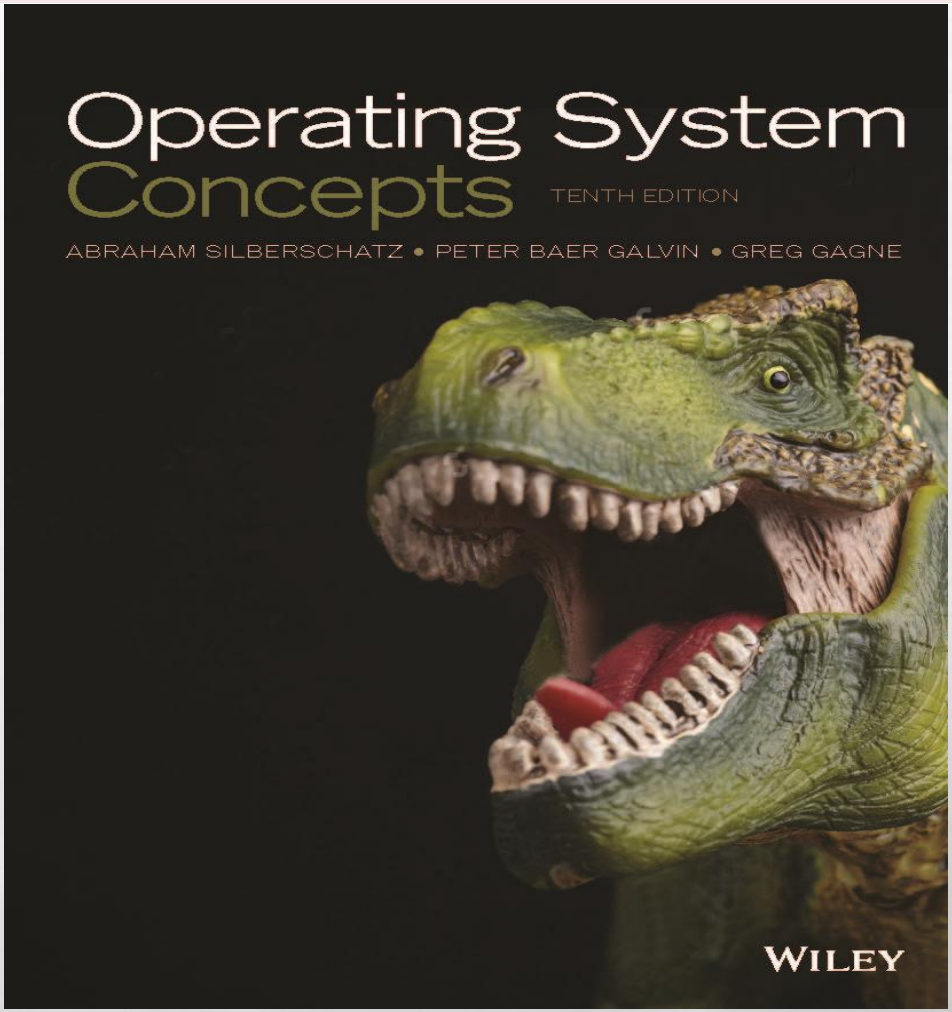
Mustansiriayah University
Collage of Education
Computers Science Department

Chapter One



Fourth Class

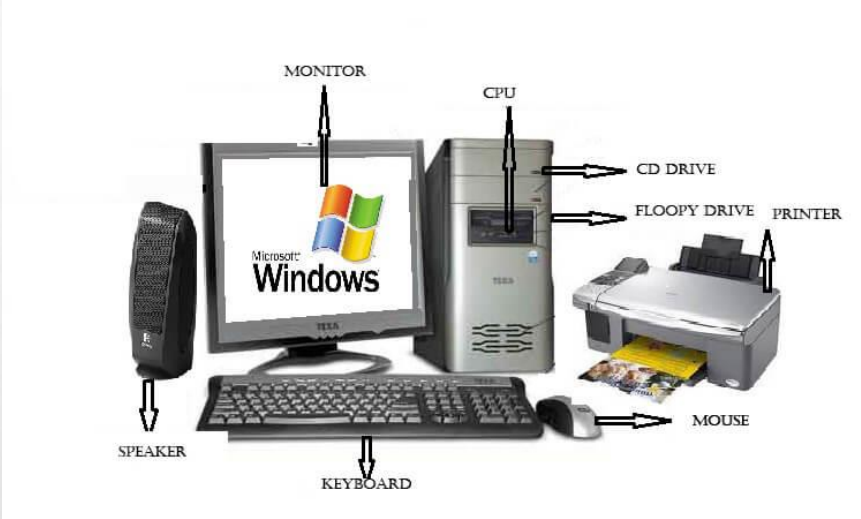
Assist. Prof. Dr. Hesham ALABBASI

2021-2022

1. Introduction

- A modern computer system consists of:

One or more processors, Main memory, Disks, Printers, a keyboard, a display, Network interfaces, and other Input/output devices (Hardware).



- Computers are equipped (مسلح) with a layer of software called the operating system, whose job is to manage all these devices and provide user programs with a simpler interface to the hardware.

أجهزة الكمبيوتر تضم مستوى من البرامج تسمى نظام التشغيل ، وتتمثل مهمتها في إدارة جميع هذه الأجهزة وتزويد برامج المستخدم بواجهة مبسطة لهذه المكونات المادية

1.1 What is an Operating System?

- **A program that acts as an intermediary (وسيط) between a user of a computer and the computer hardware.**

برنامج يعمل كوسيط بين مستخدم الحاسبة والمكونات المادية لها

- **The purpose of an operating system is to provide the environment (بيئة) in which the user can execute programs.**

الغرض من نظام التشغيل هو توفير بيئة مناسبة والتي من خلالها يستطيع المستخدم تنفيذ برامجه

- OS is a **resource allocator (مخصص الموارد)**
  - Manages all resources يدير كل الموارد
  - Decides between conflicting requests for efficient and fair resource use

يقرر بين الطلبات المتضاربة للحصول على الكفاءة والاستخدام العادل للموارد

- OS is a **control program (برنامج سيطرة)**
  - Controls execution of programs to prevent (منع) errors and improper (غير مناسب) use of the computer.

السيطرة على تنفيذ البرامج لمنح حدوث الاخطاء ومنع الاستخدام الغير المناسب للحاسبة

1.2 Computer System Components

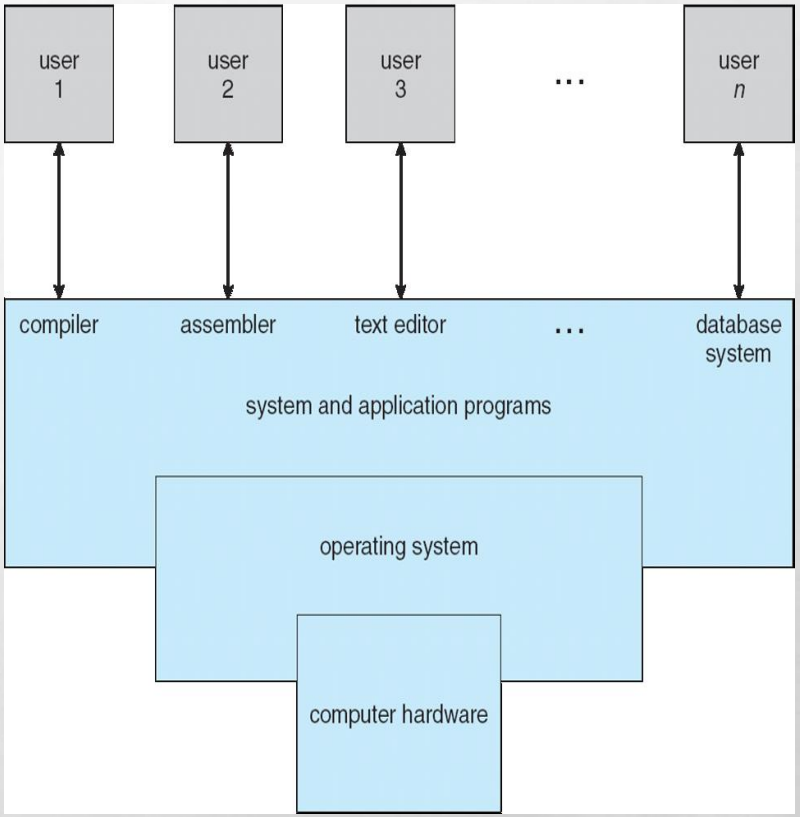
- Computer system can be divided into four components:

1- Hardware: (CPU, memory, I/O devices), provides basic computing resources

2- Operating system: Controls and coordinates (ينسق) use of hardware among various applications and users

3- Application programs: (Word processors, compilers, web browsers, database systems, video games

4- Users: People, machines, other computers



1.3. The Operating System Goals

1. The primary goal of an O.S. is to make the Computer System convenient (ملائم, مقنع) to use.

   الهدف الرئيسي لنظام التشغيل هو وضع نظام الحاسبة بصيغة ملائمة ومقنعة للاستخدام

2. A secondary goal is to use the computer H/W in an efficient manner (بطريقة فعالة).

   الهدف الثانوي هو استخدام المكونات المادية للحاسبة بطريقة فعالة

1.4 The operating Functions

A more common definition: **Operating system is the one program running at all times on the computer**, usually called the **kernel.**

تعريف اكثر شيوعاً: نظام التشغيل هو البرنامج الذي يعمل في كل الاوقات على الحاسبة وعادة ما يدعى بالنواة

O.S. performs many functions such as: يؤدي نظام التشغيل مجموعة من الدوال مثلا

1. Implementing the user interface. تمثيل واجهة المستخدم
2. Sharing H/W among users. مشاركة المكونات المادية بين المستخدمين
3. Allowing users to share data among themselves. السماح للمستخدمين بمشاركة البيانات بينهم
4. Preventing users from interfering with one another.منع التداخل بين المستخدمين
5. Scheduling resources among usersجدولة الموارد بين المستخدمين

1.4 The operating Functions Cont.

6. Facilitating I/O. تسهيل اجراء عمليات الادخال والاخراج

7. Recovering from errors.اصلاح الاخطاء

8. Accounting for resource usage. المحاسبة عن استخدام الموارد

9. Facilitating parallel operations.تسهيل العمليات المتوازية

10. Organizing data for secure and rapid access.تنظيم البيانات للوصول الامن والسريع

11. Handling network communications. التعامل مع اتصالات الشبكة

1.5 Operating System Categories

Classified into three groups, based on the nature of interaction between the computer and the user:

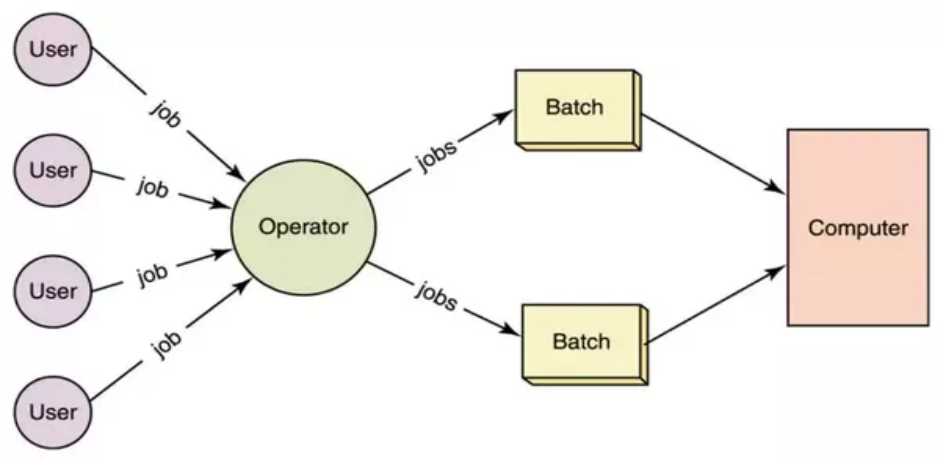
يصنف الى ثلاثة مجاميع اعتماداً على طبيعة التفاعل بين الحاسبة والمستخدم

1. **Batch System**

2. **Time-sharing System**

3. **Real Time System**

1.5.1 Batch System



- Users submit jobs on a regular (منتظم) schedule (e.g. daily, weekly, monthly) to a central place where the user of such system did not interact directly with C/S.

يرسل المستخدمون العمل وفقًا لجدول منتظم (على سبيل المثال يوميًا ، أسبوعيًا ، شهريًا) إلى مكان مركزي لا يتفاعل فيه مستخدم هذا النظام بشكل مباشر مع نظام الحاسبة.

- To speed up processing, jobs with similar needs were batched together and were run through the computer as a group.
  Thus, the programmers would leave their programs with the operator.

لتسريع المعالجة ، تم تجميع العمل ذو الاحتياجات المتشابهة معًا وتم تشغيلها من خلال الكمبيوتر كمجموعة. وبالتالي ، سيترك المبرمجون برامجهم مع المشغل.

1.5.1 Batch System Cont.

- The output from each job would be send back to the appropriate programmer.
- يتم إرسال الإخراج من كل عمل إلى المبرمج المناسب

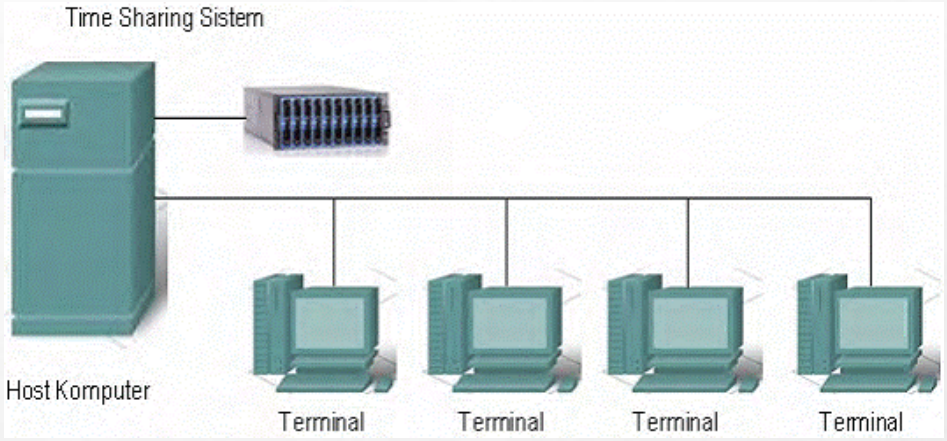
- The major task of this type was to transfer control automatically from one job to the next
- المهمة الرئيسية لهذا النوع هي نقل التحكم تلقائيًا من عمل إلى أخرى

Advantages: batch system is very simple.

