left = (n - 1) % 5;
    this.right = n % 5;

    new Thread(Run).Start();
}

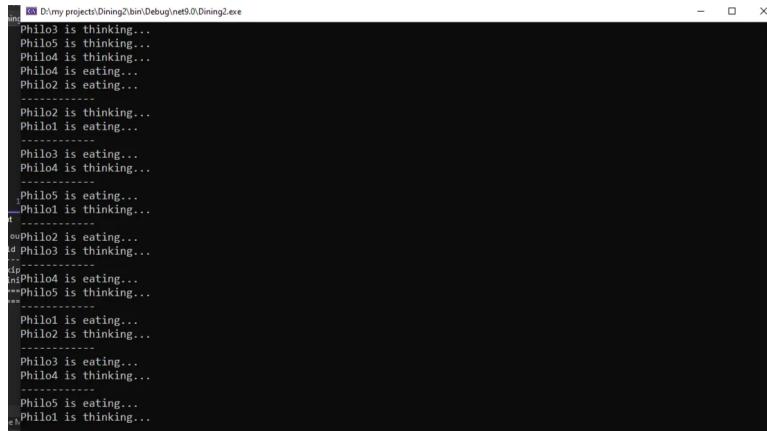
private void Run()
{
    while (true)
    {
        try
        {

            Console.WriteLine($"Philo{n} is thinking...");
            Thread.Sleep(thinkDelay);

            philofork.GetForks(left, right);

            Console.WriteLine($"Philo{n} is eating...");
            Thread.Sleep(eatDelay);

            Console.ReadKey();
            Console.WriteLine("-----");
            philofork.PutForks(left, right);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
            return;
        }
    }
}
```



**Odd # philosopher always picks up left chopstick 1st,
Even #philosopher always picks up right chopstick 1st.**

```

using System;
using System.Threading;

namespace Dining_Problem
{
    internal class Program
    {
        static void Main(string[] args)
        {
            PhiloFork p = new PhiloFork();
            for (int i = 1; i <= 5; i++)
            {
                new Philo(i, 1000, 500, p);
            }
        }
    }

    public class PhiloFork
    {
        private readonly SemaphoreSlim[] forks = new SemaphoreSlim[5];

        public PhiloFork()
        {
            for (int i = 0; i < 5; i++)
            {
                forks[i] = new SemaphoreSlim(1, 1);
            }
        }

        public void GetForks(int left, int right, bool isOdd)
        {
            if (isOdd)
            {

```

```

        forks[left].Wait();
        forks[right].Wait();
    }
    else
    {
        forks[right].Wait();
        forks[left].Wait();
    }
}

public void PutForks(int left, int right)
{
    forks[left].Release();
    forks[right].Release();
}
}

public class Philo
{
    private readonly int n;
    private readonly int left;
    private readonly int right;
    private readonly int eatDelay;
    private readonly int thinkDelay;
    private readonly PhiloFork philofork;
    private readonly bool isOdd;

    public Philo(int n, int eatDelay, int thinkDelay, PhiloFork philofork)
    {
        this.n = n;
        this.eatDelay = eatDelay;
        this.thinkDelay = thinkDelay;
        this.philofork = philofork;

        this.left = (n - 1) % 5;
        this.right = n % 5;

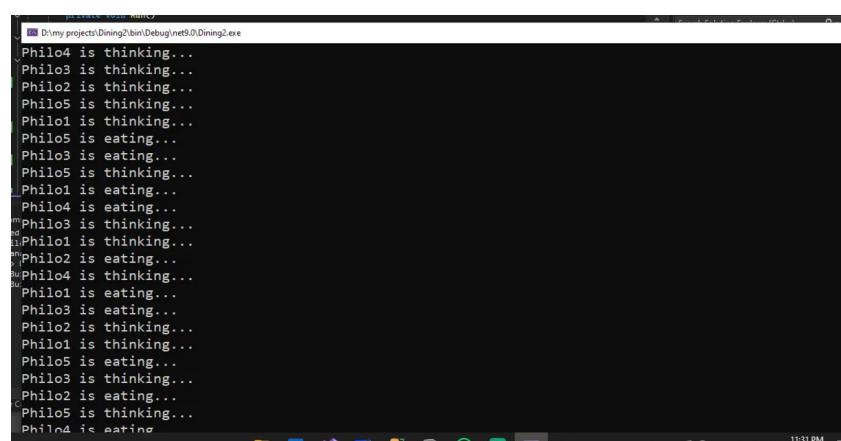
        this.isOdd = n % 2 != 0;

        new Thread(Run).Start();
    }

    private void Run()
    {
        while (true)
        {
            try
            {

```

```
        Console.WriteLine($"Philo{n} is thinking...");  
        Thread.Sleep(thinkDelay);  
  
        philofork.GetForks(left, right, isOdd);  
  
        Console.WriteLine($"Philo{n} is eating...");  
        Thread.Sleep(eatDelay);  
  
        Console.ReadKey();  
  
        philofork.PutForks(left, right);  
    }  
    catch (Exception ex)  
    {  
        Console.WriteLine($"Error: {ex.Message}");  
        return;  
    }  
}  
}  
}
```



Chapter7(CPU Scheduling)

CPU-I/O Burst Cycle



عبارة عن مجموعة اوامر يحصلها معالجة ومجموعة اوامر يتطلب I/O : Process

هي مجموعة الاوامر التي يحدث لها Processing

هي مجموعة الاوامر التي يحدث لها I/O : I/O Burst

Process start with CPU Burst and Finish with CPU Burst

You can't find two burst with same types Neighbors

CPU Scheduler



هو جزء من نظام التشغيل مسؤول عن اختيار عملية من الـ ready queue
طب بيشتغل امتي؟

1. if process terminate then choose another process from ready queue
2. if process wanted an I/O the switching from ready to waiting
3. if interrupt occur switch process from running to ready

ودا زي مثلا وقتها خلص زي ما يحصل في الـ Time Sharing

4. Switch from waiting to ready for example after it done I/O

وهنا العملية سعتها هي بقى ليها اولية اكبر في التنفيذ لأنها هي اللي كانت خارجة بمزاجها

CPU Scheduling Types

Non Preemptive :

هنا شغال بـ 1 و 2 من اللي بيقدر يعملو الـ Scheduler

مبجبش عملية على الخروج هي اللي بتخرج بمزاجها سواء خلصت او طلبت I/O

Preemptive :

هنا شغال على 3 و 4 من اللي بيقدر يعملو الـ Scheduler

يغير العملية اللي شغال على الخروج

ممكن تلاقى Non Preemptive Scheduler و ممكن تلاقى Preemptive Scheduler

بس مينفعش يكون في Preemptive Scheduler شغال

Dispatcher

هو جزء من نظام التشغيل مسؤول عن استلام الاختيار من الـ Scheduler و تنفيذه وذلك كال التالي:

1. Switching Context : ينفذ الاختيار الي اختارو المجدول
2. Switching To user Mode : To apply a protection
3. Program Counter : يخلي العملية تبدأ من مكان ما وقوفها ودا عن طريق ما يخزن في الـ Program Counter

وطبعاً عشان يبدل ويعمل كل دا أكيد هياخد وقت ودا بنسعية Dispatch Latency
ولازم يكون وقت قليل عشان يكون الأداء عالي

Scheduling Criteria



- هنتعرف على بعض الـ Algorithms الي بيتبعها المجدول في اختيار العمليات
وطبعاً عشان حد يعمل Algorithm لازم يتبع المواصفات التالية
1. CPU Utilization لازم منغليش المعالج يطلع نحرص علي ان يكون ديمًا شغال :
 2. Throughput يكون عندنا انتاجية في اقل وقت ممكن:
 3. Low Turnaround بمعنى الوقت الي هستغرقو من اول مالعملية توصل لحد متخرج يكون قليل: Turnaround=Completion Time - arrival Time = Waiting Time + burst time (وقت التنفيذ)
 4. Waiting Time be low = Turnaround time - burst time
 5. Response time be low = start time - arrival time

Scheduling Algorithms



1. First Come - First Served(FCFS)
2. Shortest Job First(SJF)
3. Round Robin(RP)
4. Priority
5. Multilevel Queue
6. Multilevel Feedback Queue

First Come - First Served(FCFS)

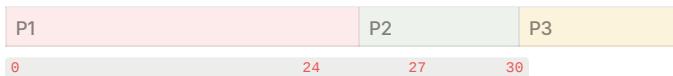
فكرة عمله ببساطة العملية الي هتيجي الاول هي الي هتدخل الاول مليش دعوة بقى هي هتاخذ وقت تنفيذ قد اي مفيش عملية هتيجي تقاطعني انا جاي الاول هدخل الاول

Example

Process	Burst Time
P1	24
P2	3
P3	3

All processes at in same Time

Gantt Chart



Turnaround Time = Completion Time - arrival Time

Waiting Time = Turnaround time - burst time

Response Time = start time - arrival time

Process	Turnaround Time	Waiting Time	Response Time
P1	24-0 = 24	24-24=0	0-0=0
P2	27-0=27	27-3=24	24-0=24
P3	30-0=30	30-3=27	27-0=27
Average	$24 + 27 + 30 / 3 = 27\text{ms}$	$0 + 24 + 27 = 17\text{ms}$	$0 + 24 + 27 = 17\text{ms}$

Implementation Code

```

using System;
using System.Collections.Generic;
using System.Linq;

class Process
{
    public int ProcessId { get; set; }
    public int ArrivalTime { get; set; }
    public int BurstTime { get; set; }
    public int CompletionTime { get; set; }
    public int TurnaroundTime { get; set; }
    public int WaitingTime { get; set; }
}

class FCFS
{
    static void Main(string[] args)
    {
        // قائمة العمليات
        List<Process> processes = new List<Process>
        {
            new Process { ProcessId = 1, ArrivalTime = 0, BurstTime = 4 },
            new Process { ProcessId = 2, ArrivalTime = 2, BurstTime = 3 },
            new Process { ProcessId = 3, ArrivalTime = 4, BurstTime = 1 }
        };

        // حساب جدول العمليات
        ScheduleProcesses(processes);

        // عرض النتائج
    }
}

```

```

Console.WriteLine("Process\tAT\tBT\tCT\tTAT\tWT");
foreach (var process in processes)
{
    Console.WriteLine($"{process.ProcessId}\t{process.ArrivalTime}\t{process.BurstTime}\t");
}

// حساب المتوسط
double avgTAT = processes.Average(p => p.TurnaroundTime);
double avgWT = processes.Average(p => p.WaitingTime);

Console.WriteLine($"Average Turnaround Time: {avgTAT}");
Console.WriteLine($"Average Waiting Time: {avgWT}");
}

static void ScheduleProcesses(List<Process> processes)
{
    // ترتيب العمليات حسب وقت الوصول
    processes = processes.OrderBy(p => p.ArrivalTime).ToList();

    int currentTime = 0;

    foreach (var process in processes)
    {
        // إذا وقت الوصول أكبر من الوقت الحالي، انتظر
        if (currentTime < process.ArrivalTime)
        {
            currentTime = process.ArrivalTime;
        }

        // حساب وقت الانتهاء
        process.CompletionTime = currentTime + process.BurstTime;

        // حساب وقت الدوران = وقت الانتهاء - وقت الوصول
        process.TurnaroundTime = process.CompletionTime - process.ArrivalTime;

        // حساب وقت الانتظار = وقت الدوران - وقت التنفيذ
        process.WaitingTime = process.TurnaroundTime - process.BurstTime;