Disadvantages: There is no direct interaction between the user and the job while that job is executing.

لا يوجد تفاعل مباشر بين المستخدم والعمل أثناء تنفيذ هذها العمل

1.5.2. Time sharing System



- Provides online communication between the user and the system

يوفر الاتصال المباشر بين المستخدم والنظام

- User gives instruction to the O.S. or to the program directly and receives an immediate response, therefore some time called an interactive system.

المستخدم يعطي الايعازات لـ O.S. أو إلى البرنامج مباشرة ويتلقى استجابة فورية ، لذلك يسمى في وقت ما بالنظام التفاعلي

- Allows many users simultaneously (في نفس الوقت) share the computer system where little CPU time is needed for each user.

يسمح للعديد من المستخدمين في نفس الوقت بمشاركة نظام الحاسبة حيث يحتاج كل مستخدم إلى وقت قليل من وحدة المعالجة المركزية

1.5.2. Time sharing System Cont.

- The system switches rapidly(بسرعة) from one user to the next user, This gives the impression (أنطباع) that they each have their own computer, while actually one C/S shared among the many users.

يقوم النظام بالتحويل بسرعة من مستخدم إلى آخر ، وهذا يعطي انطباعًا أن لكل منهما جهاز حاسبة خاص به ، بينما في الواقع يوجد نظام حاسبة واحد مشترك بين العديد من المستخدمين.

Advantages: Reduce the CPU idle time

Disadvantages: More Complex

1.5.3. Real Time System

- Used when there are rigid time requirements on the operation of a processor or the flow of data

  تستخدم عندما تكون هناك متطلبات زمنية صارمة على تشغيل المعالج أو تدفق البيانات

- A Real-time system guarantees that critical tasks complete on time

  انظمة الوقت الحقيقي تضمن أكمال المهمات الحرجة في الوقت المطلوب

- The Radar system is a good example for the real time system

1.6. Performance Development

- O.S. attempted to schedule computational activities to ensure good performance, where many facilities had been added to

نظام التشغيل يحاول جدولة الأنشطة الحسابية لكي يضمن الانجاز الجيد ,
حيث تم اضافة بعض التسهيلات اليه

O.S. some of these are:

1. **On-Line and off-Line operations**

2. **Buffering (التخزين المؤقت)**

   a- The single-buffered

   b- The Double-buffering

3- **Spooling**

1.6.1. On-Line and Off-Line Operations

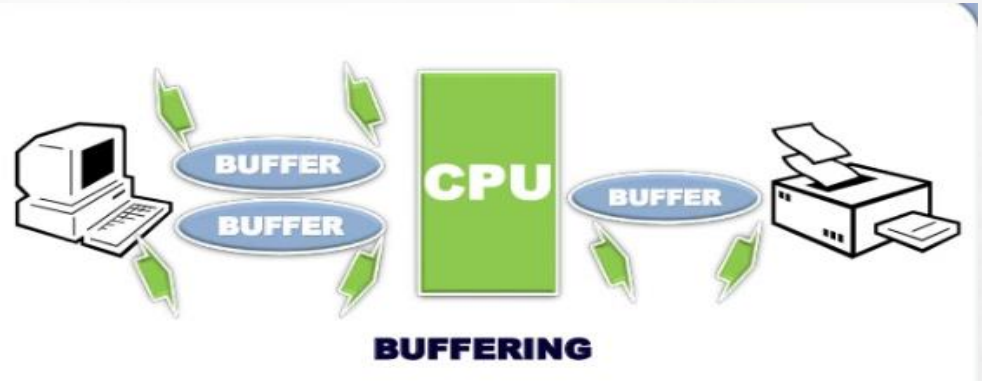
- A special subroutine was written for each I/O device called a device-driver.

برنامج فرعي خاص تم كتابته لكل I/O device يسمى بالـ device-driver

- Some peripherals (I/O devices) has been equipped (مجهزة) for:
  - either On-Line operation, in which they are connected to the processor.
  - or off-line operations in which they are run by control units not connected to the central computer system

بعض الـ I/O devices مجهزة بـ
- On-Line operation والتي تكون مرتبطة بالـ processor
- أو Off-line operations والتي تنفذ عن طريق الـ control units وتكون غير مرتبطة بالـ central computer system

1.6.2. Buffering



- **A buffer** is an area or primary storage for holding (يمسك) data during I/O transfers

الـ Buffer هو مساحة أو تخزين أساسي لحفظ بيانات أثناء عمليات نقل الإدخال / الإخراج

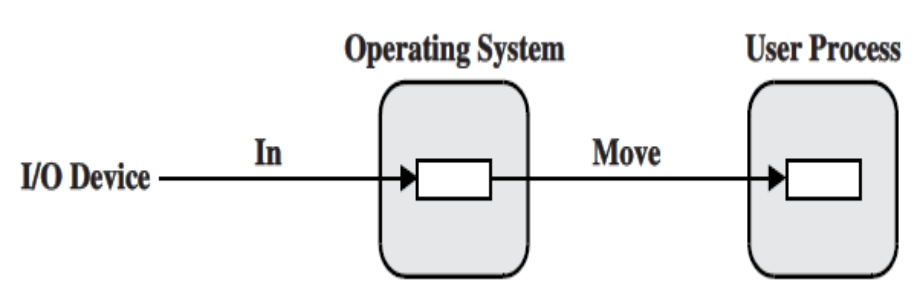
- On **input,** the data placed in the buffer by an I/O channel, when the transfer is complete the data may be accessed by the processor.

عند اجراء عمليات الإدخال ، يتم وضع البيانات في Buffer بواسطة قناة الإدخال / الإخراج ، وعند اكتمال النقل ، يمكن للمعالج الوصول إلى البيانات

There are two types of buffering:

1.6.2. Buffering Cont.

1. The single-buffered

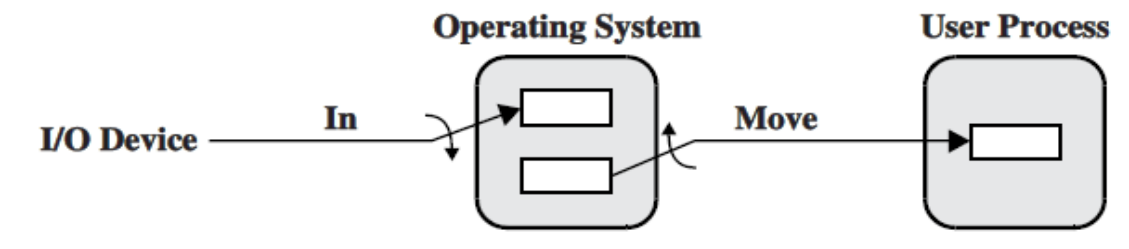


The channel deposits data in a buffer, the processor will accessed that data, the channel deposits the next data, etc. while the channel is depositing data, processing on that data may occur.

تقوم القناة بإيداع البيانات في الـ Buffer , الـ Processor يستطيع الوصول إلى تلك البيانات ، وتقوم القناة بإيداع البيانات التالية ، وما إلى ذلك أثناء قيام القناة بإيداع البيانات ، قد تحدث معالجة على تلك البيانات

1.6.2. Buffering Cont.

2. The Double Buffering



- This system allows overlap (تداخل) of I/O operations with processing

يسمح هذا النظام بالتداخل بين عمليات الإدخال / الإخراج والمعالجة

- While the channel is depositing data in one buffer the processor may be processing data in the other buffer

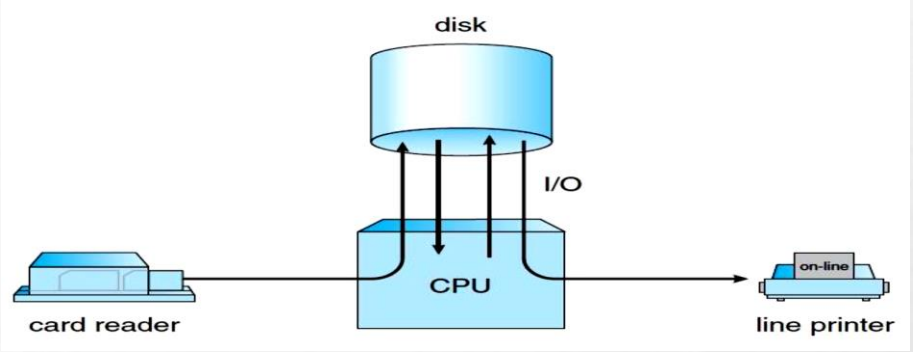
أثناء قيام القناة بإيداع البيانات في الـ Buffer الاول ، قد يقوم المعالج بمعالجة البيانات في الـ Buffer الأخر

- When the processor is finished processing data in one buffer it may process data in the second buffer

عند انتهاء المعالج من معالجة البيانات في الـ Buffer الاول ، قد يقوم بمعالجة البيانات في في الـ Buffer الأخر

- **In buffering the CPU and I/O are both busy.**

Spooling



- Spooling uses the disk as a very large buffer for:
  - Reading as input devices
  - And for storing output files until the output devices are able to accept them

يَستخدم الـ Spooling,القرص كمخزن مؤقت كبير جدًا من أجل:

- القراءة كأجهزة إدخال
- ولتخزين ملفات الإخراج حتى تصبح أجهزة الإخراج قادرة على قبولهم

- Spooling allows the computation of one job to overlap with the I/O of another jobs

يسمح الـ Spooling بالحسابات لعمل ما بالتداخل مع الإدخال / الإخراج لأعمال أخرى

- Therefore spooling can keep both CPU and the I/O devices working as much higher rates

لذلك يستطيع الـ Spooling ان يجعل كل من الـ CPU وأجهزة الإدخال / الإخراج في العمل بمعدلات عالية.

1.7. Multiprogramming

- Spooling provides an important data structure called a **job pool** kept on disk, the O.S. picks one job from the pool and begin to execute it.

يوفر الـ Spooling هيكلة بيانات مهمة تسمى **job pool** تحفظ على القرص ، الـ OS يختار عمل واحد من الـ **job pool** ويبدأ في تنفيذه.

- In **multiprogramming system:**
  - When the job may have to wait for any reason such as an I/O request, the O.S. simply switches to and executes another job.

في نظام الـ multiprogramming :

- عندما يضطر العمل إلى الانتظار لأي سبب مثل طلب الإدخال / الإخراج, الـ OS ببساطة ينتقل إلى عمل أخر وينفذه.

- When the second job needs to wait the CPU is switches to another job and so on.

عندما يحتاج العمل الثاني إلى الانتظار ، تنتقل وحدة المعالجة المركزية إلى عمل أخر وهكذا.

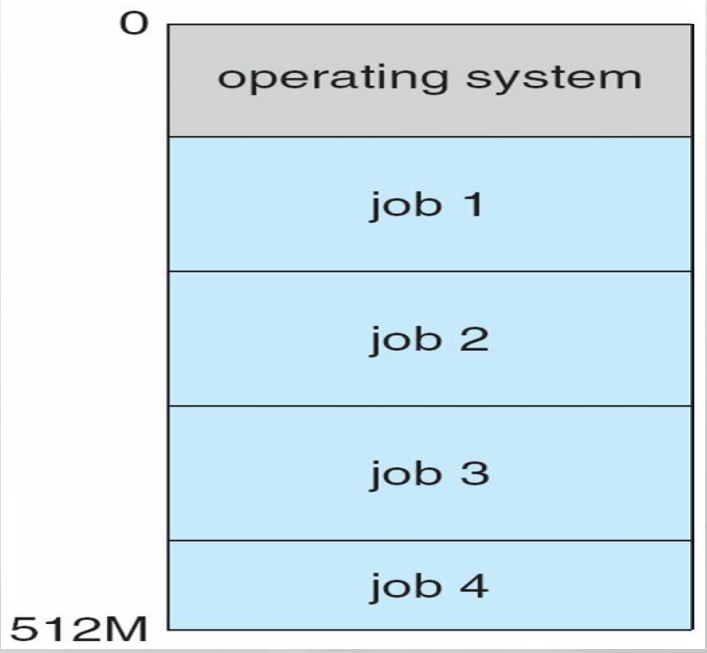
The CPU will never be idle

وبذلك لن تكون وحدة المعالجة المركزية خاملة أبدًا

1.7. Multiprogramming Cont.

- The figure shows the multiprogramming layout, where the O.S. keeps several jobs in **memory** at a time.
- This set of jobs is a subset of the jobs kept in the **job pool.**

يوضح الشكل تخطيط الـ the multiprogramming ، حيث يحتفظ الـ **OS** بالعديد من الاعمال في الذاكرة في وقت واحد.
هذه المجموعة من الاعمال هي مجموعة فرعية من الاعمال المحفوظة في **job pool**.



1.8. Parallel Systems

- Computers today, are **multiprocessors system**, also known as **Parallel systems,** these multiprocessors sharing the computer Bus, the clock, and sometimes memory and peripheral devices

أجهزة الكمبيوتر اليوم هي نظام متعدد المعالجات ، يُعرف أيضًا باسم الأنظمة المتوازية ، وهذه المعالجات المتعددة تشترك في الBus ، والClock ، وأحيانًا الMemory والأجهزة الطرفية

The advantages:

- Increase the throughput .
- Save money compared to multiple single systems because the processors can share peripherals, cabinets, and power supplies.
- Increase reliability

- زيادة الإنتاجية.
- توفير المال مقارنة بالأنظمة الفردية المتعددة لأن المعالجات يمكنها مشاركة الأجهزة الطرفية والخزائن وإمدادات الطاقة.
- زيادة الموثوقية

Two types:

1. **Asymmetric Multiprocessing**
2. **Symmetric Multiprocessing**

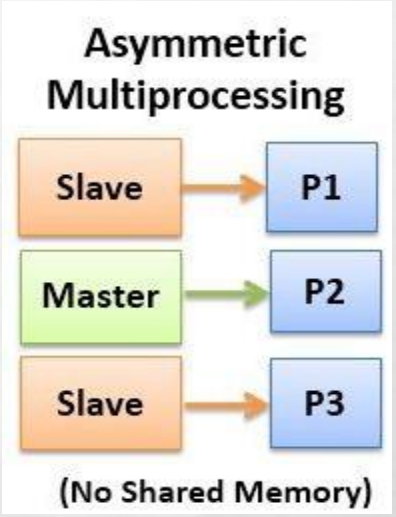
Multiprocessing Systems

1. Asymmetric Multiprocessing

المعالجة المتعددة غير المتماثلة

- Each processor is assigned a specific task.
- A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks.

- This scheme defines a Master-Slave relationship.
- The master processor schedules and allocates work to the slave processors.



يتم تعيين مهمة محددة لكل معالج.

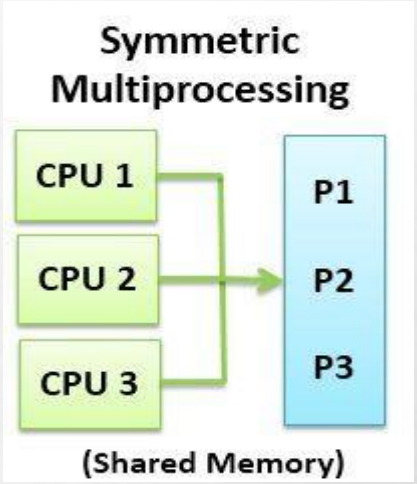
- معالج الرئيسي Master يتحكم في النظام ؛ تبحث المعالجات الأخرى إما عن المعالج الرئيسي للحصول على التعليمات أو لديها مهام محددة مسبقًا.

- يحدد هذا المخطط علاقة السيد والعبد Master-Slave relationship

- يقوم المعالج الرئيسي بجدولة العمل وتخصيصه لمعالجات الـ Slaves

Multiprocessing Systems

2. Symmetric Multiprocessing
المعالجة المتعددة المتماثلة

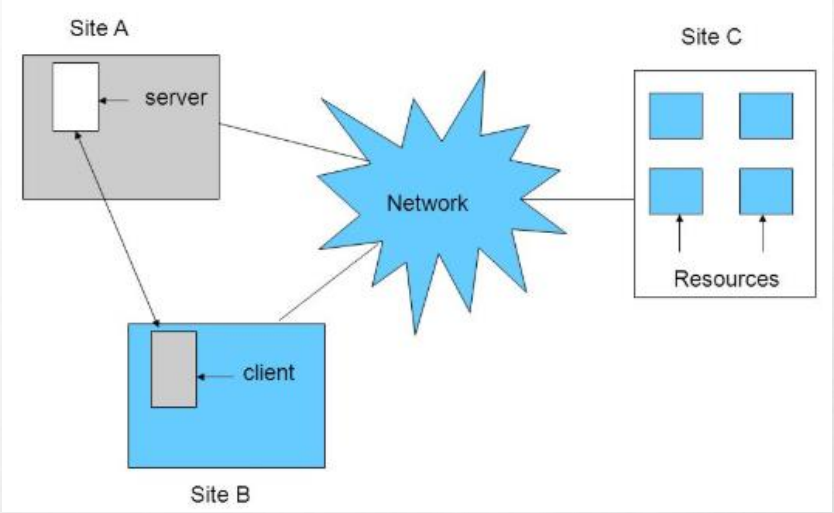


- Each processor performs all tasks, means that all processors are peers;
- No master-slave relationship exists between processors.

- يقوم كل معالج بأنجاز جميع المهام ، مما يعني أن جميع المعالجات هم أقران (متكافئة) .
- لا توجد علاقة master-slave relationship بين المعالجات.

1.8. Distributed Systems

- A resent C/S is to distribute computation among several processors. In Contrast to the parallel system, the processors do not share memory and clock.



أنظمة الحاسبات حالياً تقوم بتوزيع الحسابات بين عدة معالجات. على النقيض من النظام المتوازي ، لا تشترك المعالجات في الـ Memory او الـ clock

- The processors communicate with one another through various communication lines, such as high speed buses or telephone lines.

تتواصل المعالجات مع بعضها البعض عبر خطوط اتصال مختلفة ، مثل نواقل ذات سرعة عالية أو خطوط الهاتف.

1.8. Distributed Systems Cont.

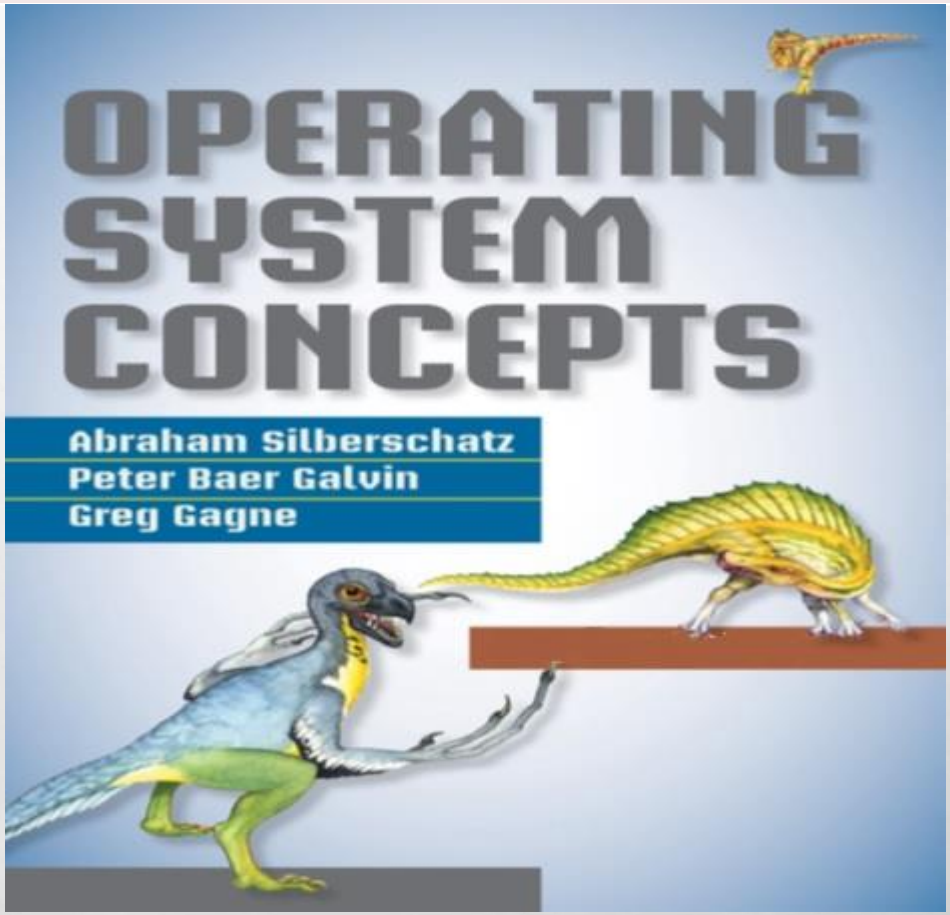
- The reasons for building distributed systems:

1. Resource sharing مشاركة الموارد
2. Computation speedup تسريع الحسابات
3. Concurrently Work العمل في نفس الوقت
4. Reliability الموثوقية

End of Chapter One

Mustansiriayah University
Collage of Education
Computers Science Department

Chapter Two



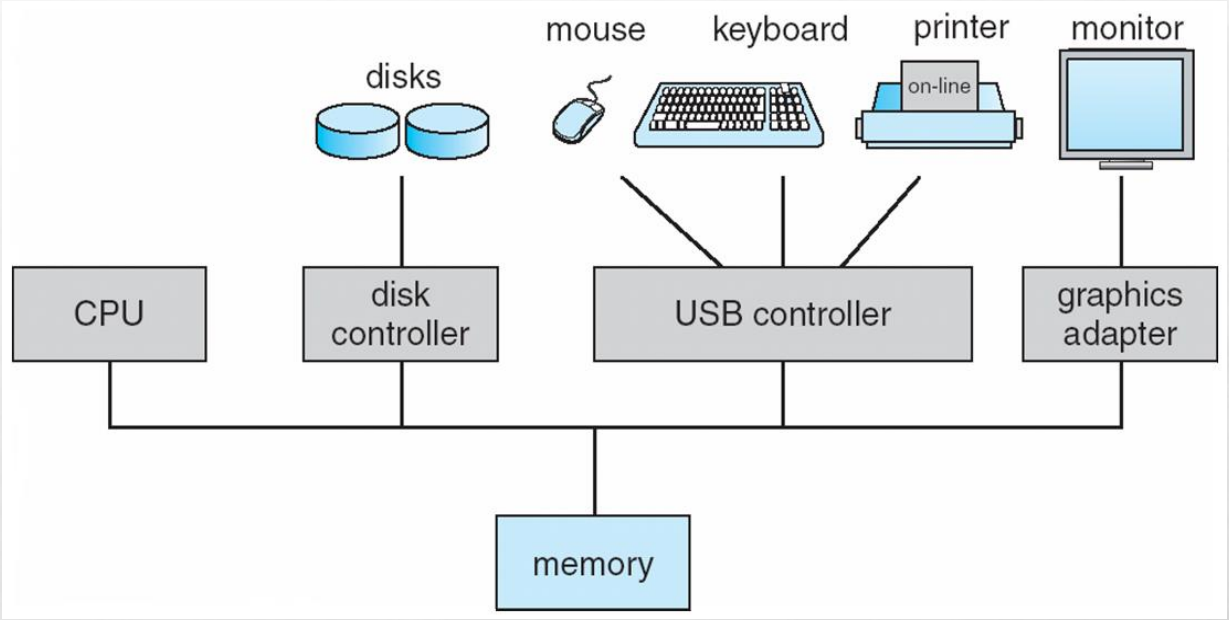
Fourth Class

Dr. Hesham Adnan ALABBASI

2022-2021

2.1 Computer System Operation
عمل نظام الحاسبة

- One or more CPUs, device controllers connect through common bus providing access to shared memory



- يتكون نظام الحاسبة من واحد أو أكثر من وحدات المعالجة المركزية ، وحدات التحكم متصلة من خلال خط أو قناة مشتركة توفير الوصول إلى الذاكرة

2.1 Computer System Operation Cont.

- Each device controller is in charge of a particular device type (for example, disk drives, audio devices, or video displays).
- The CPU and I/O devices can execute concurrently
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- Device controller informs CPU that it has finished its operation by causing an interrupt

- كل وحدة تحكم مسؤولة عن نوع جهاز معين (على سبيل المثال ، محركات الأقراص ، وأجهزة الصوت ، أو الشاشات).
- وحدة المعالجة المركزية وأجهزة الإدخال / الإخراج يمكن أن تنفذ (تعمل) في نفس الوقت.
- تحتوي كل وحدة تحكم في الجهاز على مخزن مؤقت خاص بها.
- تقوم وحدة المعالجة المركزية بنقل البيانات من / إلى الذاكرة الرئيسية من/ إلى المخازن المؤقتة الخاصة بها.
- تقوم وحدة تحكم الجهاز بإعلام وحدة المعالجة المركزية بأنتهاء عملها عن طريق التسبب في المقاطعة

Computer Startup

- When it is powered up or rebooted—it needs to have an **initial program** to run. عندما يتم تشغيل أو إعادة تشغيل الحاسبة يجب أن يكون لديه برنامج أولي للتنفيذ.
- **The initial program** or **bootstrap program** is
  1. Stored in ROM or EEPROM,
  2. Initializes (reset) all aspects (أجزاء) of the system from CPU registers to device controllers to memory contents.
  3. Loads operating system (kernel) and starts execution that program

البرنامج الأولي أو برنامج التهيئة

1. **يخزن هذا البرنامج في الـ ROM أو EPROM**
2. **إعادة تعيين جميع اجزاء نظام الحاسبة من سجلات وحدة المعالجة المركزية إلى وحدات تحكم الجهاز الى محتويات الذاكرة.**
3. **تحميل نظام التشغيل (النواة) ويبدأ تنفيذ هذا البرنامج**

- To accomplish this goal, the bootstrap program must locate the operating-system kernel and load it into memory.
- Once the kernel is loaded and executing, it can start providing services to the system and its users.

- لتحقيق هذا الهدف ، يجب على برنامج bootstrap تحديد موقع نواة نظام التشغيل وتحميله في الذاكرة.
- بمجرد تحميل النواة وتنفيذها ، يمكن أن تبدأ في تقديم الخدمات للنظام ومستخدميه.

2.2 I/O Interrupts

- The occurrence of an event is usually signaled by an **interrupt** from either the hardware or the software.

عادةً ما يتم الإشارة إلى وقوع حدث عن طريق مقاطعة Interrupt من المكونات المادية او البرامجيات وتكون على نوعين:

- **Hardware** may trigger an interrupt at any time by sending a signal to the CPU usually by way of the system bus.
  - المكونات المادية قد تؤدي الى المقاطعة في أي وقت عن طريق إرسال إشارة إلى وحدة المعالجة المركزية عادة عن طريق System bus.

- **Software** may trigger an interrupt by executing a special operation called a system call (also called a monitor call).
  - البرمجيات قد تؤدي إلى المقاطعة عن طريق تنفيذ عملية خاصة تسمى System call (تسمى أيضا Monitor call).