        // تحديث الوقت الحالي
        currentTime = process.CompletionTime;
    }
}
}

```

Output

	Process	AT	BT	CT	TAT	WT
1		0	4	4	4	0
2		2	3	7	5	2
3		4	1	8	4	3

Average Turnaround Time: 4.333333333333333

Average Waiting Time: 1.6666666666666667

Shorts Job First(SJF)

يقول ان الاقل في المدة يدخل الاول يعني الي مدة تنفيذة اقل هيدخل الاول

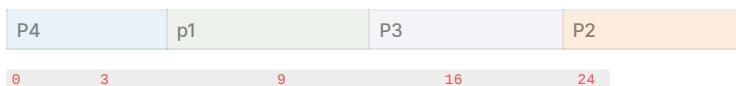
ميهمنيش انه جيت قبلي الي يهعني لو انت هتاخذ زمان اقل عشان تتنفيذ فانت هتدخل الاول

Example:

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

All processes at in same Time

Gantt Chart



Process	Turnaround Time	Waiting Time	Response Time
P1	9 - 0 = 9	9 - 6 = 3	3 - 0 = 3
P2	24 - 0 = 24	24 - 8 = 16	16 - 0 = 16
P3	16 - 0 = 16	16 - 7 = 2	9 - 0 = 9
P4	3 - 0 = 3	3 - 3 = 0	0 - 0 = 0
Average	$9+24+3 / 4 = 13$	$3+16+2+0 / 4 = 7$	$3 + 16 + 9 + 0 / 4 = 7$

ممكن نكون شغلين Non Preemptive or Preemptive

Preemptive (Shorts Job First(SJF))

Example:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt Chart



Process	Turnaround Time	Waiting Time	Response Time
P1	17-0 = 17	17 - 8 = 9	0 - 0 = 0
P2	5 - 1 = 4	4 - 4 = 0	1 - 1 = 0
P3	26 - 2 = 24	24 - 9 = 15	17 - 2 = 15
P4	10 - 3 = 7	7 - 5 = 3	5 - 3 = 2
Average	13	6.5	4.25

ال Algorithm هنا بيحيلي اقل Average waiting time طب ليه مينسخدمهوش ديمما
 لأن فكرتو نظرية بشكل كبير ومش منطقى ان نطبقو عمليا الا لو هنخمن ازاي؟
 لأن ال CPU المفترض يعرف العملية قبل مينفدة هتاخذ وقت قد اى ودا مش هيحصل غير مينفدة
 وكذا ضيعنا وقت عالفااضي طب والحل؟ زي مقالتك هنخمن بس بذكاء شوبتين
 هنطبق المعادلة التالية:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots + (1 - \alpha)^j \alpha t_{n-j} + \dots + (1 - \alpha)^{n+1} \tau_0$$

التبأ التالي $\rightarrow \tau_{n+1}$

هو الزمن الفعلي المرة الي فاتت :

الوقت المتوقع المرة الي فاتت:

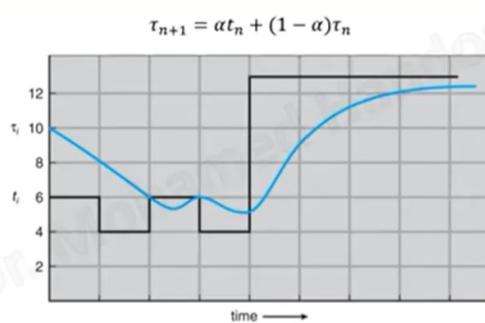
دا ثابت انا بحددو علي حسب عاززين نعتمد اكتر علي التوقع القديم ام الزمن السابق:

وهي محصورة بين 0 و 1

يعني مثلا لو حطناها بـ 1 اكن بقول ان التوقع القادم هو القيمة الحقيقية القديمة

ولو حد حطها بـ 0 اكن بقول ان التوقع القديم هو الجديد

Prediction Example ($\alpha = 0.5$)



CBU Burst 6 4 6 4 13 13 13

Guess 10 8 6 6 5 9 11 12

بنتوقع الاول ثم نشوف النتيجة الفعلية والتي يطلعلي ممكن اعتمد عليه علي حسب الالفا

Code

```

using System;
using System.Collections.Generic;
using System.Linq;

class Process
{
    public int ProcessId { get; set; }
    public int ArrivalTime { get; set; }
    public int BurstTime { get; set; }
    public int RemainingTime { get; set; }
    public int CompletionTime { get; set; }
    public int TurnaroundTime { get; set; }
    public int WaitingTime { get; set; }
}

class PreemptiveSJF
{
    static void Main(string[] args)
    {
        // قائمة العمليات
        List<Process> processes = new List<Process>
        {
            new Process { ProcessId = 1, ArrivalTime = 0, BurstTime = 8 },
            new Process { ProcessId = 2, ArrivalTime = 1, BurstTime = 4 },
            new Process { ProcessId = 3, ArrivalTime = 2, BurstTime = 9 },
            new Process { ProcessId = 4, ArrivalTime = 3, BurstTime = 5 }
        };

        // حساب جدولة العمليات
        ScheduleProcesses(processes);

        // عرض النتائج
        Console.WriteLine("Process\tTAT\ttBT\ttCT\ttTAT\ttWT");
        foreach (var process in processes)
        {
            Console.WriteLine($"{process.ProcessId}\t{process.TurnaroundTime}\t{process.BurstTime}\t");
        }

        // حساب المتوسط
        double avgTAT = processes.Average(p => p.TurnaroundTime);
        double avgWT = processes.Average(p => p.WaitingTime);

        Console.WriteLine($"Average Turnaround Time: {avgTAT}");
        Console.WriteLine($"Average Waiting Time: {avgWT}");
    }

    static void ScheduleProcesses(List<Process> processes)
    {
        int currentTime = 0;
        int completed = 0;
        int n = processes.Count;

        // إعداد الوقت المتبقى لكل عملية
    }
}

```

```

foreach (var process in processes)
{
    process.RemainingTime = process.BurstTime;
}

while (completed < n)
{
    البحث عن العملية ذات الزمن المتبقى الأقصر والتي وصلت // 
    Process shortestProcess = processes
        .Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0)
        .OrderBy(p => p.RemainingTime)
        .FirstOrDefault();

    if (shortestProcess == null)
    {
        currentTime++;
        continue;
    }

    تنفيذ العملية لمدة وحدة زمن واحدة // 
    shortestProcess.RemainingTime--;
    currentTime++;

    إذا انتهت العملية // 
    if (shortestProcess.RemainingTime == 0)
    {
        completed++;
        shortestProcess.CompletionTime = currentTime;
        shortestProcess.TurnaroundTime = shortestProcess.CompletionTime - shortestProcess.ArrivalTime;
        shortestProcess.WaitingTime = shortestProcess.TurnaroundTime - shortestProcess.BurstTime;
    }
}
}

```

OutPut

	Process	AT	BT	CT	TAT	WT
1	1	0	8	17	17	9
2	2	1	4	5	4	0
3	3	2	9	26	24	15
4	4	3	5	12	9	4

Average Turnaround Time: 13.5

Average Waiting Time: 7

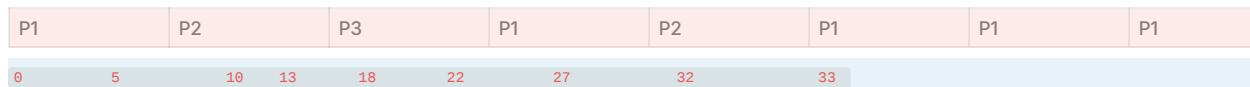
Round Robin(RP)

ودا معظم انظمة التشغيل تعتمد عليه لان متوسط وقت الاستجابة اقل ما يمكن ودا الي عوزيني
 Time Sharing Systems
 لان هو بيشتغل بطريقة توزيع العمليات ويشغل كل وحدة شوية زي فكرة الـ
Preeemptive وبردو لازم يكون شغال

Example:

Process	Burst Time
P1	21
P2	9
P3	3

Quantum=5



Process	Turnaround Time	Waiting Time	Response Time
P1	33-0=33	33-21=12	0-0=0
P2	22-0=22	22-9=13	5-0=5
P3	13-0=13	13-3=10	10-0=10
Average	21	11.66	5



This Algorithm give us a low Average Response Time

كل مهتزود الـFCFS هتقرب اكتر اكلا شغال Quantum

وكل متقلل الـ Dispatch latency كل مال زيد لان الـ context switching

كدا هيبقى الجهاز شغال بس عالفاضي في وقت التبديل الي بيضيع دا

Code

```

using System;
using System.Collections.Generic;
using System.Linq;

class Process
{
    public int ProcessId { get; set; }
    public int ArrivalTime { get; set; }
    public int BurstTime { get; set; }
    public int RemainingTime { get; set; }
    public int CompletionTime { get; set; }
    public int TurnaroundTime { get; set; }
    public int WaitingTime { get; set; }
}

class RoundRobin
{
    static void Main(string[] args)
    {

```

```

// قائمة العمليات
List<Process> processes = new List<Process>
{
    new Process { ProcessId = 1, ArrivalTime = 0, BurstTime = 8 },
    new Process { ProcessId = 2, ArrivalTime = 1, BurstTime = 4 },
    new Process { ProcessId = 3, ArrivalTime = 2, BurstTime = 9 },
    new Process { ProcessId = 4, ArrivalTime = 3, BurstTime = 5 }
};

int timeQuantum = 3; // شريحة الزمن

// حساب جدول العمليات
ScheduleProcesses(processes, timeQuantum);

// عرض النتائج
Console.WriteLine("Process\tTAT\tBT\tCT\tTAT\tWT");
foreach (var process in processes)
{
    Console.WriteLine($"{process.ProcessId}\t{process.TurnaroundTime}\t{process.BurstTime}\t{process.CompletionTime}\t{TAT}\t{WaitingTime}");
}

// حساب المتوسط
double avgTAT = processes.Average(p => p.TurnaroundTime);
double avgWT = processes.Average(p => p.WaitingTime);

Console.WriteLine($"Average Turnaround Time: {avgTAT}");
Console.WriteLine($"Average Waiting Time: {avgWT}");
}

static void ScheduleProcesses(List<Process> processes, int timeQuantum)
{
    int currentTime = 0;
    Queue<Process> readyQueue = new Queue<Process>();

    // إعداد الوقت المتبقي لكل عملية
    foreach (var process in processes)
    {
        process.RemainingTime = process.BurstTime;
    }

    while (processes.Any(p => p.RemainingTime > 0))
    {
        // إضافة العمليات الجاهزة إلى قائمة الانتظار
        foreach (var process in processes.Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0))
        {
            readyQueue.Enqueue(process);
        }

        if (readyQueue.Count == 0)
        {
            currentTime++;
            continue;
        }

        Process process = readyQueue.Dequeue();
        currentTime += timeQuantum;
        process.TurnaroundTime = currentTime;
        process.CompletionTime = process.TurnaroundTime + process.BurstTime;
        process.WaitingTime = process.TurnaroundTime - process.CompletionTime;
        process.RemainingTime -= timeQuantum;
    }
}

```

```

// إخراج العملية الأولى من قائمة الانتظار
var currentProcess = readyQueue.Dequeue();

// تنفيذ العملية لمدة شريحة زمنية
int executionTime = Math.Min(timeQuantum, currentProcess.RemainingTime);
currentProcess.RemainingTime -= executionTime;
currentTime += executionTime;

// إذا انتهت العملية
if (currentProcess.RemainingTime == 0)
{
    currentProcess.CompletionTime = currentTime;
    currentProcess.TurnaroundTime = currentProcess.CompletionTime - currentProcess.ArrivalTime;
    currentProcess.WaitingTime = currentProcess.TurnaroundTime - currentProcess.BurstTime;
}

// إعادة العملية إلى قائمة الانتظار إذا لم تنتهي
foreach (var process in processes.Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0))
{
    readyQueue.Enqueue(process);
}

if (currentProcess.RemainingTime > 0)
{
    readyQueue.Enqueue(currentProcess);
}
}
}
}
}

```

OUTPUT

Process AT BT CT TAT WT

1	0	8	23	23	15
2	1	4	10	9	5
3	2	9	26	24	15
4	3	5	17	14	9

Average Turnaround Time: 17.5

Average Waiting Time: 11

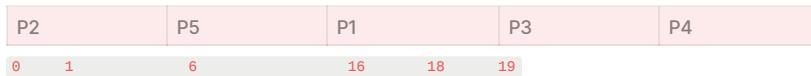
Priority Scheduling

هي الاولوية التي ينديها للعمليات عن طريق رقم وكل مال رقم كان قليل كل ما كانت اولوية العملية اكبر

Example:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4

Process	Burst Time	Priority
P4	1	5
P5	5	2



Process	Turnaround Time	Waiting Time	Response Time
P1	16	6	6
P2	1	0	0
P3	18	16	16
P4	19	18	18
P5	6	1	1
Average	12	8.2	8.2

نفس الكلام ممكن نشتبه هنا

الموضوع دا ممكن يتسبب في مشاكل زي

Starvation and Aging



يعني يحصل مجاعة طب ازاي؟

لان لو عملية اولويتها اقل من الموجود وكل شوية تجي عملية اولويتها اكبر منها وتدخل فهي مش هتعرف تدخل والحل بتعها هو الـ Aging يعني اي؟

يعني العملية دي كل مدة معينة نزود اولويتها يعني نقل الرقم الي وخداء يعني لو الـ Priority=5 يعني بعد شوية مثل 3 نذلها

CODE

```
using System;
using System.Collections.Generic;
using System.Linq;

class Process
{
    public int ProcessId { get; set; }
    public int ArrivalTime { get; set; }
    public int BurstTime { get; set; }
    public int Priority { get; set; }
    public int CompletionTime { get; set; }
    public int TurnaroundTime { get; set; }
    public int WaitingTime { get; set; }
}

class NonPreemptivePriorityScheduling
{
```

```

static void Main(string[] args)
{
    // قائمة العمليات
    List<Process> processes = new List<Process>
    {
        new Process { ProcessId = 1, ArrivalTime = 0, BurstTime = 8, Priority = 3 },
        new Process { ProcessId = 2, ArrivalTime = 1, BurstTime = 4, Priority = 1 },
        new Process { ProcessId = 3, ArrivalTime = 2, BurstTime = 9, Priority = 4 },
        new Process { ProcessId = 4, ArrivalTime = 3, BurstTime = 5, Priority = 2 }
    };

    // حساب جدول العمليات
    ScheduleProcesses(processes);