2.2 I/O Interrupts Cont.

- Interrupts are an important part of a computer architecture.
- الـ Interrupts هي جزء مهم من معمارية الحاسبة

- When the CPU is **interrupted**, it **stops** what it is doing and immediately **transfers** execution to a fixed location.
- The **fixed location** usually **contains** the starting address where the service routine for the interrupt is located.
- The interrupt service routine **executes**; on **completion**, the CPU **resumes** the interrupted computation.

- عندما يتم مقاطعة وحدة المعالجة المركزية ، فإنها تتوقف عن ما تقوم به وعلى الفور تنقل التنفيذ إلى موقع ثابت.
- يحتوي الموقع الثابت عادة على عنوان البداية حيث يوجد روتين الخدمة للمقاطعة.
- يتم تنفيذ روتين خدمة المقاطعة ؛ عند الانتهاء ، تستأنف وحدة المعالجة المركزية العمل المتقطع.

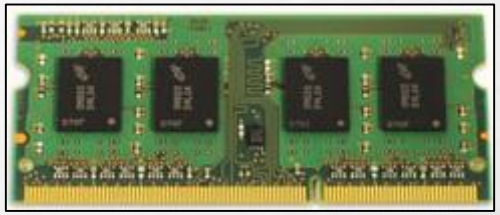
2.2 I/O Interrupts Cont.

- Interrupt transfers control to the interrupt service routine, through the interrupt vector, which contains the addresses of all the service routines
- المقاطعة تنقل التحكم إلى روتين خدمة المقاطعة ، من خلال متجه المقاطعة ، الذي يحتوي على عناوين جميع روتينات الخدمة
- This transfer would be to invoke a generic routine to examine the interrupt information. This routine, would call the interrupt-specific handler.
- سيكون هذا النقل هو استدعاء روتين عام لفحص معلومات المقاطعة. هذا الروتين ، سيدعو معالج المقاطعة المحدد.
- Interrupts must be handled quickly. Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed.
- يجب التعامل مع المقاطعات بسرعة. نظرا لأنه لا يمكن سوى عدد محدد مسبقا من المقاطعات ، يمكن استخدام جدول المؤشرات لمقاطعة الروتين بدلا من ذلك لتوفير السرعة اللازمة.
- The interrupt routine is called indirectly through the table, with no intermediate routine needed. Generally, the table of pointers is stored in low memory (the first hundred or so locations). These locations hold the addresses of the interrupt service routines for the various devices.
- يستدعى روتين المقاطعة بشكل غير مباشر من خلال الجدول، مع عدم الحاجة إلى روتين وسيط. عموما ، يتم تخزين جدول المؤشرات في ذاكرة منخفضة (أول مائة موقع أو نحو ذلك من المواقع). تحتوي هذه المواقع على عناوين إجراءات خدمة المقاطعة للأجهزة المختلفة.

2.2 I/O Interrupts Cont.

- This array, or interrupt vector, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.
- ثم يتم فهرسة هذه المصفوفة ، أو متجه المقاطعة ، من العناوين بواسطة رقم جهاز فريد ، معطى مع طلب المقاطعة ، لتوفير عنوان روتين خدمة المقاطعة لجهاز المقاطعة.
- The Interrupt architecture must save the address of the interrupted instruction.
- يجب أن تقوم معمارية المقاطعة بحفظ عنوان التعليمات التي تمت مقاطعتها.
- The operating system preserves the state of the CPU by storing registers and the program counter.
- يحافظ نظام التشغيل على حالة وحدة المعالجة المركزية عن طريق تخزين السجلات وعداد البرنامج.
- After the interrupt is serviced, the saved return address is loaded into the program counter, and the interrupted computation resumes as though the interrupt had not occurred.
- بعد انتهاء خدمة المقاطعة ، يتم تحميل عنوان الإرجاع المحفوظ في عداد البرنامج ، ويتستأنف المعالجة للعمل المتقطع كما لو أن المقاطعة لم تحدث.

2.3 Storage Structure



Main memory is the only large storage area that the CPU can access directly.

- The CPU can load instructions only from memory, so any programs must be in main memory (also called **Random-Access Memory** or **RAM**) to be executed.
- Implemented in a semiconductor technology called **dynamic random-access memory (DRAM).**

الذاكرة الرئيسية هي مساحة التخزين الكبيرة الوحيدة التي يمكن لوحدة المعالجة المركزية الوصول إليها مباشرة.

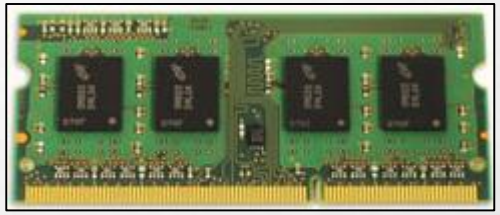
وحدة المعالجة المركزية يمكنها تحميل التعليمات فقط من الذاكرة ، لذلك يجب أن يكون أي برامج في الذاكرة الرئيسية (وتسمى أيضا ذاكرة الوصول العشوائي أو RAM) ليتم تنفيذها.

هذه الذاكرة مصنوعة من خلال تقنية أشباه الموصلات تسمى ذاكرة الوصول العشوائي الديناميكية (DRAM)

- **Read-only memory, ROM**), one of its types is **Electrically Erasable Programmable Read-Only Memory, EEPROM**). Because ROM cannot be changed, only static programs, such as the bootstrap program, are stored there.

- **ذاكرة للقراءة فقط ROM ، أحد أنواعه هو ذاكرة للقراءة فقط قابلة للمسح كهربائيا ، EEPROM).**
- **نظرا لأنه لا يمكن تغيير ROM، يتم تخزين البرامج الثابتة فقط فيها، مثل برنامج bootstrap.**

2.3 Storage Structure Cont.



- We want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons;

1. Main memory is usually too small to store all needed programs and data permanently.

2. Main memory is a **volatile** storage device that loses its contents when power is turned off or otherwise lost.

- نريد ان تكون البرامج والبيانات الموجودة في الذاكرة الرئيسية بشكل دائم. هذا الترتيب عادة غير ممكن للسببين التاليين;

1. الذاكرة الرئيسية عادة ما تكون صغيرة جدا لتخزين جميع البرامج والبيانات اللازمة بشكل دائم.2
2. الذاكرة الرئيسية هي جهاز تخزين متقلب يفقد محتوياته عند إيقاف تشغيل الطاقة أو فقدها.

2.3 Storage Structure Cont.

Thus, most computer systems provide:

- **Secondary storage** as extension of main memory that provides large nonvolatile storage capacity (Magnetic disk, CD-ROM (740 MB), DVD (4.7, 9 GB)).
- **Hard disks** is a rigid metal or glass platters covered with magnetic recording material

  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**.

  - The **disk controller** determines the logical interaction between the device and the computer

- **Solid-state disks** – faster than hard disks, nonvolatile

- Various technologies (Flash memory, personal digital assistants (PDAs).

Compact flash (CF) & secure digital (SD) cards, a Sony memory stick, and a USB memory key.

The main differences among the various storage systems are speed, cost, size, and volatility.

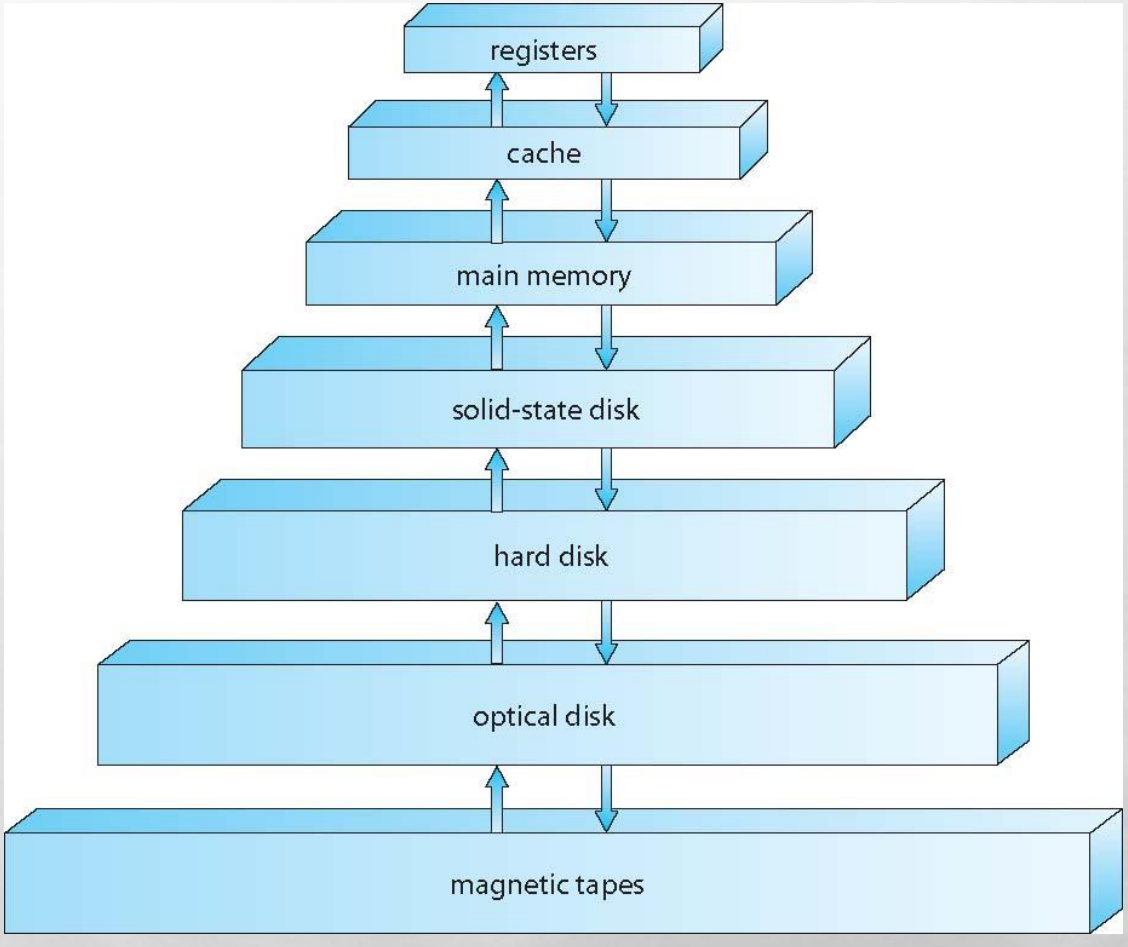
Storage-Device Hierarchy

- Storage systems organized in a hierarchy according to:

- Speed

- Cost

- Volatility



2.4 Hardware Protection

- To improve system utilization, the O.S began to share system resources among several programs simultaneously.

لتحسين استخدام النظام، بدأ نظلم التشغيل في مشاركة موارد النظام بين العديد من البرامج في وقت واحد.

- Multi programming put several programs in memory at the same time. This sharing created both **improved utilization** and **increased problems**.

البرمجة المتعددة تضع العديد من البرامج في الذاكرة في نفس الوقت .وقد أدى هذا الشيء إلى تحسين الاستخدام ولكن زيادة المشاكل.

- When the system was run **without** sharing an error in a program could cause problems for only the one program that was running. **With** sharing many processes could be affected by a bug in one program.

عندما يعمل نظام التشغيل من دون المشاركة فأن الخطأ في برنامج يمكن أن يسبب مشاكل فقط في نفس البرامج قيد التشغيل .مع المشاركة , العديد من الـ **processes** يمكن أن تتأثر بخطأ في برنامج واحد.

2.4.1 Dual Mode Operation

- To ensure proper operation, we must protect the O.S and all programs and their data from any malfunctioning program.

لضمان العمل السليم يجب حماية نظام التشغيل وجميع البرامج والبيانات الخاصة بهم من أي برنامج يسبب خلل في العمل.

- Protection is needed for any shared resource. The H/W support to differentiating among various modes of executions. Therefore we need two separate modes of operation:

هناك حاجة إلى حماية لأي مورد مشترك .الـ مكونات المادية صممت لكي تدعم التمييز بين مختلف أساليب التنفيذ . لذلك نحن بحاجة إلى وضعين منفصلين للتشغيل:

1- **User mode**

2- **Monitor mode** (also called **kernel mode, system mode, or privileged mode**).

- A bit called **Mode bit** is added to H/W to indicate (تشير) the current mode;

- **Monitor (0):** execution is done on behalf of the O.S يتم التنفيذ نيابة عن
- **User (1):** execution is done on behalf of the USER يتم التنفيذ نيابة عن

This protect the O.S from errant (المخطئين) users and errant users from one another

2.4.2. I/O Protection

- To prevent (نمنع) a user from performing illegal (غير القانونية) I/O:

-We define all I/O instructions to be privileged instructions.

-Thus user cannot issue I/O instructions directly, they must do it through the O.S

يجب ان نُعرف ان جميع ايعازات I/O هي ايعازات مميزة.
وبالتالي لا يمكن للمستخدم إصدار ايعازات I /O بصورة مباشرة وانما يجب عليه القيام بذلك من خلال O.S

- For I/O protection to be complete:

لكي تكتمل حماية الـ I/O

* We must be sure that a user program can never gain control of the computer in monitor mode.

يجب أن نكون متأكدين من أن برنامج المستخدم لا يمكن أبدا ان يحصل على سيطرة الحاسبة في الـ Monitor mode

2.4.3. Memory Protection

- To ensure correct operation: we must protect the **interrupt service routines** in the O.S from modification.

لضمان التشغيل الصحيح :يجب حماية interrupt service routines في O.S. من التعديل.

1- We must protect the **interrupt vector** from modification by a user program.

يجب علينا حماية متجه المقاطعة من التعديل من قبل المستخدم البرنامج

2- Also we must protect the **interrupt service routines** in the O.S from modification.

أيضا يجب علينا حماية إجراءات خدمة المقاطعة في O. S من التعديل

- What we need to separate each program's memory space, the ability to determine the range of legal addresses that the program may access, and to protect the memory outside that space.