    // عرض النتائج
    Console.WriteLine("Process\tTAT\tBT\tPriority\tCT\tTAT\tWT");
    foreach (var process in processes)
    {
        Console.WriteLine($"{process.ProcessId}\t{process.ArrivalTime}\t{process.BurstTime}\t");
    }

    // حساب المتوسط
    double avgTAT = processes.Average(p => p.TurnaroundTime);
    double avgWT = processes.Average(p => p.WaitingTime);

    Console.WriteLine($"Average Turnaround Time: {avgTAT}");
    Console.WriteLine($"Average Waiting Time: {avgWT}");
}

static void ScheduleProcesses(List<Process> processes)
{
    int currentTime = 0;

    // الترتيب حسب الأولوية و وقت الوصول
    var sortedProcesses = new List<Process>();

    while (processes.Any(p => p.CompletionTime == 0))
    {
        var readyQueue = processes
            .Where(p => p.ArrivalTime <= currentTime && p.CompletionTime == 0)
            .OrderBy(p => p.Priority)
            .ThenBy(p => p.ArrivalTime);

        if (readyQueue.Any())
        {
            var currentProcess = readyQueue.First();
            currentTime += currentProcess.BurstTime;
            currentProcess.CompletionTime = currentTime;
            currentProcess.TurnaroundTime = currentProcess.CompletionTime - currentProcess.ArrivalTime;
            currentProcess.WaitingTime = currentProcess.TurnaroundTime - currentProcess.ArrivalTime;
            sortedProcesses.Add(currentProcess);
        }
        else
        {
    
```

```
        currentTime++;  
    }  
}
```

OUTPUT

Process	AT	BT	Priority	CT	TAT	WT
2	1	4	1	5	4	0
4	3	5	2	10	7	2
1	0	8	3	18	18	10
3	2	9	4	27	25	16

Average Turnaround Time: 13.5

Average Waiting Time: 7

Multilevel Queue Scheduling



هنا نعمل خطوة من الـ Algorithms السابقة

عن طريق هنطاخ الـ ready queue وشغل كل واحدة بـ different Algorithms

ونخلى فى Algorithm ما بنهم من حيث الاولوية فى التنفيذ ودا ليه نوعين

1. Fixed Priority Scheduling

معنی ان هيكون في اولوية لمثلا اول ready queue بعدها غير لما تكون الاولى فاضية يعني مكنش فيها عمليات ايق عليه ال Algorithm الى هى بتطبق

2. Time Slice Scheduling

عن طريقة هنستغل في كل ready queue وقت محدد ثم ننتقل لغيرها

CODE

```
using System;
using System.Collections.Generic;
using System.Linq;

class Process
{
    public int ProcessId { get; set; }
    public int ArrivalTime { get; set; }
    public int BurstTime { get; set; }
    public int Priority { get; set; } // تعيين المجموعة
    public int CompletionTime { get; set; }
    public int TurnaroundTime { get; set; }
    public int WaitingTime { get; set; }
}

class MultilevelQueueScheduling
{
    static void Main(string[] args)
```

```

{
    // قائمة العمليات
    List<Process> processes = new List<Process>
    {
        new Process { ProcessId = 1, ArrivalTime = 0, BurstTime = 8, Priority = 1 }, // 1
        new Process { ProcessId = 2, ArrivalTime = 1, BurstTime = 4, Priority = 2 }, // 2
        new Process { ProcessId = 3, ArrivalTime = 2, BurstTime = 9, Priority = 1 }, // 1
        new Process { ProcessId = 4, ArrivalTime = 3, BurstTime = 5, Priority = 2 } // 2
    };

    int timeQuantum = 3; // 1 شريحة الزمن للمجموعه

    // تقسيم العمليات إلى مجموعات
    var queue1 = processes.Where(p => p.Priority == 1).ToList(); // العمليات التفاعلية
    var queue2 = processes.Where(p => p.Priority == 2).ToList(); // العمليات الدفعية

    // جدولة العمليات في كل مجموعة
    ScheduleRoundRobin(queue1, timeQuantum);
    ScheduleFCFS(queue2);

    // عرض النتائج
    Console.WriteLine("Process\tTAT\tBT\tPriority\tCT\tTAT\tWT");
    foreach (var process in processes.OrderBy(p => p.CompletionTime))
    {
        Console.WriteLine($"{process.ProcessId}\t{process.ArrivalTime}\t{process.BurstTime}\t");
    }

    // حساب المتوسط
    double avgTAT = processes.Average(p => p.TurnaroundTime);
    double avgWT = processes.Average(p => p.WaitingTime);

    Console.WriteLine($"Average Turnaround Time: {avgTAT}");
    Console.WriteLine($"Average Waiting Time: {avgWT}");
}

// جدولة العمليات باستخدام Round Robin
static void ScheduleRoundRobin(List<Process> processes, int timeQuantum)
{
    int currentTime = 0;
    Queue<Process> readyQueue = new Queue<Process>();

    foreach (var process in processes)
    {
        process.CompletionTime = 0; // Reset to allow processing
    }

    while (processes.Any(p => p.BurstTime > 0))
    {
        foreach (var process in processes.Where(p => p.ArrivalTime <= currentTime && p.BurstTime > 0))
        {
            readyQueue.Enqueue(process);
        }

        if (readyQueue.Count == 0)

```

```

    {
        currentTime++;
        continue;
    }

    var currentProcess = readyQueue.Dequeue();
    int executionTime = Math.Min(timeQuantum, currentProcess.BurstTime);
    currentTime += executionTime;
    currentProcess.BurstTime -= executionTime;

    if (currentProcess.BurstTime == 0)
    {
        currentProcess.CompletionTime = currentTime;
        currentProcess.TurnaroundTime = currentProcess.CompletionTime - currentProcess.ArrivalTime;
        currentProcess.WaitingTime = currentProcess.TurnaroundTime - currentProcess.BurstTime;
    }
    else
    {
        readyQueue.Enqueue(currentProcess);
    }
}

}

// جدول العمليات باستخدام FCFS
static void ScheduleFCFS(List<Process> processes)
{
    int currentTime = 0;

    foreach (var process in processes.OrderBy(p => p.ArrivalTime))
    {
        if (currentTime < process.ArrivalTime)
        {
            currentTime = process.ArrivalTime;
        }

        currentTime += process.BurstTime;
        process.CompletionTime = currentTime;
        process.TurnaroundTime = process.CompletionTime - process.ArrivalTime;
        process.WaitingTime = process.TurnaroundTime - process.BurstTime;
    }
}
}

```

OUTPUT

Process	AT	BT	Priority	CT	TAT	WT
2	1	4	2	7	6	2
4	3	5	2	12	9	4
1	0	8	1	17	17	9
3	2	9	1	26	24	15

Average Turnaround Time: 14

Average Waiting Time: 7.5

Multilevel Feedback Queue



نفس اليMultilevel Queue Scheduling ولكن مع وجود امكانية ان تنتقل عملية من

Queue to another

ودا هي حللي بدو مشكلة الـ Starvation لو حصل في Queue لـ Another Queue سعتها العمليات تقدر تنتقل لـ

طب هنعمل كام Ready queue واي الالجوريزم المتبعة في كل منهم

وعلي اي اساس هتنقل عملية من queue to another

ودا الي لازم تبقي عرفو لو هتعمل الجوريزم من النوع دا

CODE

```
using System;
using System.Collections.Generic;
using System.Linq;

class Process
{
    public int ProcessId { get; set; }
    public int ArrivalTime { get; set; }
    public int BurstTime { get; set; }
    public int RemainingTime { get; set; }
    public int CompletionTime { get; set; }
    public int TurnaroundTime { get; set; }
    public int WaitingTime { get; set; }
}

class MultilevelFeedbackQueueScheduling
{
    static void Main(string[] args)
    {
        // قائمة العمليات
        List<Process> processes = new List<Process>
        {
            new Process { ProcessId = 1, ArrivalTime = 0, BurstTime = 8 },
            new Process { ProcessId = 2, ArrivalTime = 1, BurstTime = 4 },
            new Process { ProcessId = 3, ArrivalTime = 2, BurstTime = 9 },
            new Process { ProcessId = 4, ArrivalTime = 3, BurstTime = 5 }
        };

        foreach (var process in processes)
        {
            process.RemainingTime = process.BurstTime; // لاتبع الوقت المتبقى لكل عملية
        }

        //تعريف المجموعات وشريحة الزمن
        int timeQuantum1 = 3; // شريحة الزمن للمجموعة الأولى
        int timeQuantum2 = 6; // شريحة الزمن للمجموعة الثانية
        List<Process> queue1 = new List<Process>();
```

```

List<Process> queue2 = new List<Process>();
List<Process> queue3 = new List<Process>();

int currentTime = 0;

while (processes.Any(p => p.RemainingTime > 0))
{
    // تعيينة المجموعة الأولى بالعمليات الجاهزة
    foreach (var process in processes.Where(p => p.ArrivalTime <= currentTime && p.RemainingTime > 0))
    {
        queue1.Add(process);
    }

    // معالجة العمليات في المجموعة الأولى (Round Robin)
    ProcessQueue(queue1, timeQuantum1, ref currentTime, queue2);

    // شريحة زمن أطول من أطول (Round Robin) معالجة العمليات في المجموعة الثانية
    ProcessQueue(queue2, timeQuantum2, ref currentTime, queue3);

    // معالجة العمليات في المجموعة الثالثة (FCFS)
    ProcessQueue(queue3, int.MaxValue, ref currentTime, null);
}

// حساب النتائج وعرضها
Console.WriteLine("Process\tTAT\tBT\tCT\tTAT\tWT");
foreach (var process in processes)
{
    process.TurnaroundTime = process.CompletionTime - process.ArrivalTime;
    process.WaitingTime = process.TurnaroundTime - process.BurstTime;
    Console.WriteLine($"{process.ProcessId}\t{process.ArrivalTime}\t{process.BurstTime}\t{process.TurnaroundTime}\t{process.CompletionTime}\t{process.WaitingTime}");
}

double avgTAT = processes.Average(p => p.TurnaroundTime);
double avgWT = processes.Average(p => p.WaitingTime);

Console.WriteLine($"Average Turnaround Time: {avgTAT}");
Console.WriteLine($"Average Waiting Time: {avgWT}");
}

static void ProcessQueue(List<Process> queue, int timeQuantum, ref int currentTime, List<Process> queue2)
{
    for (int i = 0; i < queue.Count; i++)
    {
        var process = queue[i];
        if (process.RemainingTime <= 0) continue;

        int executionTime = Math.Min(timeQuantum, process.RemainingTime);
        process.RemainingTime -= executionTime;
        currentTime += executionTime;

        if (process.RemainingTime == 0)
        {
            process.CompletionTime = currentTime;
            queue.Remove(process);
        }
    }
}

```