نحتاج الى فصل مساحة ذاكرة كل برنامج، والقدرة على تحديد نطاق العناوين التي قد يصل إليها البرنامج، وحماية الذاكرة خارج تلك المساحة.

Memory Protection Cont.

- This protection can provide by using two registers usually a **base** and a **limit**

- The **base** register holds the smallest legal physical memory address

يحمل أصغر عنوان الذاكرة فعلي قانوني

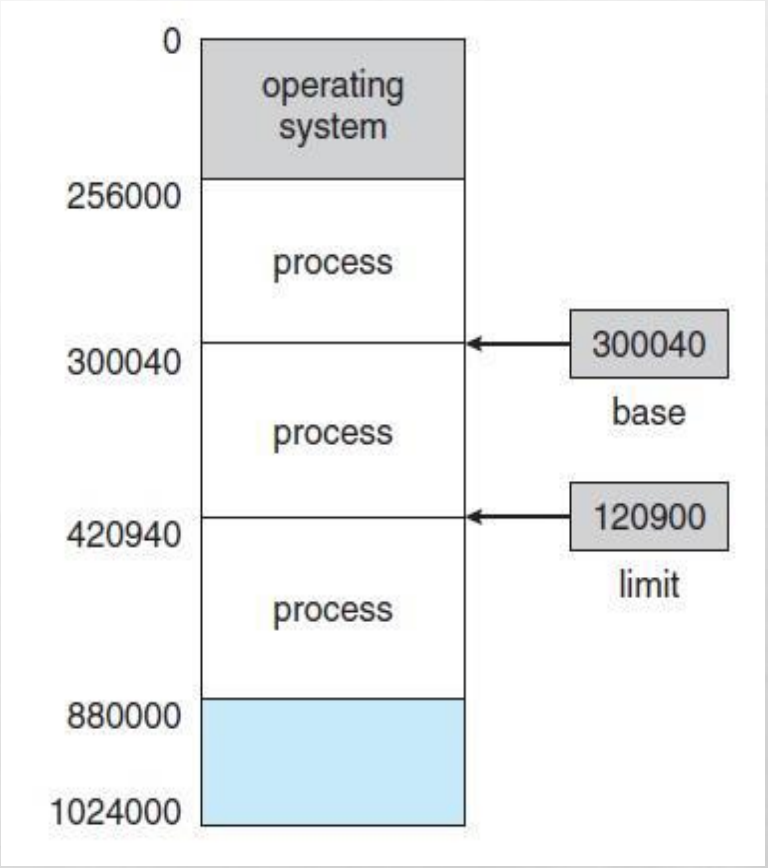
- The **limit** register contains the size of the range

يحتوي على حجم المدى المسموح

- Example:

Base register is 300040

Limit register is 120900



Then the program can legally access all addresses from 300040 through 420940 (base + limit).

اذن يمكن للبرنامج الوصول بشكل قانوني إلى جميع العناوين من 300040 إلى 420940

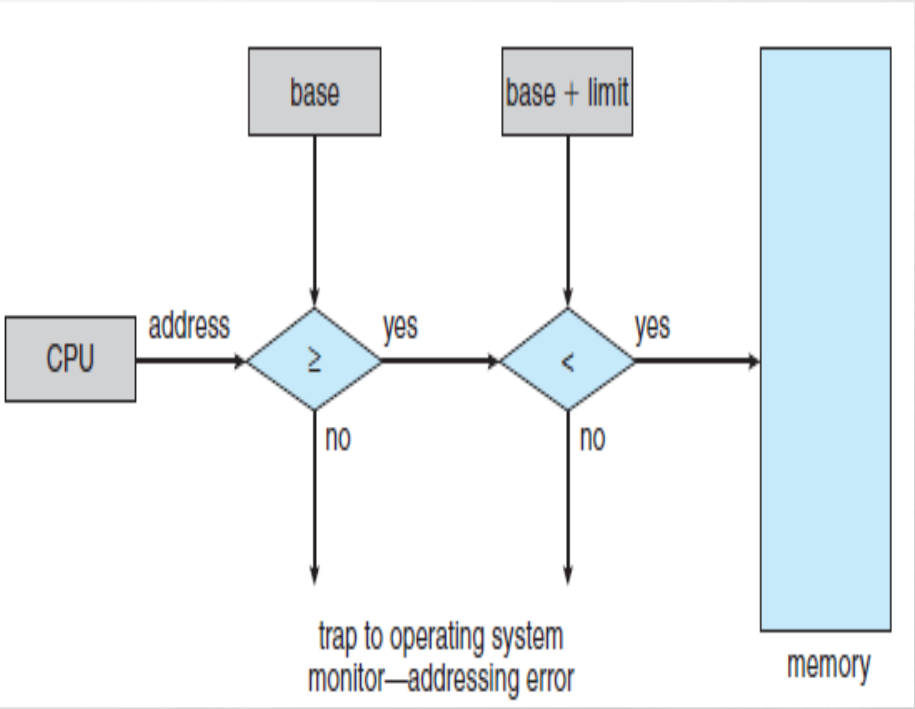
Memory Protection Cont.

- The CPU comparing (تقارن) every address generated in user mode with registers accomplishes this protection

وحدة المعالجة المركزية تقارن كل عنوان تم إنشاؤه في وضع المستخدم مع السجلات لتحقيق هذه الحماية

- Any attempt by a program executing in user mode to access monitor memory or other user's memory results in a trap to the monitor which treats the attempt as a fatal error

أي محاولة من قبل برنامج ينفذ في الـ User mode للوصول إلى الذاكرة لخاصة بالـ monitor (سيطرة النظام) أو ذاكرة المستخدمين الآخرين يؤدي إلى اصدار مقاطعة (trap) إلى الـ monitor والتي تعامل هذه المحاولة كخطأ فادح



- This scheme prevents the user program from modifying the code or data structures of either the O.S or other users.

تمنع هذا الصيغة برنامج المستخدم من تعديل ال code او data structures لكل من O.S أو المستخدمين الآخرين.

2.4.4. CPU Protection

- The third piece of the protection is ensuring that the O.S maintains control

الجزء الثالث من الحماية هو ضمان أن يحافظ الـ O.S على السيطرة

- We must prevent a user program from an infinite loop, and never returning control to the O.S

يجب منع برنامج المستخدم من الدخول في infinite loop و لا يرجع السيطرة الى O.S

- **To achieve this goal** we can use **a timer,** a timer can be set to interrupt the computer after a specified period. The period may be fixed (1/60 second) or variable (from 1 millisecond to 1 second).

يمكن تعيين timer لمقاطعة الحاسبة بعد فترة محددة .قد تكون الفترة ثابتة 1/60 ثانية أو متغيرة من 1مللي ثانية إلى ثانية واحدة

- **To control the timer:** The O.S sets the counter, according to fixed-rate clock. Every time that the clock ticks the counter is decremented. When the counter reaches (0) an interrupt occurs, and control transfers automatically to the O.S, which may treat the interrupt as a fatal error or may give the program more time.

الـO.S يضع قيمة محدد في counter وفقا لمعدل clock ثابت .في كل مرة يحدث ال clock يتم تخفيض ال counter وعندما يصل الى (0) يحدث الـ interupt ، ويتم نقل التحكم تلقائياً إلى O.S، والذي قد يعامل المقاطعة كخطأ فادح أو قد يعطي البرنامج المزيد من الوقت.

2.5. System Calls

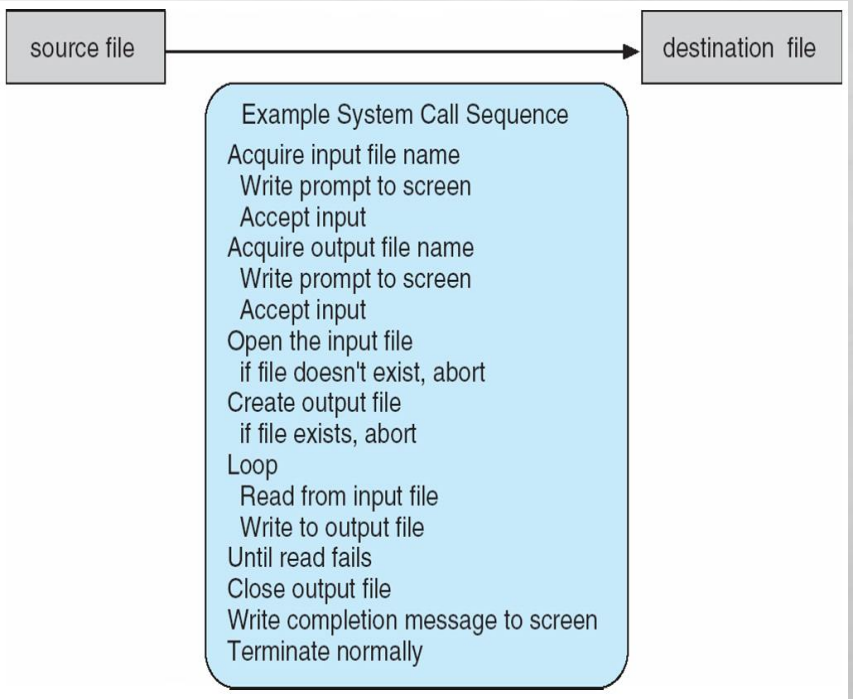
- **System calls** provide an interface to the services made available by an operating system توفر واجهة للخدمات التي يوفرها نظام التشغيل

These calls are generally available as:

- Routines written in **C and C++,**
- Although certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using **assembly-language instructions**

- From the example you can see
- Even simple programs may make heavy use of the operating system.
- Systems execute thousands of system calls per second.

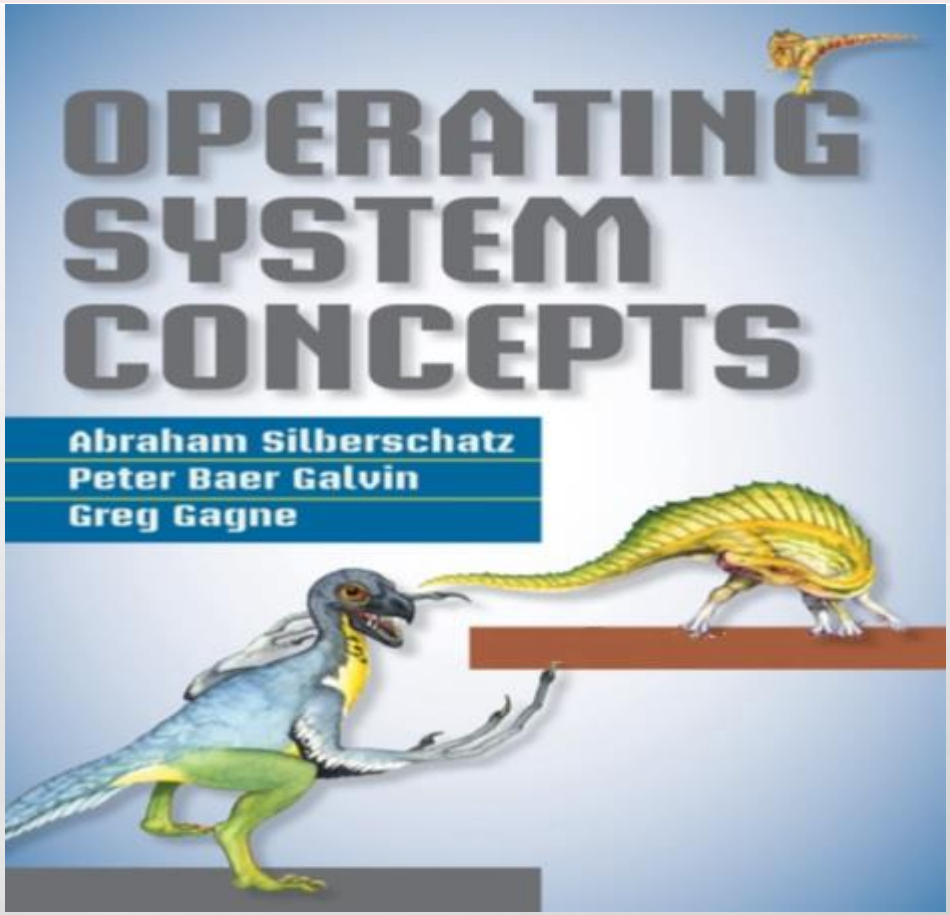
- حتى البرامج البسيطة قد تستخدم نظام التشغيل بصورة كبيرة .
- تقوم الأنظمة بتنفيذ آلاف منها في الثانية.



End of Chapter 2

Mustansiriayah University
Collage of Education
Computers Science Department

Chapter Three
Processes



Fourth Class

Dr. Hesham Adnan ALABBASI

2021-2022

3.1- 3.2 Process Management- Process Concepts

- A process is a program in execution; process execution must progress in sequential fashion.

الـ Process هو برنامج في التنفيذ .يجب أن يكون تقدم تنفيذ الـ Process بطريقة متتابعة.

- A process need certain resources such as CPU time, memory, files, and I/O devices to accomplish its task. These resources are allocated to the process either when it is created or while it is executing.

تحتاج ال Process إلى موارد معينة مثل CPU time, memory, files, and I/O devices لإنجاز مهمتها .يتم تخصيص هذه الموارد للـ Process إما عند إنشائها أو أثناء تنفيذها .

- Early C/S allowed only one program to be executed at a time. This program had complete control of the system and had access to all of the system resources.
- Today C/S allows multiple programs to be executed concurrently, therefore consists of a collection of processes.

- أنظمة الحاسبات الاولى كانت تسمح بتنفيذ برنامج واحد فقط في وقت واحد .وكان هذا البرنامج يسيطر بصورة كاملة على النظام ويستطيع الوصول إلى جميع موارد النظام .
- أنظمة الحاسبات اليوم تسمح بتنفيذ عدة برامج في نفس الوقت ، وبالتالي يحتوي النظام على مجموعة من الـ processes

3.3 Process State

- As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:

أثناء تنفيذ الـ Process ، تُغير حالتها. يتم تعريف حالة الـ Process جزئيًا من خلال النشاط الحالي لتلك الـ Process. كل Process ممكن ان يكون في أحدي الحالات التالية:

- **New**: The process is being created
- **Running**: Instructions are being executed
- **Waiting**: The process is waiting for some event to occur
- **Ready**: The process is waiting to be assigned to a processor
- **Terminated**: The process has finished execution

- These names are arbitrary, and they vary across operating systems.
- The states that they represent are found on all systems.
- It is important to realize that only one process can be running on any processor at any instant.
- Many processes may be ready and waiting, however. The state diagram corresponding to these states is presented in Figure 3.1.

من المهم أن ندرك أن Process واحد فقط يمكن تشغيلها على أي معالج في أي لحظة.

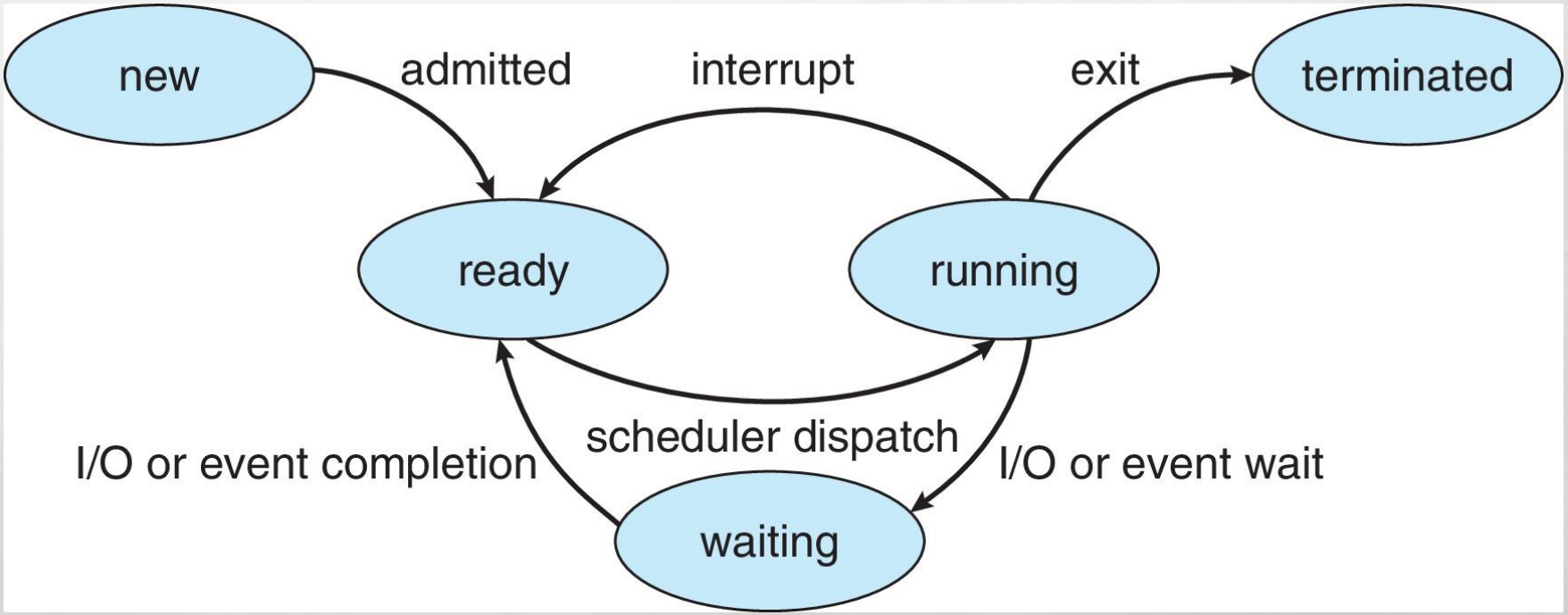


Figure 3.1 Diagram of Process State

3.4 Process Control Block (PCB)

❖ Each process is represented in the operating system by a **Process Control Block** (**PCB**), also called a **Task Control Block**.

يتم تمثيل كل Process في نظام التشغيل بواسطة Process Control Block (PCB)، وتسمى أيضا Task Control Block.

❖ It contains many pieces of Information associated with each process, including these:

يحتوي على العديد من المعلومات المرتبطة مع كل Process ، بما في ذلك:

- **Process state:** running, waiting, etc.
- **Program counter:** the address of the next instruction to execute
- **CPU registers:** like, accumulators, index registers, stack pointers
- **CPU scheduling information**: priorities, scheduling queue pointers
- **Memory-management information:** the value of the base and limit registers, the page tables
- **Accounting information** : CPU used, clock time, start, time limits
- **I/O status information**: list of I/O devices allocated to process, list of open files

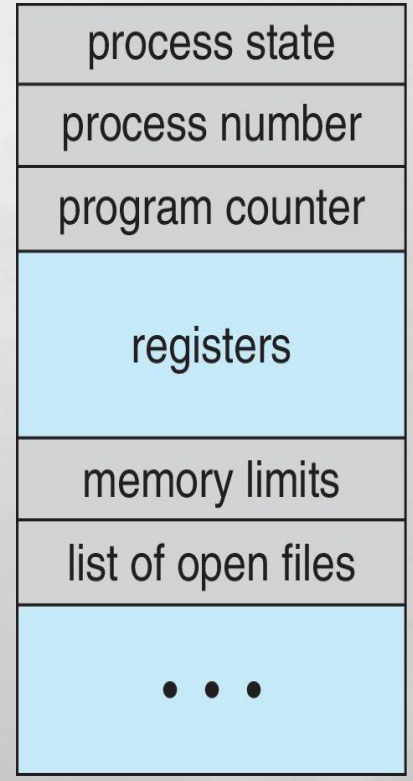


Figure 3.2 Process control block (PCB)

3.5 Process Scheduling

- The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization

الهدف من البرمجة المتعددة هو أن يكون بعض الـ Process قيد التنفيذ في جميع الأوقات ، لتحقيق أقصى قدر من استخدام الـ CPU

- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.

الهدف من الـ time sharing هو تبديل CPU بين الـ Processes بشكل متكرر بحيث يمكن للمستخدمين التفاعل مع كل برنامج أثناء تشغيله.

- **The process scheduler** selects an available process (possibly from a set of several available processes) for program execution on the CPU.

الـ **process scheduler** يختار Process متوفر (ربما من مجموعة من ال Processes المتوفرة) لتنفيذ البرنامج في الـ CPU

- For a single-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

- بالنسبة لنظام ذو معالج واحد ، لن يكون هناك أكثر من تنفيذ لـ Process واحدة. إذا كان هناك المزيد من الـ Processes ، فسيتعين على الباقي الانتظار حتى تكون وحدة المعالجة المركزية غير مشغولة (فارغة) ويمكن إعادة جدولتها.

3.5 Process Scheduling Cont.

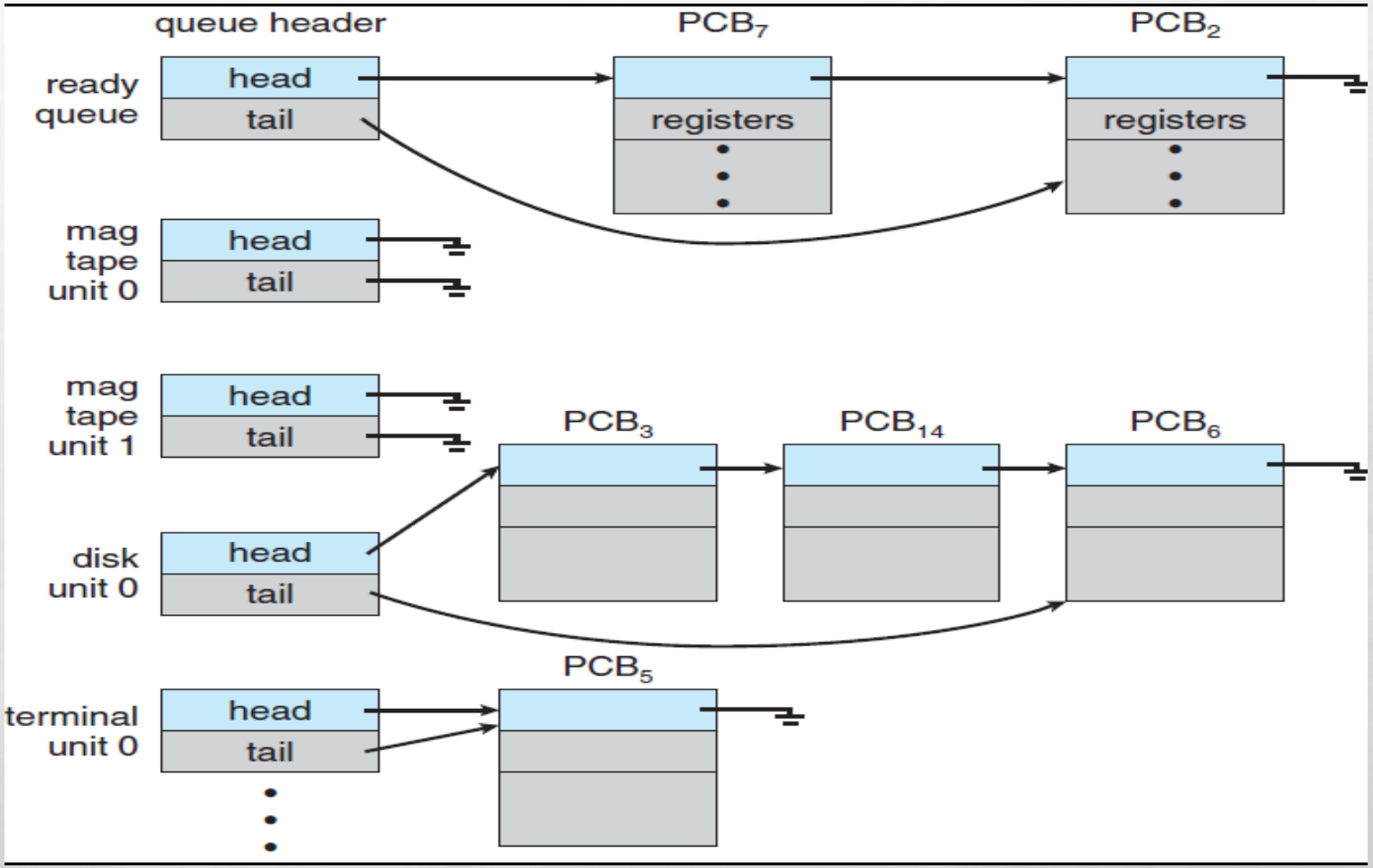
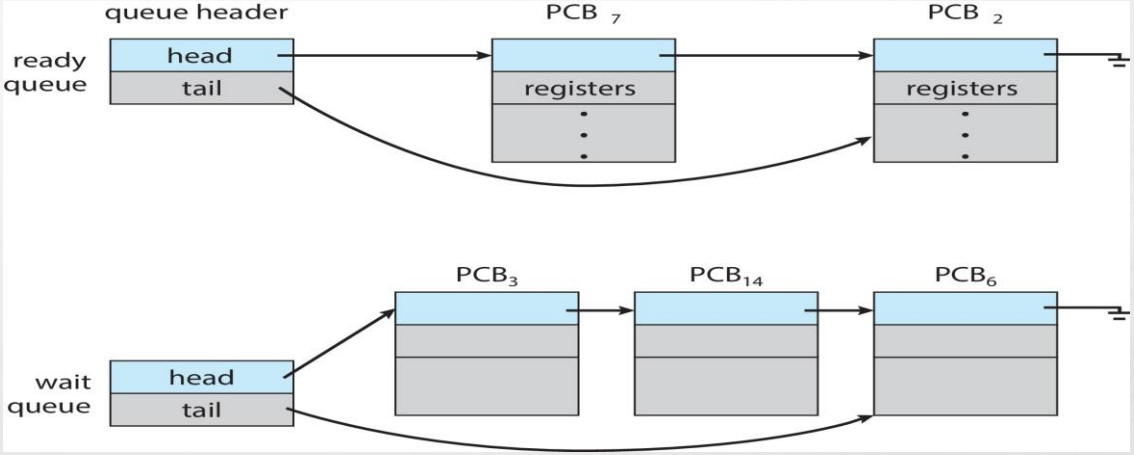


Figure 3.3 The ready queue and various I/O device queues

3.6 Scheduling Queues

- As processes enter the system, They are put into a **job queue**, which consists of all processes in the system

عندما تدخل الـ Processes الى النظام ، يتم وضعها في قائمة انتظار الوظائف ، والتي تتكون من جميع الـ Processes الموجودة في النظام.



- **Ready queue:** set of all processes residing in main memory, ready and waiting to execute.