```

        تعديل المؤشر بعد الإزالة // ;
    }
    else if (nextQueue != null)
    {
        nextQueue.Add(process);
        queue.Remove(process);
        تعديل المؤشر بعد الإزالة // ;
    }
}
}
}

```

OUTPUT

	Process	AT	BT	CT	TAT	WT
1		0	8	17	17	9
2		1	4	7	6	2
3		2	9	26	24	15
4		3	5	12	9	4

Average Turnaround Time: 14

Average Waiting Time: 7.5

Chapter8(Deadlock)



System Model

النظام يتكون من مجموعة من العمليات التي تتنافس على الموارد different resources وهي عبارة عن أنواع

Resources Such as : CPU Cycle - Memory Space -I/O Devices

ممكن موجود أكثر من مرة زي مثلاً ممكن يكون عندي معالجين او موصل طبعتن مثلاً

وعشان الـ process يستخدم الـ Resource هتمشي بـ 3 مراحل

1. request :to get it
 2. use
 3. release
- ترجمة عشان لو عملية تانية عوزة تستخدمو:



Deadlock Characterization

قلنا قبل كدا ان ال Deadlock يحصل لما عملية مستندة حدث يحصل يكون في عملية اخري مستندياها تخلص فكدا وللي دي هتخصل طب اي الحجات الي تحصل عشان اقول ان حصل Deadlock يعني ان عملية واحدة مستخدمة ال resource في نفس الوقت والي عوزاه تستني يعني تكون في عملية معها عالاقل one resource ولكن هي مستندة resources يعني تكون في مساحة فبالتالي هتسندي انها تفضي يعني ال resource يحيطها عن طريق العملية نفسها مش هتسبيو غير لما تخلص يعني مفيش اجراء انها تسبيو بيكون عندي مجموعة من العمليات كل وحدة مستندة الثانية انها تخلص فدخلنا في loop بردو ممكن نوصف مشكلة ال Deadlock عن طريق رسمة Graph



Resource Allocation Graph

Graph : contain a some Vertices(V) connected with each other with some edges(E)

Types of V

1. V is a Processes that is in system
2. V is a Resources that is in system

Types of E

1. Request Edge : Direct Edge $P \rightarrow R$ From Process to Resource لسه بتطلبو بس لسه لم تحصل عليه
2. assignment Edge : Direct Edge $R \rightarrow P$ حصلت عليه ومستحوزة عليه دلوقتي

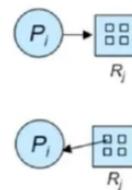
- Process



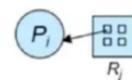
- Resource Type with 4 instances



- P_i requests instance of R_j



- P_i is holding an instance of R_j





طلب بعد مرسمت الـ Graph هل تعرف متى كان فيDeadlock

1. If graph has no cycle then there are no deadlock

2. If graph has cycle

فهنا هيكون عندنا شك ان معكן يكون في او لا ودا علي حسب

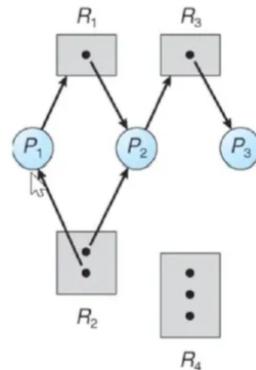
1. If there are one instance per resource type , then there are a deadlock

2. If there are many instance per resource type , then there are a possibility deadlock

one instance : الي هو في طابعة واحدة مثلا

many instance : الي هو عندي اكتر من طابعة مثلا موصلهم في الكمبيوتر

Examples



The sets P, R, and E:

- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_3, R_3 \rightarrow P_3\}$

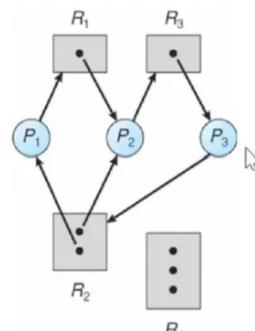
No Deadlock

P1 had (R2) and wanted R1

P2 had (R1,R2) and wanted R3

P3 had R3

There are no cycle then there are no deadlock

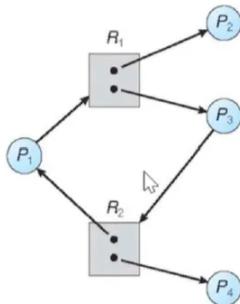


Two cycles exist in the system:

$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

There are two cycles then there are deadlock



a cycle is exist:
P1 → R1 → P3 → R2 → P1



هنا في deadlock بس مفيش cycle طب ليه
لان كل resource has many instance
تعالي نشوف p1 نظرا لان p2 مش عوزة another resources
وهي معاهما فلما هتخلص الـ P1 هتندو
ونفس الكلام مع الـ P3
فمش ديمًا اول متشوف cycle تقول في deadlock لزم تعمل check resources that had many instances

Methods for handling Deadlock



1. Deadlock Prevention: منع حدوثه قبل ميحصل
2. Deadlock avoidance: نعمل الجوريزم يتتجنب حدوثه
3. Deadlock detection and Recovery :Deadlock detection and Recovery او ميحصل يبدأ يحدل المشكلة ويخرجو من الـ deadlock
4. Ignore The Problem : ودا ان مبنعملش الجروزمات تحل المشكلة ونفترض أنها مش هتحصل

Deadlock Prevention

ودا عن طريق ان نحاو نمنع حدوث الـ 4 شروط الي بتؤدي لحدوثه

1. Allow a Mutual Execution: يعني العمليات اللي بنهاها resources مشتركة نسمح أنها تتعامل معها سويا

ولكن دا ميحصلش مع العمليات الي منش محتوية على resources shared لان دا هيؤدي الي

Data Inconsistency

نمنع حدوثه عن طريق لما عملية تطلب مورد معين لازم ميكونش مستحوذة على مورد اخر: Hold and wait

ودا عن طريق

1. ان هنديها كل الموارد الي محتاجها قبل ما تبدأ تنفذ

او ان انا اسمح ليها انها تطلب مورد بس بشرط متكونش مستحوذة على اخر 2.

المشكلة في الحل دا ان يحصل Low resource utilization and starvation

معني ان العملية هتبقي محتاجة كل الموارد قبل متشتغل فممكن يكون في واحد مشغول وبالتالي هتفضل مستوية وغيرها نفس الكلام وبالتالي هتحصل مجاعة للموارد

3. No Preemption هنا بقى هنجبر العملية انها تسيب الموارد الي معهاها لو طلبت موارد زيادة:

4. Circle Wait هنا حلها عن طريق هنرتب الـ resources تصاعديا ولما عملية تطلب واحد جديد :

لازم رقم الـ resource الي هتطلبو يكون اكبر من رقم المورد الي معهاها

Deadlock avoidance

ودا هعملو عن طريق هطلب من كل عملية قبل متشتغل انها تقول اي هي الموارد الي هتحتهاها وهي بتتنفذ

ودا هعملو عن طريق الجوربزم هيقوم قبل ميدي مورد لعملية هيبيقي عارف من قبلها هل لو انا ادخلها ممكن مع عملية اخري مثلًا كانت ممكنتيariadeadlock يسبلي

Safe State

دلوقتي جت عملية طلبة مورد معين مثلًا عوزة الطابعة وهي كانت فاضية لازم قبل مدخلها بيقي عارف لو وللي ممكن يحصل وطبعا هو كدا ادخلها النظام هيبيقا امن لسه يعني منش هيحصل deadlock

كل العمليات قبل مهتمشتنغل اي الموارد الي ممكن تدتهاها فلو كان هو عارف ان في وحدة هتعوز الطابعة

سعتها دا ممكن يسبب الـ deadlock

طب امتي يكون المنظام في الـ Safe State

بساطة لو اتوفر لها كل الموارد اي محتاجها عشان تتنفذ وكمان توفر كل الموارد الي كانت مستخدمة من عمليات قبلها فبتقى متاحة بدو هو او منش هتسخدمها بس دا دليل علي ان النظام شغال بشكل مظبوط

يبيكي طول والنظام في الـ Safe State منش هيحصل فيه deadlock

Avoidance Algorithms Type

1. if recourse has a single instance then use a resource allocation graph

2. if recourse has a many instances then use a Banker's Algorithm

resource allocation graph

زي ما قلنا المفروض كل عملية قبل متبدأ تعرفنا اي الموارد الي هتعتهاها فهنسخدم حاجة اسمها

يعني العمليات هندلها الموارد الي محتاجها بخط منقط Claim Edge(Priori): P.....R

ولما العملية تطلب فعليها المورد يتتحول الى Claim Edge

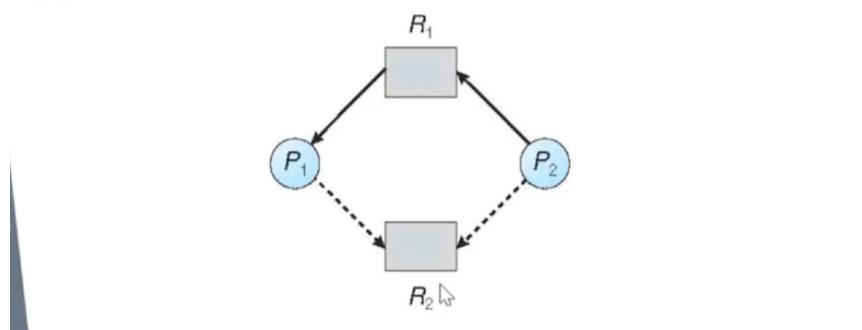
ولما العملية تحصل عليه سعتها هيبيقي Assignment Edge

ولما تسبها ترجع تاني Claim Edge

طب دلوقتي ازاي هنسخدم الـ jorbzem دا عشان نمنع حدوث الـ deadlock

لو قدرت تدول من Request To Assignment Eges
ومحصلش Cycle wait فانت كدا مashi تمام لو حصل منش هتسمع ان العملية تأخذ المورد

Resource-Allocation Graph

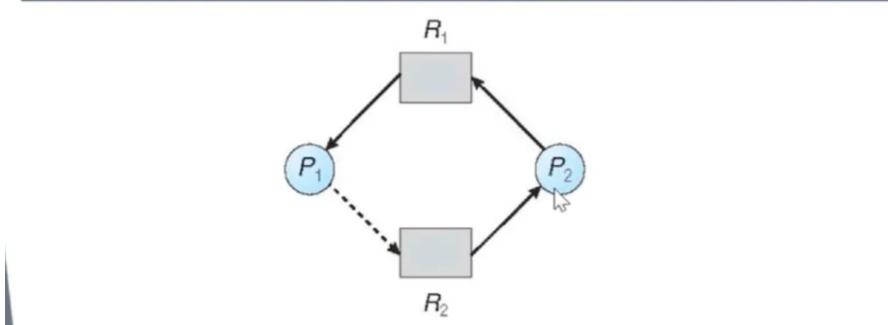


P1 and P2 هي علمنا ان R2

P2 wanted R1

R1 assigned by P1

Unsafe State In Resource-Allocation Graph



الطلب هنا الي طلبتو P1 هررخه ولان لو اتقبل هيسبب (Deadlock Cycle Wait)

Banker's Algorithm

Here we had a multiple Instances of Resource

هنا بدو لازم العملية تعلن عن الموارد الي محتاجها وعلس اساسهم هنعرف هل سعتها النظام هيبيقي
in safe state or not if not then not accepted a process to get a resource

Data Structure of Banker's Algorithm

تعالي نفترض ان عندي n of Processes and m of resources

Available: Vector of Length m

القيم الي فيه هي عبارة عن الـ number of instances of resource

اکن بقول عندي کام واحد من نوع مورد معین $\rightarrow k$
 $k \rightarrow$ is a number of instances of type j

Max : is a matrices n*m

if $\max[i,j] = k$ then

اکن پردازی Process || اسٹرنچنگ

K (number of) resource of type j الثانية

هنا هنقول كل الموارد التي مرت بها العملية من نوع معين

Allocation: matrices n*m Allocation[i,j]=k

يعنى العملية حذت موايد عددها من نوع

Need: باقی || resources, نقصانی، و لسه مفترها

$$\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$$

Safety Algorithm

Banker's algorithm in safe state , || , لجی چیز

رسالة عن طيبة، إن آناء، نعاف ترتيب العمليات وله قدمنا نسب الترتيب برقى، النظاره فم، اليمان:

جواب

1 here we had a two vectors Work and Finish

Work : Available

ثم نبحث عن العميم المطبقة للشرط التالي

2. check ? $\text{Finish}[i] = \text{false}$ & $\text{Need} \leq \text{Wor}$

If a check result = false then go to step 4

3. Work=Work + Allocation & Finish[i]=true

go to step 2

4. if $\text{finish}_i = \text{false}$ for all i then system in safe state

تعالٰی ناخد امثالہ عشاں نفهم کل الکلام دا

Example of Banker's Algorithm

- 5 processes P_0 through P_4 ;
- 3 resource types:
A (10 instances), B (5 instances), and C (7 instances)

- Snapshot at time T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

نـي ما واضح في الصورة عندنا مجموعة عمليات وفي منها مستدروـز علىـي resources(Allocation)

والـيـبـوـضـعـ عددـ المـوـارـدـ الـيـ الـعـمـلـيـةـ مـحـتـجـاـهـاـ عـشـانـ تـشـتـغلـ

هيـ المـوـارـدـ المـتـاحـةـ وـلـوـ تـفـتـكـرـ قـلـاـنـ هـتـزـيدـ لـمـاـ عـمـلـيـةـ تـشـتـغلـ وـتـسـيـبـ المـوـارـدـ الـيـ كـانـتـ مـعـاهـاـ

Example (Cont.)

- The content of the matrix **Need** is defined to be **Max – Allocation**

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	3 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

P0 not work and P1 will work then it will set an allocation resources after finish

Example (Cont.)

- Find a process i where $\text{Need}_i \leq \text{Available}$: P_1

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	3 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

-
- Update **Available** by **Available + Allocation_i**,

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	3 3 2
P_1	2 0 0	3 2 2	1 2 2	5 3 2
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

- Find a process i where $\text{Need}_i \leq \text{Available}$: P_3

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	5 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

-
- Update **Available** by **Available + Allocation_i**,

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	5 3 2
P_1	2 0 0	3 2 2	1 2 2	7 4 3
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

Find a process i where $\text{Need}_i \leq \text{Available}$: P_4

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	7 4 3
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

- Find a process i where $\text{Need}_i \leq \text{Available}: P_0$

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	7 4 5
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

- Find a process i where $\text{Need}_i \leq \text{Available}: P_0$
- Update **Available** by $\text{Available} + \text{Allocation}_i$

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	7 4 5
P_1	2 0 0	3 2 2	1 2 2	7 5 5
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

- Find a process i where $\text{Need}_i \leq \text{Available}: P_2$
- Update Available by $\text{Available} + \text{Allocation}_i$

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	7 5 5
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

The system is in a safe state since the sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety criteria

Resource Request Algorithm

دلوقي لـ عملية طلبة موارد اعلنت عنها لازم اتأكد الاول هل هيفضل النظام امن ام لا
هنهشي على الخطوات التالية

العملية طلبة موارد لم تعلن عنها

2. Request \leq Available \rightarrow if false then wait

نفترض ان احنا ادناها الي عوزا وبعد كدا نبقي نشوف هل لسه احنا في الامان ام لا عن طريق

1. Available=Available - Request

2. Allocation = Allocation - Request

3. Need=Need - Request

هنا بقا لـ عملية كل دا وفينا في الـ safe state

ولو لـ سعتها مش هنطبق اي حاجة من الي افترضناه

Example

Example: P_1 Request (1,0,2)

- Check that Request \leq Need₁ (that is, $(1,0,2) \leq (1,2,2) \Rightarrow \text{true}$)
- Check that Request \leq Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow \text{true}$)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	3 3 2
P_1	2 0 0	1 2 2	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

$$\text{Available} = \text{Available} - \text{Request}_i$$

-
- Check that Request \leq Need₁ (that is, $(1,0,2) \leq (1,2,2) \Rightarrow$ true)
 - Check that Request \leq Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	2 0 0	1 2 2	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

*Available = Available – Request;
Allocation_i = Allocation_i + Request_i;
Need_i = Need_i – Request_i;*

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- **Executing safety algorithm** shows that sequence $< P_1, P_3, P_4, P_0, P_2 >$ satisfies safety requirement
 - Can request for (3,3,0) by P_4 be granted?
 - Can request for (0,2,0) by P_0 be granted? 

Example 2: P_4 Request (3,3,0)

-
- Check that Request \leq Need₁ (that is, $(3,3,0) \leq (4,3,1) \Rightarrow$ true)
 - Check that Request \leq Available (that is, $(3,3,0) \leq (2,3,0) \Rightarrow$ False)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Can request for (3,3,0) by P_4 be granted?
 - No, a request for (3,3,0) by P_4 cannot be granted, since the resources are not available. 

Example 3: P_0 Request (0,2,0)

- Check that Request \leq Need₁ (that is, $(0,2,0) \leq (7,4,3) \Rightarrow \text{True}$)
- Check that Request \leq Available (that is, $(0,2,0) \leq (2,3,0) \Rightarrow \text{True}$)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Example 3: P_0 Request (0,2,0)

- Check that Request \leq Need₁ (that is, $(0,2,0) \leq (7,4,3) \Rightarrow \text{True}$)
- Check that Request \leq Available (that is, $(0,2,0) \leq (2,3,0) \Rightarrow \text{True}$)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 3 0	7 2 3	2 1 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Example 3: P_0 Request (0,2,0)

- Check that Request \leq Need₁ (that is, $(0,2,0) \leq (7,4,3) \Rightarrow \text{True}$)
- Check that Request \leq Available (that is, $(0,2,0) \leq (2,3,0) \Rightarrow \text{True}$)

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 3 0	7 2 3	2 1 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Can request for (0,2,0) by P_0 be granted?
 - a request for (0,2,0) by P_0 cannot be granted, even though the resources are available, since the resulting state is unsafe

Chapter9(Memory Management)

تعالي نفتكر كنا بنقول اي عن الموضوع دا في اول شافت

storage structure in chapter 1 لرجوع



في البداية نعرف لما تبيجي عملية معينة من الـ CPU

عوزة تعمل في الـ memory

هنهعرف منين ان كان العنوان دا صح ام هيتبعدي المحتاج ليها ؟

الموضوع بيهمشي كالتالي:

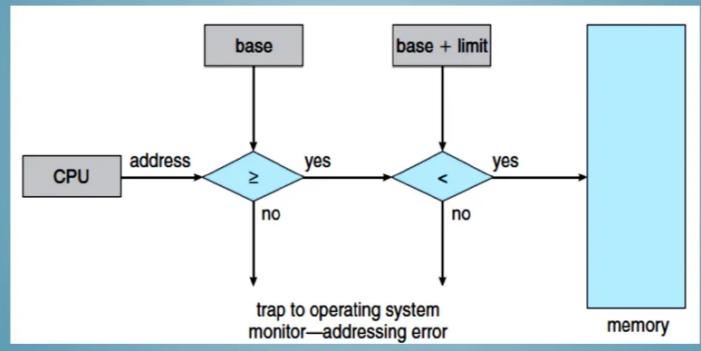
بنأخذ العنوان ونقارنو بيديتها في الـ memory الي بنسميه الـ base

لو كان اكبر او بيساوي فنخش على الاختبار الي بعدو والي بيقول

نجمع الـ base + limit ولو كان العنوان الي معهاها اقل

فههـي كـد ا تمام وقدر تدخل غير كـدا مش مسمـلـها وهـيـطـلـع Trap

□ Memory protection using base and limit registers.



طب هو اي العنوان الي بيجي مع العملية ومنين بيطلعوه ؟

CPU Generated this address known as Logical Address

نبدأ نأخذ الـ Logical Address

ونقارنو وفي عندنا 3 طرق للحصول على الـ physical address ودمجـة مع الـ logical address

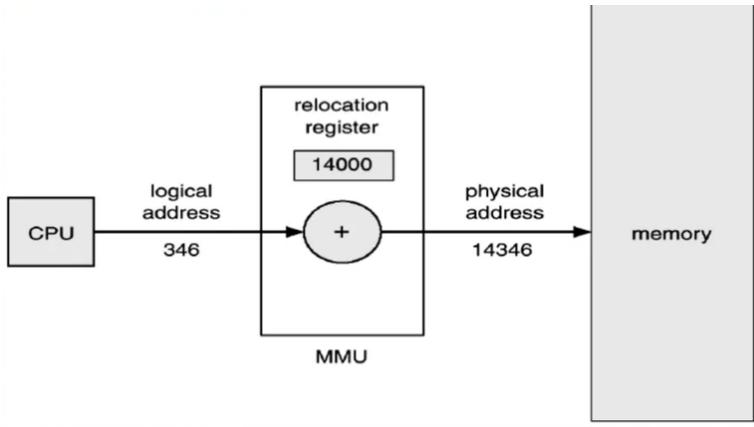
وهـنا هـنـجـبـ العنـوانـ اـثـنـاءـ التـرـجمـةـ وـيـطـلـعـانـاـ

1. **Compile time :** absolute code