- مجموعة من كل الـ Processes الباقية في الذاكرة الرئيسية وتكون جاهزة وتنتظر لكي تُنفيذ

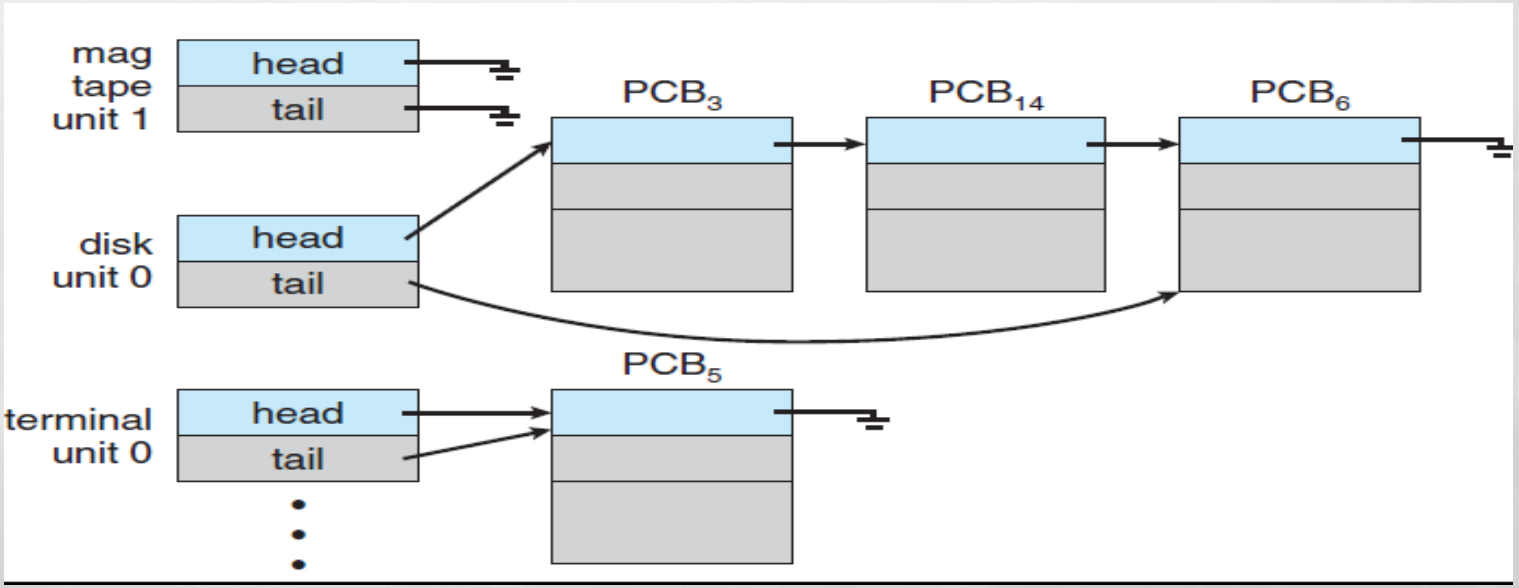
- **Wait queues:** set of processes waiting for an event (i.e. I/O)

مجموعة من الـ Processes التي تنتظر لحدث قراءة/كتابة

Scheduling Queues Cont.

- There are also other queues in the system. Such as list of processes, waiting for a particular I/O device is called a **device queue**.

- هنالك انواع اخرى من الـ queues أخرى في النظام. مثل قائمة الـ Processes التي هي في انتظار جهاز I/O معين يسمى **device queue**.



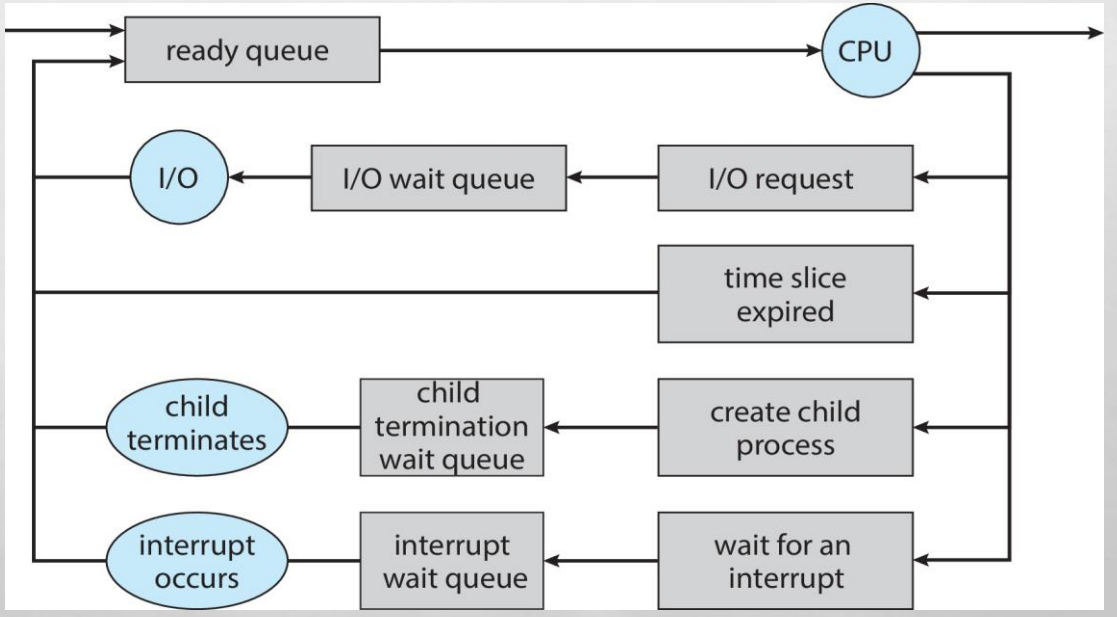
Scheduling Queues Cont.

- A new process is initially put in the **ready queue**. It waits there until it is selected for execution, or is **dispatched.**

الـ process الجديدة في البداية يتم وضعها في **ready queue**. تبقى ينتظر هناك حتى يتم تحديدها للتنفيذ ، أو يتم إرساله.

- Once the process is allocated the CPU start execute it, one of several events could occur:

- بمجرد تخصيص الـ process تبدأ وحدة المعالجة المركزية في تنفيذها ، يمكن أن يحدث أحد الأحداث التالية:



Scheduling Queues Cont.

a. The process could issue an **I/O request** and then be placed in an **I/O queue**.

b. The process could create a new sub-process and wait for the sub-process's termination.

c. The process could be removed from the CPU, as a result of an interrupt, and be put back in the ready queue.

أ. يمكن أن تصدر الـ process طلب الإدخال / الإخراج ثم توضع في **I/O queue**

ب. يمكن للprocess تخلق sub-process جديدة وتنتظرها الى ان ينتهي تنفيذها.

ج. يمكن إزالة الـ process من وحدة المعالجة المركزية ، نتيجة للمقاطعة ، وإعادة وضعها في ready queue.

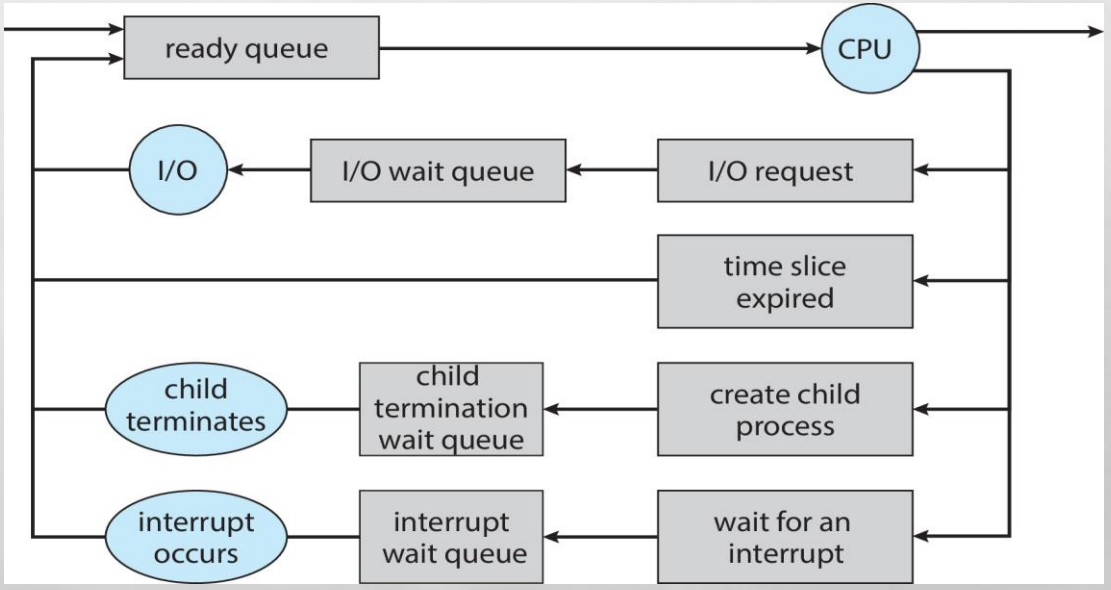


Figure 3.4 Queuing-diagram representation of process scheduling

3.7 Scheduling Levels

- A process migrates (يهاجر) among the various scheduling queues throughout its life time. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate **scheduler.**

- **يجب أن يختار نظام التشغيل، لأغراض الجدولة، الـ processes من الـ queues بطريقة ما..**
- **يتم تنفيذ عملية الاختيار من قبل المجدول المناسب.**

Types of scheduler

1. **Long-term scheduler** (or **job scheduler**) (**L.T.S.**):

- selects which processes should be brought into the ready queue .
- Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
- The long-term scheduler controls the **degree of multiprogramming**

- **أختيار الـ processes التي يجب إحضارها إلى الـ ready queue.**
- **يتم استدعاء الـ .L.T.S بشكل غير متكرر (ثواني دقائق) وبالتالي قد يكون بطيئاً.**
- **يتحكم الـ .L.T.S في درجة البرمجة المتعددة.**

- Processes can be described as either:

a. **I/O-bound process** – spends more time doing I/O than computations.

b. **CPU-bound process** – spends more time doing computations.

- **يمكن وصف الـ processes بأنها إما :**
- **أ- I/O-bound process تقضي وقت في إجراء I/O اكثر من الحسابات.**
- **ب- CPU-bound process تقضي وقت أكثر في القيام بالحسابات.**

Scheduling Levels Cont.

2. Short-term scheduler (or CPU scheduler) (S.T.S)

- selects which process should be executed next and allocates CPU .
  - أختيار أي من الـ process يجب تنفيذها بعد ذلك وتخصيص الـ CPU لها.

- Sometimes the only scheduler in a system.
  - في بعض الأحيان يكون هذا النوع من المجدول هو الوحيد في النظام

- Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast).
  - يتم استدعاء الS.T.D بشكل متكرر (milliseconds) وبالتالي يجب ان يكون سريع

- If all processes are I/O bound the ready queue will almost be empty and the S.T.S will have little to do.
  - إذا كانت كل الـ processes هي من نوع I/O bound فأن الـ ready queue سيكون فارغ تقريبًا ولن يكون لدى S.T.S الكثير للقيام به .

- If all processes are CPU-bound the waiting queue will almost be empty
  - إذا كانت كل الـ processes هي من نوع CPU–bound فأن الـ waiting queue سيكون فارغ .

Scheduling Levels Cont.

3. The Medium-Term Scheduler (M.T.S)

- Some O.S such as time-sharing systems may introduce an additional intermediate level of scheduling.
  - قد تقدم بعض أنظمة O.S التشغيل مثل time–sharing مستوى متوسط إضافي من الجدولة .

- The key behind the M.T.S is that sometimes it can be advantageous to remove processes from memory and thus to reduce the degree of multiprogramming.
  - المفتاح وراء M.T.S هو أنه في بعض الأحيان يمكن أن يكون من المفيد إزالة الـ processes من الذاكرة وبالتالي تقليل درجة الـ multiprogramming.

- The process can be swapped out and swapped in later by the M.T.S swapping may be necessary to improve the process mix.
  - يمكن عمل swapped out and swapped in لـ Process في وقت لاحق من قبل M.T.S لانها تكون ضرورية لتحسين مزيج لل Process

Scheduling Levels Cont.

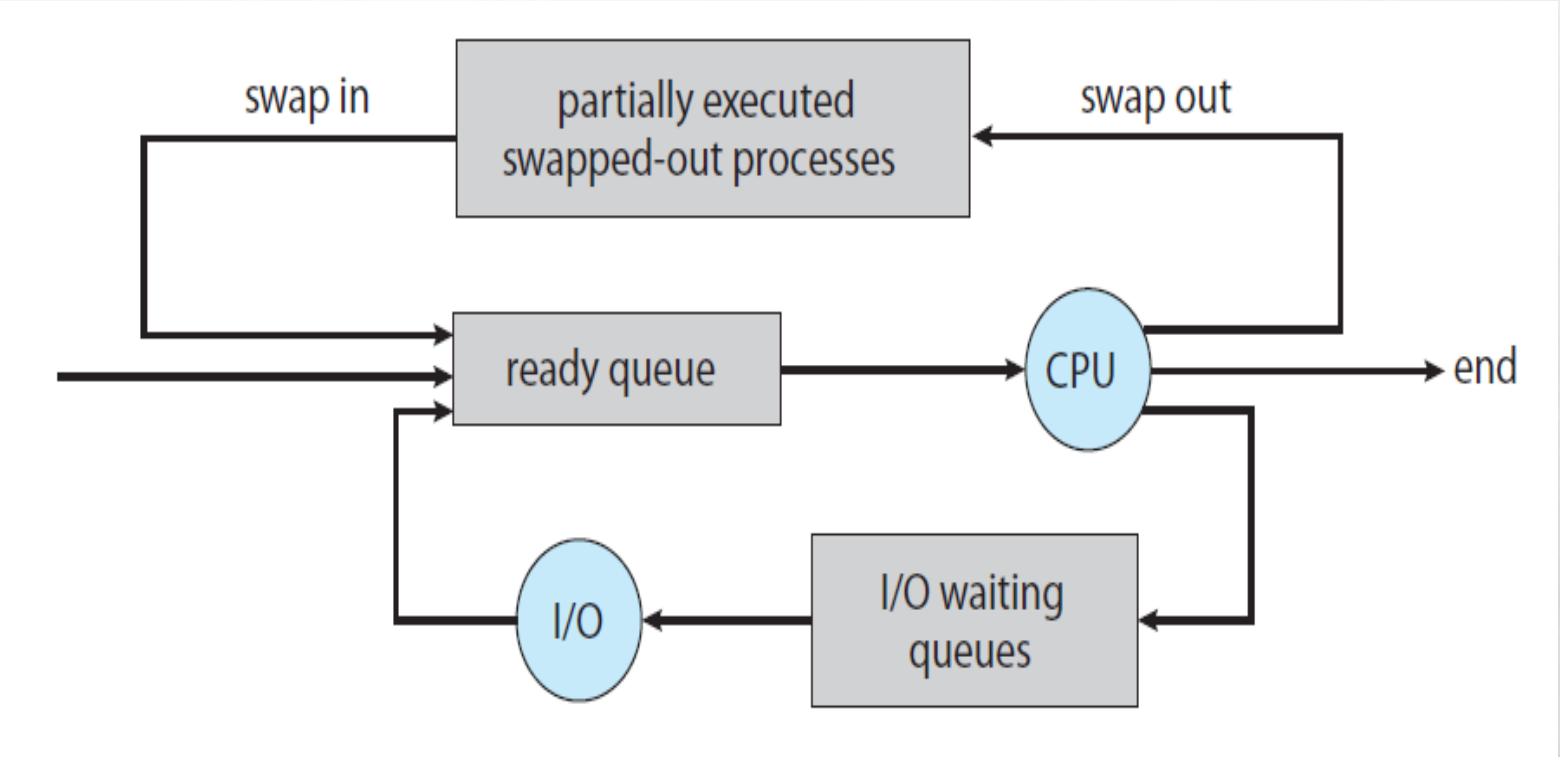


Figure 3.5 Addition of medium-term scheduling to the queueing diagram.