2. Load time : generated a relocatable Code

3. Execution time : here we wanted a H.W(MMU)

Memory management unit(MMU)



Advantages:



1. OS can easily move a process during execution
 2. OS can allow a process to grow over time
 3. Simple, fast hardware two special registers, an add, and a compare
- بساطة MMU مكتننا ان نقدر ننقل العملية وهي شغاله
لان وهي شغلة طبيعي انها مساحتها بتزيد وبالتالي دا ساعدنا ان نتعامل معها

Disadvantages



1. Slows down hardware due to the add on every memory reference
ودا بسبب وقت المقارنة الي بنأخذو
2. Can't share memory (such as program text) between processes
لان كل عملية ليها بدايتها ونهايتها في الـ Memory فالتتعامل هنا هيبيقي بالـ Message passing
3. Process is still limited to physical memory size
4. Degree of multiprogramming is very limited since all memory of all active processes must fit in memory
ودا بسبب العيب الي قبلها انها محدودة بمساحتها في الذاكرة

Swapping



لو تفتقرا حاجة اسمها

Medium Term Scheduler : Memory الخاصة بعملية معينة تتملي

تنقل جزء منها لل Disk عشان نفتح المجال للزيادة

المكان الي نخزن فيه علي الهايد بنسمية

وعملية التبديل بتأخذ طبعا وقت وهو ال Swiping time

Backing store is fast disk large enough to accommodate copies of all memory images for all users it must provide direct access to these memory images

Major part of swap time is transfer time where total transfer time is directly proportional to the amount of memory swapped

Swapping on Mobile Systems

mobile operating system designers avoid swapping Why?



ودا بسبب الامكانيات المحدودة للجهاز وكمان المساحة (flash memory)

وكمان عشان تحافظ على عمر البطارية لفتره اطول ونظرا لان ال flash memory لها عمر محدود

لعدد المرات المسموحة بالكتابة عليها فهياكون كتر منهونعمل تبديل مساحة مختلف



when free memory falls below a certain threshold what OS do ?

لما الذاكرة الفارغة في الموبايل

بتنخفض تحت حد معين، النظام بيبدأ يتصرف عشان يحافظ على الأداء ويعن التباطؤ. والطرق دي (سواء في أندرويد أو iOS) بتختلف شوية بين النظمين، بس في العموم بيحصل كده:

في أندرويد:

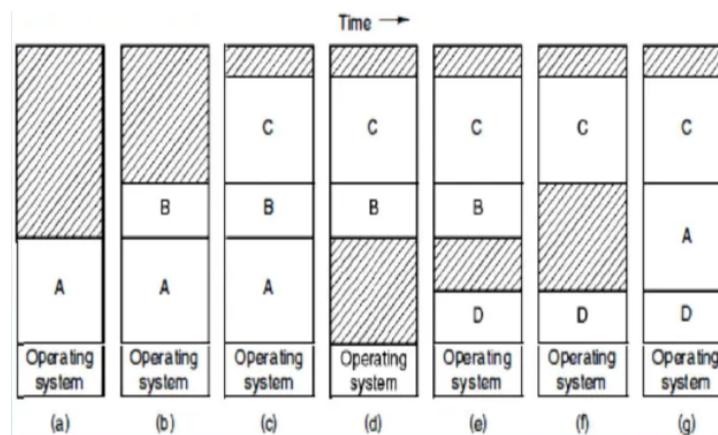
- إغلاق التطبيقات في الخلفية:** لو الذاكرة بقت قليلة، أندرويد بيبدأ يقفل التطبيقات اللي مش شغالة في الوقت الحالي (في الخلفية) عشان يوفر مساحة للبرامج اللي شغالة قدام المستخدم يعني بدل ما يحط البيانات في، أندرويد بيستخدم تقنية اسمها "ضغط الذاكرة"
- ضغط الذاكرة:** أندرويد يضغط البيانات دي عشان يأخذ مساحة أقل في الذاكرة ويخلص مساحة أكبر للتطبيقات اللي شغالة الذاكرة بطريق بتتطلب مساحة كبيرة، بيضغط البيانات دي عشان يأخذ مساحة أقل في الذاكرة ويخلص مساحة أكبر للتطبيقات اللي شغالة
- لو الذاكرة بقت قليلة جدًا ولسه في حاجة لمزيد من المساحة، أندرويد (في الحالات القصوى) swap استخدام التخزين كـ لكن ده بيكون في الحالات القصوى عشان swap كـ SSD معكين يستخدم جزء من التخزين الداخلي (ال فلاش ميموري) زي الـ RAM مبيكونوش سريع زي الذاكرة العشوائية

في iOS:

- يوقف التطبيقات اللي مش شغالة في الخلفية عشان يحافظ على الذاكرة iOS، **إغلاق التطبيقات في الخلفية:** زي أندرويد لتطبيقات المستخدم اللي شغالة قدامهم
- يحاول يوازن بين استخدام الذاكرة من خلال إدارة أفضل للموارد. النظام بيقفل أو يوقف التطبيقات iOS : **إدارة أفضل للذاكرة** اللي مش تحتاجها في الوقت الحالي ويعطي الأولوية للتطبيقات النشطة زي أندرويد، لأن النظام مصمم بطريقة أفضل لإدارة الذاكرة swap عموماً مش بيستخدم التخزين كـ iOS **مفيش** من غير ما يحتاج لذلك. لو الذاكرة بقت منخفضة، عادةً ما بيتم إيقاف التطبيقات أو تقليل استهلاك الذاكرة من التطبيقات اللي في الخلفية

Contiguous Allocation

يتم تسكين العمليات في أماكنها الخاصة بشكل متصل والعمليات هتنتحط وري بعضها ولو مثلاً وحدة طلعت البقفين هييفضلوا في مكانهم طب مكدا هيطلع عندي اجزاء كبيرة واجزاء صغيره ولو جت عملية مثلاً مساحتها مش مقصورة الي موجود وتهت้อง ان ندمج المساحات الموجودة عشان هي بتتنحن بشكل متصل فظاهر شوية الجروزمات عشان تدرلي عمليات التسخين بشكل فعال





Memory Allocation Algorithms:

1. First Fit Allocate the first hole that is big enough

اول مكان هنلاقيه بده فيه

2. Next Fit Allocate the first hole that is big enough (like first fit) however, the searching process starts at the location where the previous searching ended

نفس الكلام بس هبدأ من اخر مكان كنت فيه مش من الاول خاليص

3. Best fit Allocate the smallest hole that is big enough must search entire list, unless ordered by size Produces the smallest leftover hole

هنخزن في اقل مكان يكون كافي

4. Worst fit Allocate the largest hole must also search entire list Produces the largest leftover hole

هنخزن في اكبر مكان موجود

متيجي ناخد مثال عشان نفهم



Example

Example Consider a swapping system in which the memory contains the holes of the following sizes in order 50 KB, 12 KB, 20 KB, 40 KB, 32 KB, 9 KB, 22 KB and 45 KB Which holes are allocated for successive segment requests of

(a)

20 KB (25 KB (7 KB

using

first fit best fit worst fit and next fit

(

Hint if the segment request is smaller than the allocated hole, ignore the remaining hole)

هنا هنسكن العملية وتجاهل الباقي منها

	First Fit	Best Fit	Worst Fit	Next Fit
(a) 20 KB	50	20	50	50
(b) 25 KB	40	32	45	40
(c) 7 KB	12	9	40	32



- Example Consider a swapping system in which the memory contains the holes of the following sizes in order 10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB, and 15 KB Which holes are allocated for successive segment requests of
(a)12KB (b)10 KB (c)9 KB

using

first fit best fit worst fit and next fit

	First Fit	Best Fit	Worst Fit	Next Fit
(a) 12 KB	$20 \Rightarrow 8$	12	$20 \Rightarrow 8$	$20 \Rightarrow 8$
(b) 10 KB	10	10	$18 \Rightarrow 8$	$18 \Rightarrow 8$
(c) 9KB	$18 \Rightarrow 9$	9	$15 \Rightarrow 6$	9



Fragmentation

اصغر من مشكلة بتظاهر عندنا بتخلينا منقدرش نستغل الذاكر لأن بيبقا الاذاطرة متقطعة لاجزاء وكل جزء الى العمليه تحتاجها ودا بيحصل في ال

Types of Fragmentation:

1. External fragmentation: هنا هيكون عندي مساحة كافية لكن غير متصلة
 2. Internal fragmentation: يعني هيكون عندي جزء من block هقدر استفيد منه
- زي مثلاً عاوز اخزن 1000 واعل 1024block وهذن ال 1000 فخذنوا ال



Solutions for external fragmentation

1. Compaction: يعني هخلي المساحات المليانه جنب بعضها والفااضي جنب بعضه
2. Another possible solution to the external fragmentation problem is to permit the logical address space of the processes to be noncontiguous ..
(Paging and Segmentation)

يعني اخلي العمليه الواحدة تتخزن في اكتر من مكان عن طريق تقطيعها لاجزاء وكل جزء في مكان

Paging

1

Paging

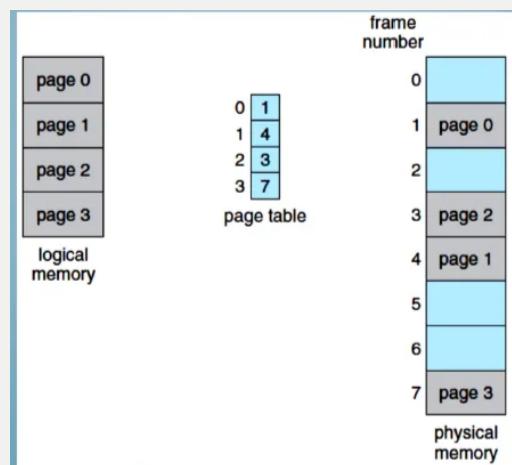
هي طريقة للدارة الـ memory بتسمح ان الـ physical address to be noncontiguous

وذا هيت عن طريق هنقسم العمليه الى اجزاء متساوية والذاكرة الي اجزاء بنفس الحجم ونبأ نحط كل جزء من العمليه في جزء من الذاكرة

- Divide logical memory into blocks of same size called pages
- Divide physical memory into fixed sized blocks called frames

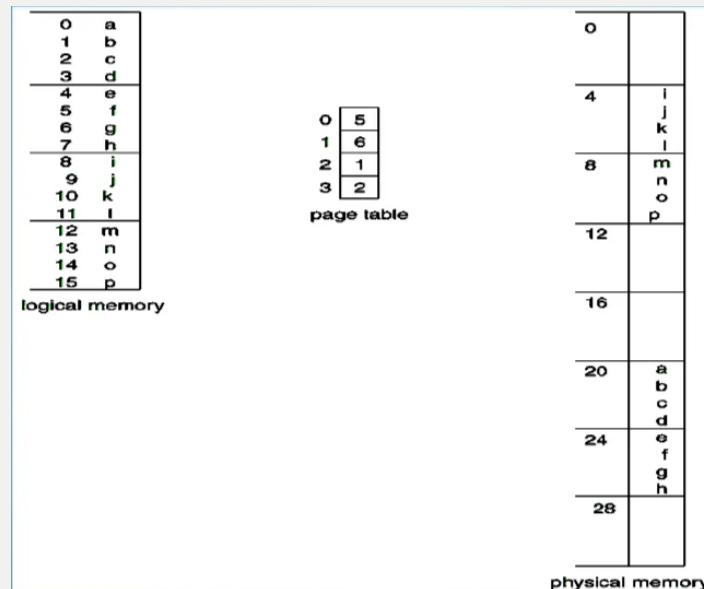
Page size frame size (size is power of 2)

To run a program of size n pages, need to find n free frames and load program



كدا حلينا مشكلة ال External Fragmentation

pages has same size لان الـ internal Fragmentation بس لسه نوعاني من





Address Translation Scheme

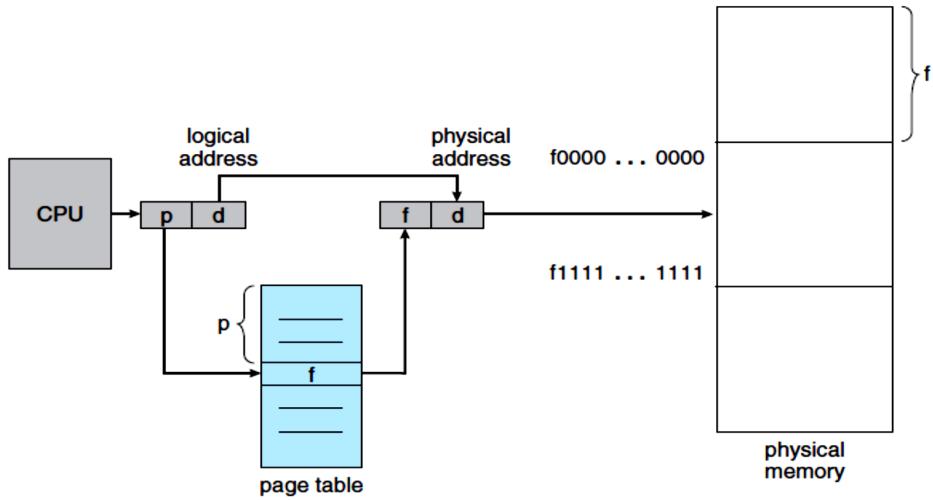
Logical address: Address generated by CPU is divided into:

- Page number (p): Used as an index into a page table which contains base address of each page in physical memory
- Page offset (d): Combined with base address to define the physical memory address sent to the memory unit

Physical address: Address seen by memory is divided into:

- Frame number (f): Obtained from page table
- Frame offset (d): Combined with base address to define the physical memory address sent to the memory unit

	Logical address	page number(Logical Memory) = logical / size(4)=بنحدد القيمة الصحيحة	frames number(Physical Memory)	offset in logical memory= logical % size	Physical Address	page number(Physical Memory) = logical / size(4)=بنحدد القيمة الصحيحة	offset in Physical memory= logical % s
a	0	0	5	0	20	5	0
b	1	0	5	1	21	5	1
c	2	0	5	2	22	5	2
d	3	0	5	3	23	5	3
e	4	1	6	0	24	6	0
f	5	1	6	1	25	6	1
g	6	1	6	2	26	6	2
h	7	1	6	3	27	6	3
i	8	2	1	0	4	1	0
j	9	2	1	1	5	1	1
k	10	2	1	2	6	1	2
l	11	2	1	3	7	1	3
m	12	3	2	0	8	2	0
n	13	3	2	1	9	2	1
o	14	3	2	2	10	2	2
b	15	3	2	3	11	2	3



Example

: Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.

- How many bits are there in the logical address?
- How many bits are there in the physical address?

تعالی نمشی واحدة واحده

How many bit of

- Pages = $\log(8)$ base 2 = 3 bits
- offset bits = $\log(1024)$ base 2 = 10
- frames bits = $\log(32)$ base 2 = 5
- logical address bits = $3+10=13$ bit
- Physical address bits = $5+10 = 15$ bit

ممكن بقى يديك ال physical address وبقلبك عاوز ال logical address and page table

فاحنا لو ضرينا رقم الصفحة * حجم الصفحة هيديننا بداية الصفحة نجمع عليها ال offset

Translate Logical address to physical address

- Example : Find physical address for the logical address char f at 5

- Solution:

Logical address

$$p = \text{int}(\text{logical address} / \text{page size}) = \text{int}(5/4) = 1$$

$$d = \text{mod}(\text{logical address} / \text{page size}) = \text{mod}(5/4) = 1$$

physical address

$$\text{from page table } f = 6$$

$$d = 1$$

$$\text{physical address} = (f \times \text{page size}) + d$$

$$= (6 \times 4) + 1 = 25$$

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

0	5
1	6
2	1
3	2

page table

logical memory

logical and wanted a physical address wanted a physical address

Translate physical address to Logical address

- Example : Find logical address for the physical address char o at 10

- Solution

physical address

$$\bullet \quad f = \text{int}(\text{physical address} / \text{frame size}) = \text{int}(10/4) = 2$$

$$\bullet \quad d = \text{mod}(\text{physical address} / \text{frame size}) = \text{mod}(10/4) = 2$$

logical address

$$\bullet \quad \text{from page table } p = 3$$

$$\bullet \quad d = 2$$

$$\text{logical address} = (p \times \text{page size}) + d$$

$$= (3 \times 4) + 2 = 14$$

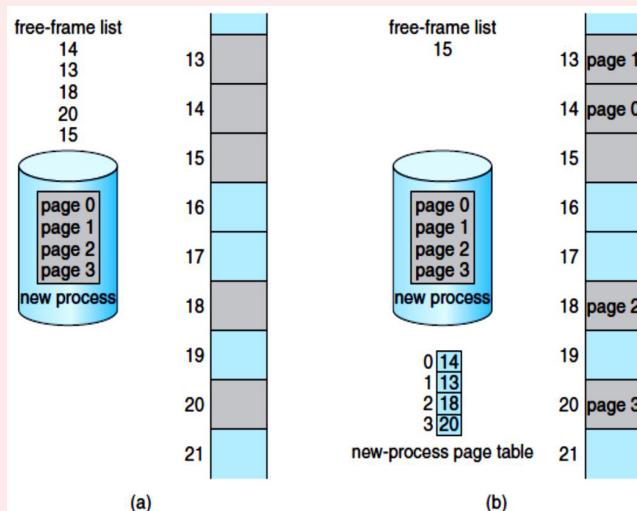
0	
4	i j k l
8	m n o p
12	
16	
20	a b c d

12



Free Frames:

عشنان يقدر نظام التشغيل يعرف اي المتاح عنده بيقي عنده جدول بردو يعرف من و هو frame table و عشان لما تيجي عملية عوزة تتسكن نقبي عرفين فين الاماكن المارحة



When context switch occur the dispatcher loads the paging table of the process into hardware page table

- Paging increase context switch time

لأن هاخد وقت على مجمع العملية كاملة تاني ودا علي عكس ال Contiguous Allocation
Internal fragmentation in last frame
مازلنا نعاني من مشكلة ال pages increase فقاً ممكن نعمل حل ان نقل حجم ال pages بس بردو هيظهر مشكلة ان عدد ال



Examples of Internal fragmentation problem in paging

Example:

A process is 3.5 KB and frame size is 2 KB. Calculate the internal fragmentation.

Solution:

Number of frames = $\lceil \text{process size} / \text{frame size} \rceil = 2 \text{ frames}$

Internal fragmentation = (number of frames \times frame size) - process size = 0.5 KB

Example:

A process is 209 KB. Calculate the internal fragmentation for frame sizes of:

0.5, 1, 2, 4, and 8 KB.

Solution:

Frame size	No of frames	Internal fragmentation
0.5	418	0
1	209	0
2	105	1
4	53	3
8	27	7



طبع دلوقتي فين ال page table of every process

Page table base register (PTBR) : هيبيقي موجود فيه بدايتو

Page table length register PTLR : هيبيقي موجود فيه طولو عشان نعرف اخر و فين

memory two time وهي انك بتروح لل

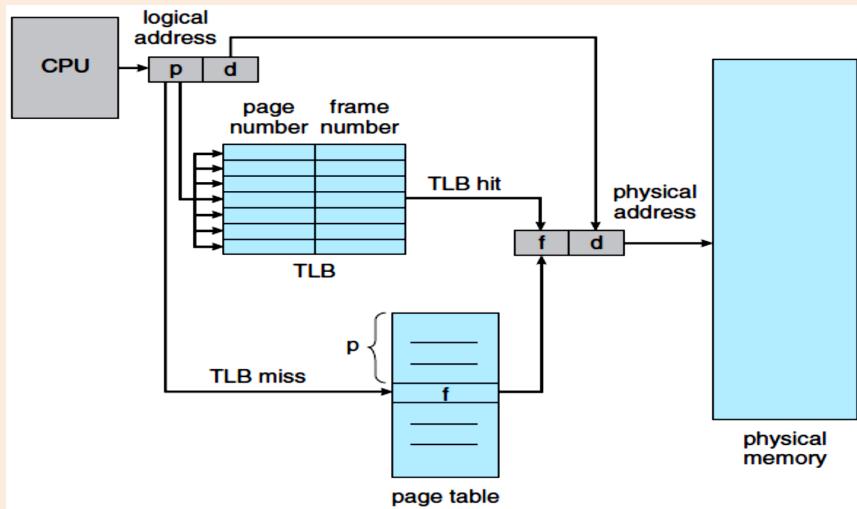
1. to find page number from page table
2. go to frame in memory

فالموضوع هنا هيبيقي بطيء فعملو حل ان نعمل نسخة في مكان اسمه (TLBs)

ونعمل caching in TLBs وهو سريع جدا



translation look aside buffers(TLBs):



Hit : TLB \Rightarrow Memory

Miss : TLB \Rightarrow Page Table \Rightarrow Memory : (1-Hit)

Effective Access Time (EAT) = $H * (\text{TLB access time} + \text{MA}) + (1 - H) * (\text{TLB access} + \text{PT access} + \text{MA})$.



Example

Suppose the paging hardware with TLB has a 90 percent hit ratio. Page numbers found in the TLB have a total access time of 100 nanoseconds. Those which are not found there have a total access time of 200 nanoseconds. What is the effective access time?

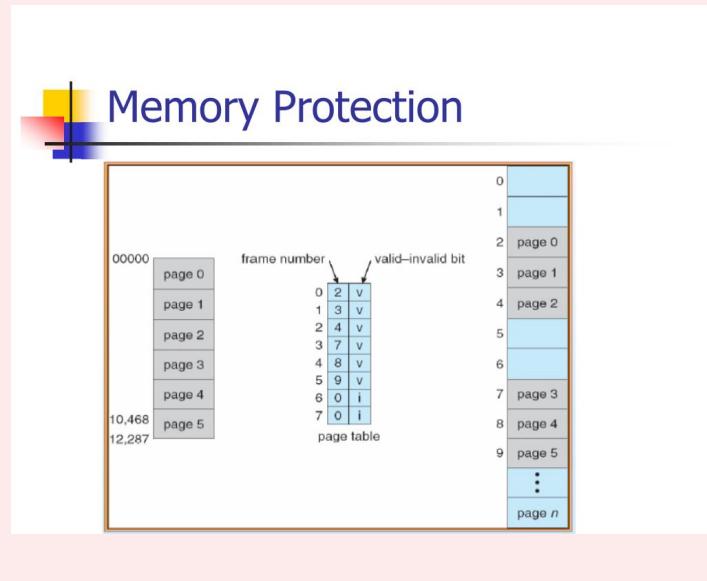
- a. 100 nanoseconds
- b. 110 nanoseconds
- c. 190 nanoseconds
- d. 200 nanoseconds

$$\text{EAT} = 90/100 (100) + (1 - (90/100)) * (200) = 110\text{ns}$$



Memory Protection:

ودا عن طريق زودنا bit in page table لـ هل الصفحة محجوزة ام لا
فبالتالي لو جيت تقللي عاوز الصفحة كذا هتشوف الاول هي محجوزة ام لا



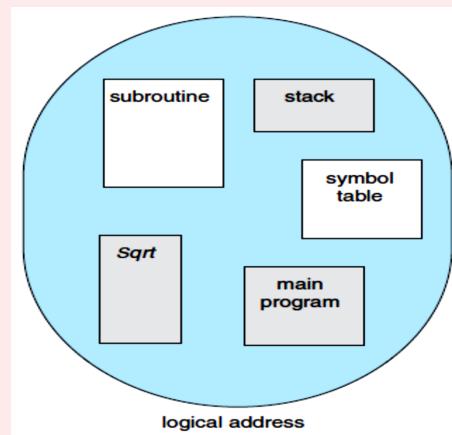
Segmentation

2

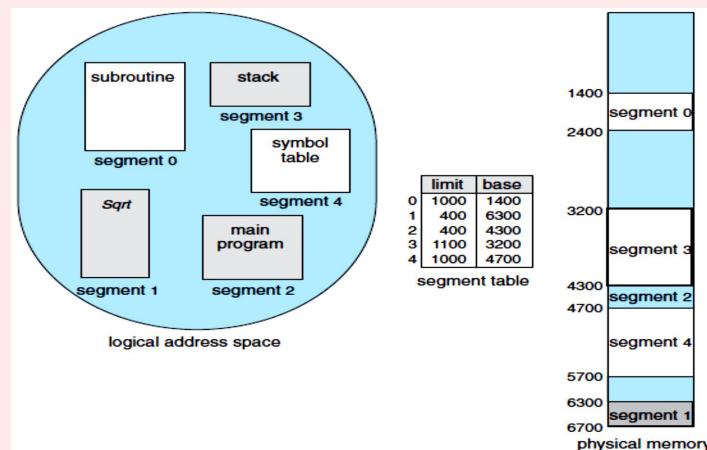
Segmentation:

عن طريق هنقسم الى اجزاء غير متساوية

وهي منقية اكتر من الـ paging هيبيقي العناصر المترابطة معا



هنا لما نيجي نسكن الـ segments هنحط كل واحد في المكان الي يناسبو
ودا عن طريق segment table

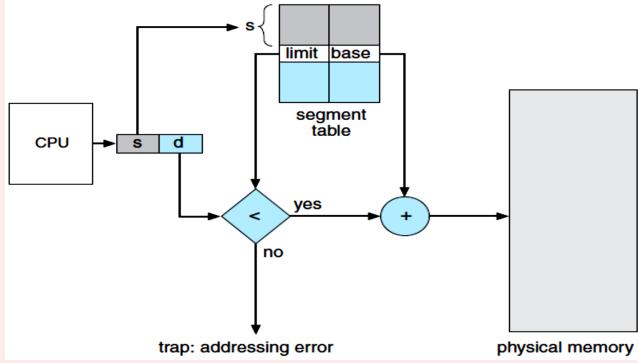


process come with a segment and his offset

EX: segment 1 & offset 100 \Rightarrow physical address = 6400

نفس الكلام هنعرف مكان الـ segment table طريقة

segment table base register (STBR) and Segment Table Length register(STLR)



Advantages

- Segment sharing
- Easier to relocate segment than entire program

- Flexible protection

- Efficient translation

- Segment table small

Disadvantages

- Segments have variable lengths ↪ how to fit?

- Segments can be large ↪ external fragmentation

وكذا رجعنا لنفس المشكلة بتاتع ال Contiguous Allocation

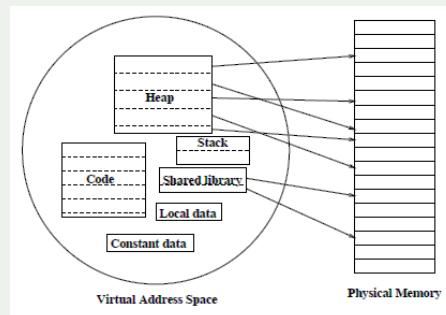
Consideration	Paging	Segmentation
Programmer aware	No	Yes
How many addr spaces	1	Many
VA size > PA size	Yes	Yes
Protect individual procedures separately	No	Yes
Accommodate elements with changing sizes	No	Yes
Ease user sharing	No	Yes
Internal fragmentation	Yes	No, in principle
External fragmentation	No	Yes
Placement question	No	Yes
Replacement question	Yes	Yes



Paging with Segmentation

حاول يأخذ المزايا في كلها عن طريق تقسيم ال process into segments and every segment into pages

Logical address will be \Rightarrow segment number and page number and offset



Here we had a Segment table contain a segment number \Rightarrow address of page number \Rightarrow offset

Physical Address : Frame number \Rightarrow offset

VIRTUAL MEMORY(last chapter)



اتعرفنا في السابق على الطرق المختلفة في التخزين
وكان من الضروري تواجد ال process
بالكامل في ال memory وبالتالي دا هبيط حمل عليها لأن معنكم محتاجش كل اللي حملتو
ومن هنا ظهرت ال virtual memory
من خلاها هيقي في ال memory جزء منها ولما احتاج أجزاء أخرى هيقي اعملها physical address
وكمان مبقاش يحكمنا ال logical address \leq physical address
فيما سبق كان عشان نخزن لازم يكون ال logical address $>$ physical address لو كان virtual memory
ولكن مع وجود ال virtual memory هيتهم اللجوء الي تقنية ال Swapping

Swapping is the process of moving data between **RAM** (physical memory) and the **disk** (storage) to free up space in RAM when it's full. This allows the system to run programs that require more memory than the available RAM.

معنكم بقى نعرف تعريف بسيط لل virtual memory

Virtual memory is a computer system feature that allows programs to use more memory than physically available by creating an illusion of a larger memory space, combining **RAM** and **disk storage**.

الخلاصة: شبهها كدا بفكرة تحميل ال kernel فقط لنظم التشغيل لأن مش منطقى مثلاً يكون نظام التشغيل مثل 10 جيجا وانت عندك ram8 فمش هينفع تدعلو كلو فبندمل الجزء المهم والباقي لما بنعوزو بنعملو وقت التشغيل



Benefits

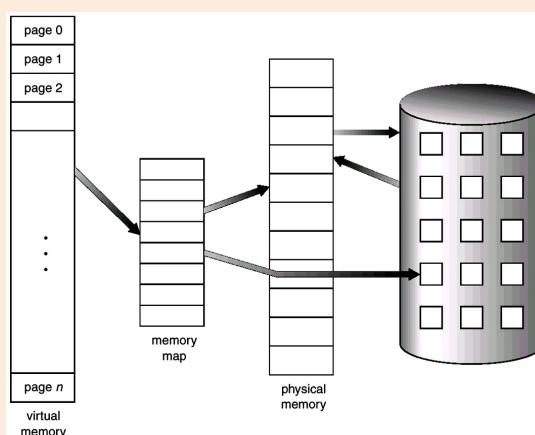
1. Only part of the program needs to be in memory for execution
2. Logical address space can therefore be much larger than physical address space
3. Allows address spaces to be shared by several processes
4. Allows for more efficient process creation
5. Programming much easier free programmers from concern of memory storage limitations

Demand Paging ب حاجة اسمها Virtual memory ينطبق ال



Demand Paging

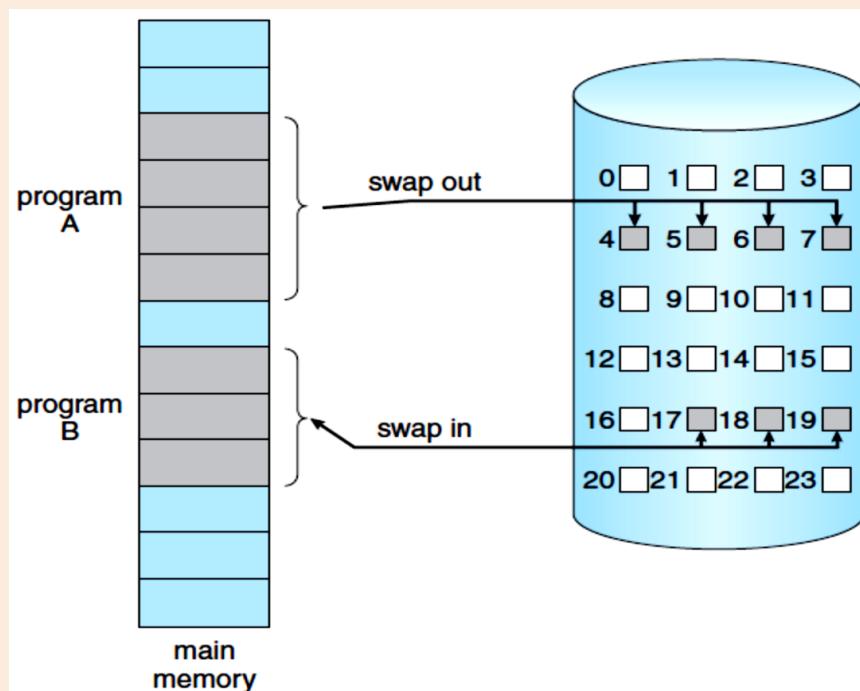
نفس فكرة الـ paging بس الفرق ان الـ logical memory اكبر من الـ physical memory



فبالتالي لما حجم الـ physical اتملي هنلأ الي الـ swapping

Demand paging is a memory management technique where only the required parts (pages) of a program are loaded into **RAM** when they are needed, rather than loading the entire program at once. If a required page is not in RAM, a **page fault** occurs, and the system retrieves the page from disk (virtual memory) into RAM.

This helps optimize memory usage and allows running large programs on systems with limited physical memory.



Swap in : from Disk to Ram

Swap out : from Ram to Disk

Advantages:

1. Less I/O needed.
2. Less memory needed.
3. Faster response.
4. More users.

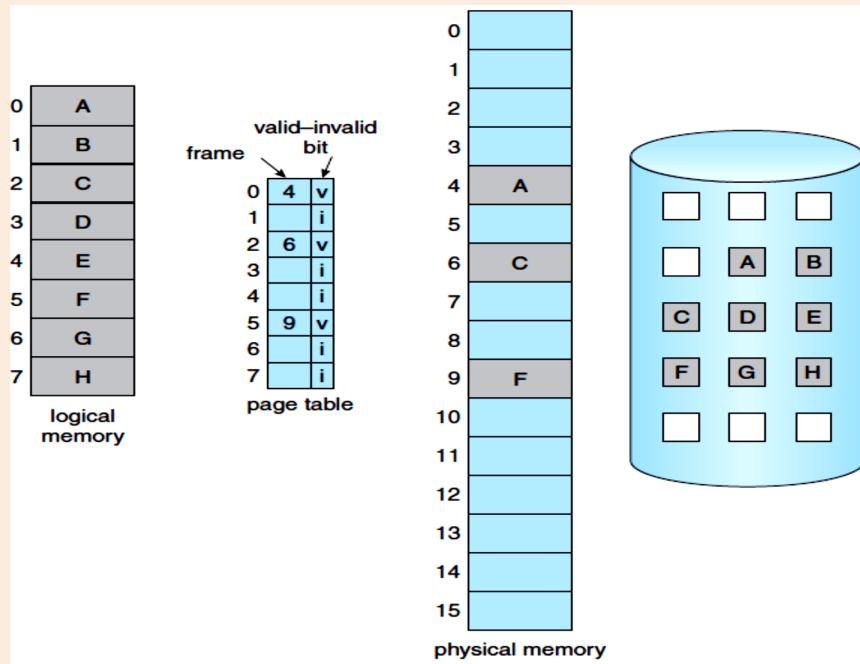
When a process is executed, demand paging uses lazy swapper
pager) that swap only pages that will be needed into memory
swapping processes
مِلْشُ زِيَّ إِلَيْ كَنَا خَدْنَاهُ قَبْلَ كَذَا كَانَ
swapping pages
لَكِنْ هَنَا هَنْجَمْل

Swapper : swap entire processes.
Pager(lazy swapper): swap individual pages of a process.

A valid or not valid bit في ال page table ان ضفنا bit نعرف منها ان الاختلاف هنا ان

V(valid): page is a part of process(logical space) and loaded in memory

I.Invalid): page is not a part of process or not loaded in memory(then bring it)



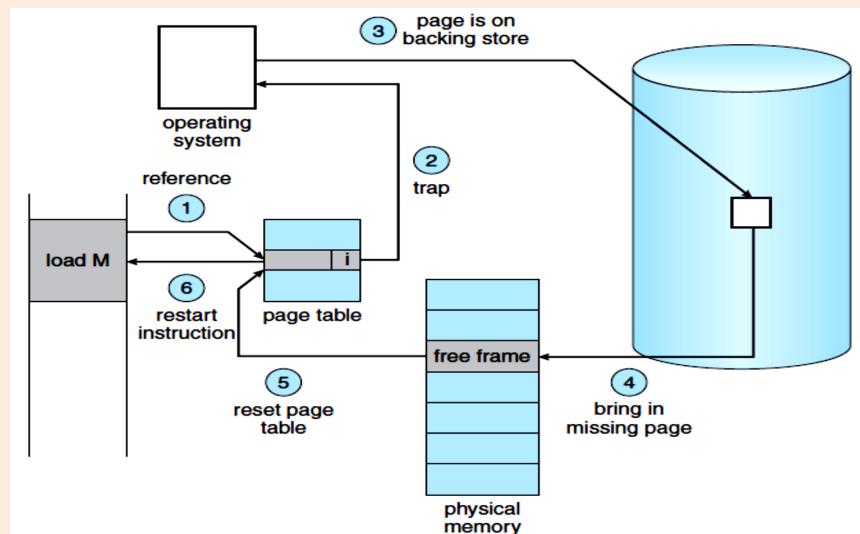
When page is needed, reference to it in internal table.

يعني اطلع وقلبي في مشكلة

If not in memory then bring it to memory. \Rightarrow روح هتها

طلب الـ page اللي متبلاش لسه اتحملت في الـ memory اي

Page Fault is interrupt that arises upon a reference to a page that is not in main memory.



ي بيحصل لـ page fault في الـ memory

1. process : يتم الاشارة الى الـ page وتأكد انها جزء من الـ process
2. trap \Rightarrow interrupt to OS
3. OS : Find the location of this page in Disk
4. load this page in free frame in memory
5. reset page table from valid to invalid : يغير حالتها
6. Restart again to use page that we wanted before

طبعاً افترض جه ينفذ الخطوة رقم 4 ملماً شاش مساحة فاضية هي متصرف ازاي؟

What happens if there is no free frame?

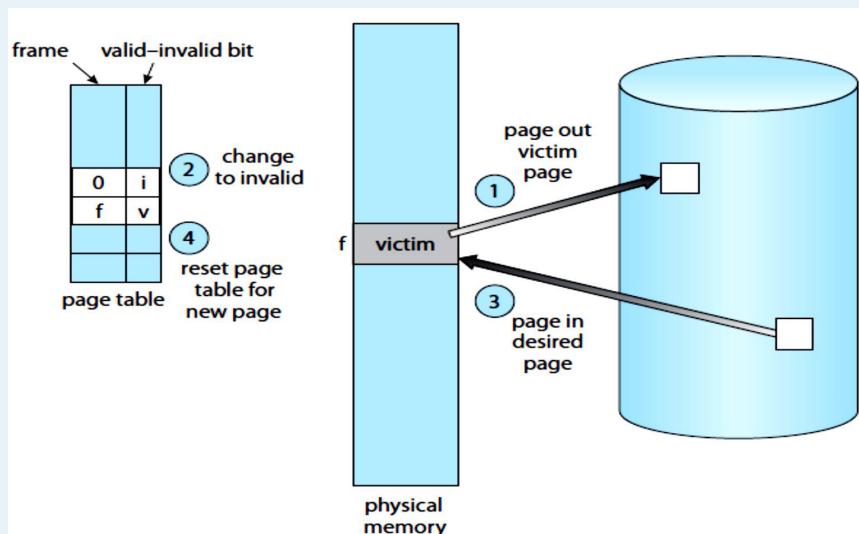
Page replacement finds some page in memory, but not really in use, swap it out

Replacement algorithm should result in minimum number of page faults

Same page may be brought into memory several times



Page Replacement



1. page out victim page : يحدد الضحية التي هنخرجها عشان نفعي مساحة طبقاً لـ الجوريزم معين :

2. Change to invalid : نغير حالتها في الجدول:

3. page in desired page : يجيب الـ page الذي عوزين نسكنها ونحطها مكانها:

4. Reset page table for new page : نغير حالتها في الجدول :



Page Replacement Algorithms

1. First In First Out (FIFO)
2. Optimal Page Replacement.
3. Least Recently Used (LRU).

1

First In First Out (FIFO)

بساطة هبيقي عدنا ونخزن فيها الـ pagesطبقاً انت عارف هي شغالة بمبدأ اي

Advantages

Low overhead implementation

Disadvantages

May replace heavily used pages

وبكدا هيزيد عدد الـ page fault هي كانت اي ارجع شوبيه هتلقيها

مش هيهمك غير المثال فيلا بينا



Implementation of FIFO

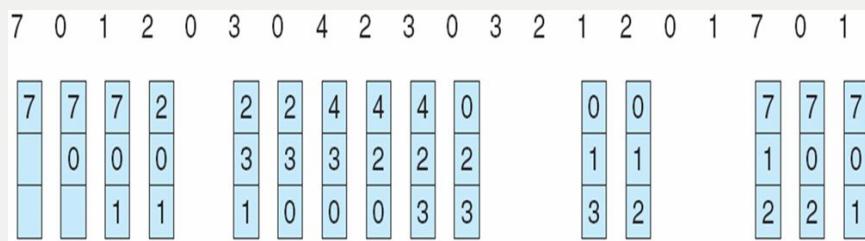
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

دول هما الـ pages عوزين نسكنهم

How many page faults would occur, assuming 3 frames?

متاح معاك 3 اماكن فقط وعاوز يحسب عدد الـ page faults

ركز كدا هنخزن واحدة بوحدة ولما ميبقاش عندي اماكن هطلع اول واحدة كانت داخله وندخل الي عليها الدور ولو في واحدة موجودة بالفعل مش هسكنها تاني بص عالصورة وهتهفthem



Total page fault occurs 15

Page Replacement = Total page fault occurs - Frames = 15 - 3 = 12

اول 3 مرات مبدلناش خلي بالك بدلنا لما مكنش عندنا اماكن فاضيه

2

Optimal Page Replacement

هنا بقى هنبعض لقادم يعني اي؟

وبالتالي page faults يعني هنبعض علي الي جوة ونشوف مين فيهم مش محتجها قريب هخرجها والي هحتاجها قريب هنلها
هنقل ال

Advantages

Optimal solution and can be used as an off line analysis method.

يعني مثلا لو انت جيت اخترت الجوربزم وعاوز تختبرو من حيث قلت الـ

Optimal solution هتقارنو مع الـ

Disadvantages

No online implementation.

صعب انها تتطبق لأنها نظرية أكثر

زي ببدو مكان حاصل في

انها نظرية وصعب انها تتنفذ عشان هنعرف منين ان كنت هحتاجها قريب ولی لا ؟

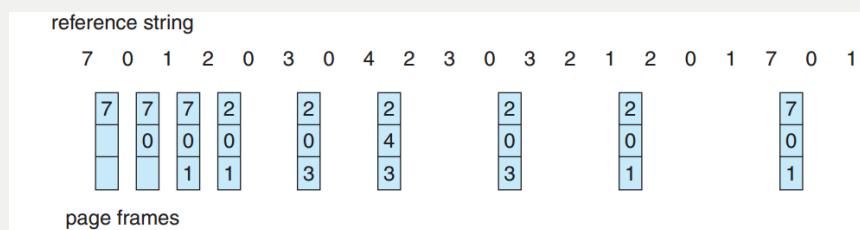
Optimal Vs. FIFO

FIFO algorithm uses the time when a page was brought into memory

Optimal algorithm uses the time when a page is to be used



Implementation of Optimal Page Replacement



1. 7 \Rightarrow add
2. 0 \Rightarrow add
3. 1 \Rightarrow add
4. to add 2 look after it 0 and 1 and 7 then 7 is last then kill it and add 2
5. 0 \Rightarrow exists
6. to add 3 look after it 0 \Rightarrow 2 \Rightarrow 1 in last then add replace 1 by 3
7. 0 \Rightarrow exists
8. to add 4 look after it 2 \Rightarrow 3 \Rightarrow 0 then replace 0 by 4
9. 2 exists
10. 3 exists
11. to add 0 look after it 3 \Rightarrow 2 then remove 4 and add 0
12. 3 exists
13. 2 exists
14. to add 1 look after it 2 \Rightarrow 0 remove 3 and add 1
15. 2 & 0 & 1 exists
16. to add 7 look after it 0 \Rightarrow 1 then remove 2 and add 7
17. here you had 7 0 1

Page faults: 12 & page Replacement = 12 - 3 = 9

3

Least Recently Used (LRU)

بنقص للخلف بمعنى بنشوف الـ pages اللي تستخدم بكثرة وونخرج الاقل طلبا

Disadvantages

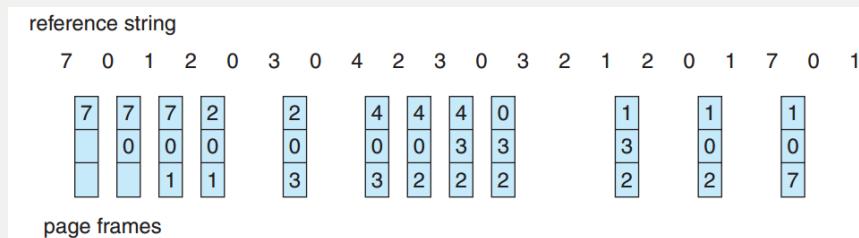
Expensive

maintain list of pages by references and update
on every reference

هيبقى في حمل كبير على نظام التشغيل



Implementation of Least Recently Used (LRU)



1. 7 \Rightarrow add
2. 0 \Rightarrow add
3. 1 \Rightarrow add
4. 2 add replaced by 7 because 7 is used before 0 , 1
5. 0 exists
6. 3 added replaced by 1 because 1 is last used before 2 , 0
7. 0 exists
8. 4 added by replaced by 2 because 2 is last used before 0 , 3
9. 2 added by replaced by 3
10. 3 added by replaced by 0
11. 0 added by replaced by 4
12. 3 & 2 exists
13. 1 added by replaced by 0
14. 2 exists
15. 0 added by replaced 3
16. 1 exists
17. 7 added by replaced 2

Page faults: 12 & page Replacement = 12 - 3 = 9



وبكدا يا صديقي نكون غطينا كل حاجة في الكورس تفصيلياً
مش طالب منك غير دعوة لو استفدت بمعلومة



في حالة وجود اخطاء او حاجة مش واضحة تواصل معايا