3.8 Context Switch

A **context switch** occurs when the CPU switches from one process to another.

- يحدث **context switch** عند تبديل الـ **CPU** بين **Process** وآخر.

- Switching the CPU to another process requires performing the system to **save the state** of the current process and a load the **saved state** of the new process via a **context switch**.

- تبديل الـ **CPU** الى **Process** آخر يتطلب أنجاز النظام الى حفظ حالة ال **Process** الحالية وتحميل الحالة المحفوظة لل **Process** الجديدة عبر **context switch**

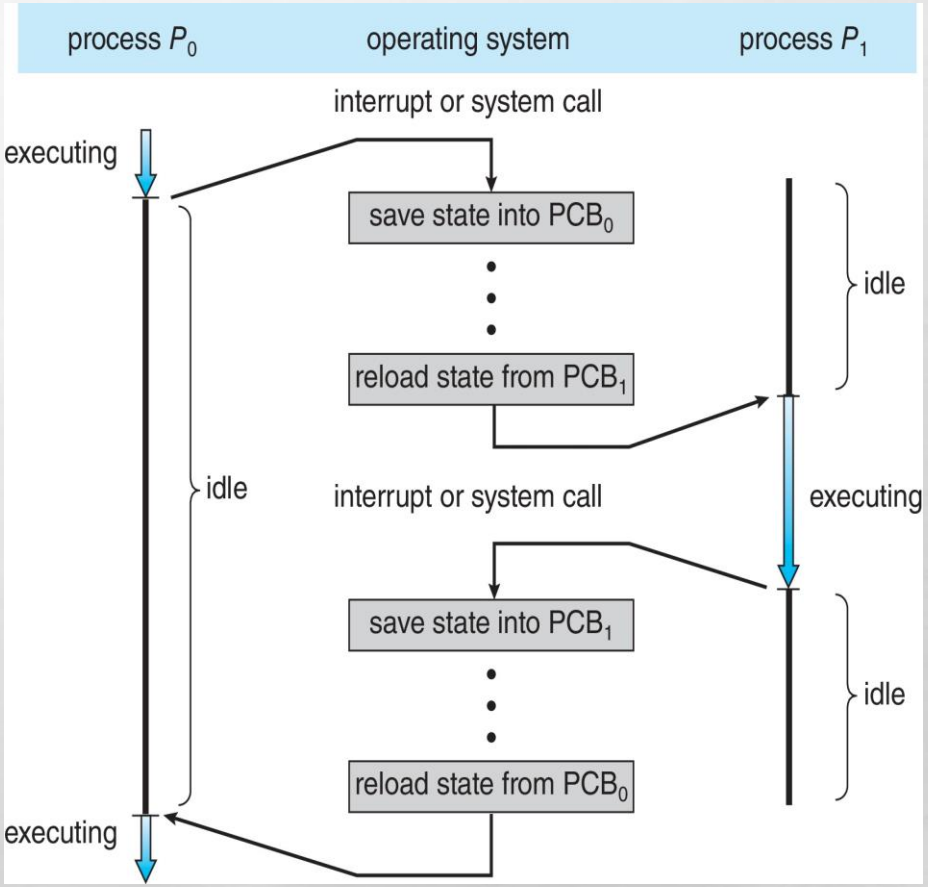


Figure 3.6 Diagram showing CPU switches from process to process.

Context Switch Cont.

- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

  - عند حدوث **context switch**، يحفظ الـ **kernel** سياق الـ **process** القديمة في **PCB** الخاصة بها ويحمّل الـ **context** المحفوظة لـ **process** الجديدة المجدولة للتشغيل.

- Context-switch speed varies from machine to machine, depending on the **memory speed**, **the number of registers** that must be copied, and the **existence of special instructions** (such as a single instruction to **load** or **store** all registers).

  - تختلف سرعة **context switch** من جهاز إلى آخر، اعتمادًا على سرعة الذاكرة وعدد السجلات التي يجب نسخها ووجود ايعازات خاصة (مثل ايعاز واحد لتحميل جميع السجلات أو تخزينها).

3.9 Operations on Processes

- O.S that mange processes must be able to perform certain operation on and with processes.

These include: **create**, **destroy**, **suspend**, **resume**, **change a process priority**, **block a process**, **wake up a process, dispatch a process**, **enable a process to communicate with another process**.

- **O.S الذي يدير الـ Processes يجب أن يكون قادرة على تنفيذ عملية معينة على و مع الـ Processes.**
- **وتشمل هذه :إنشاء، تدمير، تعليق، استئناف، تغيير أولوية، منع، ايقاظ، إرسال، تمكين التواصل بين الـ processes.**

- System must provide mechanisms for:
  - Process creation
  - Process termination

Process Creation Cont.

- Creating a process involves many operations including: name a process, insert it in the ready queue, determine the process initial priority, create the PCB and allocate the process's initial resources.

- الـ Creating يتضمن عدة عمليات :
- تسمية الـ Process , إدراجها في ready queue، تحديد الأولوية الأولية للـ Process وإنشاء PCB وتخصيص الموارد الأولية للـ Process.

- A process may create a new process, the creating process is called the **parent process** and the created process is called the **child process**.

- قد تقوم Process بإنشاء Process جديدة، وتسمى الـ Process التي انشأت او خلقت Process اخرى بـ parent process والتي تم خلقها تسمى child process.

- When a process creates a new process two possibilities exist in terms of **execution**:

  a. The parent continues to execute concurrently with its children.

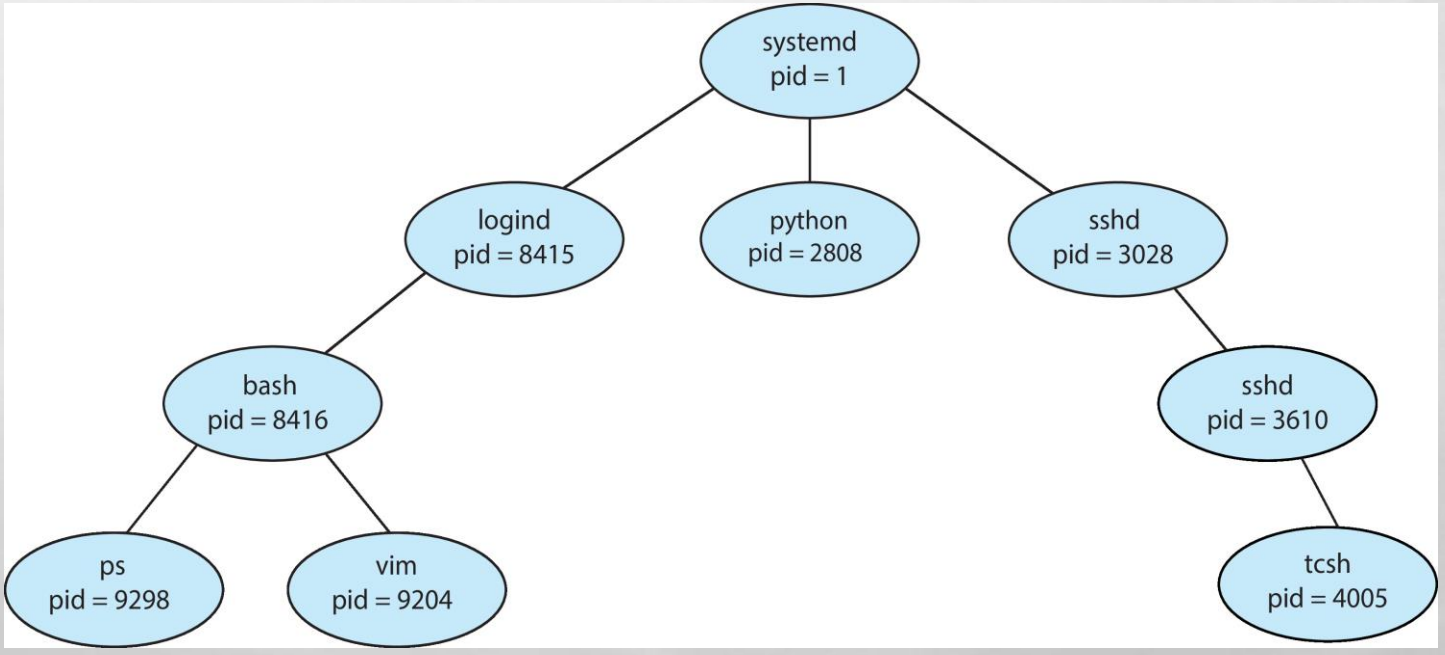
  b. The parent waits until some or all of its children have terminated.

- عندما تقوم Process بإنشاء Process جديدة يوجد احتمالان من حيث التنفيذ :
- أ. يستمر الـ parent في التنفيذ بالتزامن مع children.
- ب. ينتظر الـ parent حتى يتم إنهاء بعض أو كل children.

Process Creation Cont.

- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources

- خيارات مشاركة الموارد
- 1- يتقاسم Parent and children جميع الموارد
- 2- يشترك Children في مجموعة فرعية من موارد Parent
- 3- لا يشترك Parent and child في أي موارد



Process Termination

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the **exit ()** system call.

- ينتهي تنفيذ الـ process عند انتهاء تنفيذ آخر عبارة او ايعاز موجود ضمنه , ويطلب من نظام التشغيل حذفه باستخدام exit () system call

- Parent may terminate the execution of children processes using the abort() system call. Some reasons for doing so:
  a. Child has exceeded allocated resources
  b. Task assigned to child is no longer required
  c. The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

- قد ينهي الـ Parent تنفيذ الـ children processes باستخدام abort() system call .

- بعض الأسباب للقيام بذلك:
- أ –تجاوز الـ Child على الموارد المخصصة له
- ب .المهمة المعينة للـ Child لم تعد مطلوبة
- ج– يخرج الـ parent وأنظمة التشغيل لا تسمح للـ child بالمتابعة إذا تم إنهاء parent

3.10 Cooperating Processes

- Processes executing within a system may be ***independent*** or ***cooperating***
  - قد تكون الـ Processes المنفذة داخل النظام مستقلة أو متعاونة

- A process is **independent** if it is:

  1. Cannot affect or be affected by the other processes executing in the system.
  2. Does not share data with any other process.

  - تكون الـ Process مستقلة إذا كانت:
  1. لا تؤثر أو تتأثر بالـ Processes الأخرى المنفذة في النظام.
  2. لا تشارك البيانات مع أي Process أخرى.

- A process is **cooperating** if it is:

  1. Can affect or be affected by the other processes executing in the system.
  2. Share data with any other process.

  - تكون الـ Process متعاونة إذا كانت:
  1. تؤثر أو تتأثر بالـ Processes الأخرى المنفذة في النظام.
  2. تشارك البيانات مع أي Process أخرى.

- Reasons for cooperating processes:
  - **INFORMATION SHARING** تبادل المعلومات
  - **COMPUTATION SPEEDUP** تسريع الحسابات
  - **MODULARITY** النمطيه
  - **CONVENIENCE** الاقناع

3.11 Thread structure

- A thread sometimes called lightweight process (**LWP**) is a basic unit of CPU utilization and consists of a program counter, a register set, and a stack space. It shares with peer threads its code section, data section and O.S resources such as open files and signals collectively known as a task. A traditional or heavy weight process is equal to a task with one thread.

- يعتبر مؤشر(خيط, سلسلة) الترابط الذي يطلق عليه أحيانًا lightweight process (**LWP**)الوحدة الأساسية لاستخدام وحدة المعالجة المركزية ويتكون من عداد البرنامج ومجموعة مسجلات ومساحة مكدس. يشارك مع مؤشرات الترابط النظير, المقطع البرمجي وقسم البيانات وموارد O.S مثل الملفات المفتوحة والإشارات الجماعية المعروفة على انها مهمة. ال PROCESS التقليدية أو الثقيلة مساوية الى مهمة اوعملية ذات خيط واحد.

Thread structure Cont.

Threads can be in one of several states ready, blocked, running, or terminated. Threads can create child threads if one thread is blocked another thread can run. Unlike processes threads are not independent of one another, because all threads can access in the task.

يمكن أن تكون Threads في واحدة من عدة حالات , جاهزة , محظورة , قيد التنفيذ أو منتهية. ويمكنها أن تنشئ (تخلق) ابناء من الـ Threads إذا كان احد الThread قد قام بحظر Thread آخر يستطيع ان يعمل. بخلاف ال Threads Processes تكون غير مستقلة عن بعضها البعض ، لأن كل Threads يمكنها الوصول في المهمة.

3.12. Interrupt Processing

An interrupt is an event that alters the sequence in which a processor executes instructions. The H/W of C/S generates the interrupt. When an interrupt occurs the following actions will be taken:

a. The O.S gains control.

b. The O.S saves the state of interrupted process in its PCB.

c. The O.S analyzes the interrupt and passes control to the appropriate routine to handle the interrupt.

d. The interrupt handler routine processes the interrupt.(IHR)

e. The state of the interrupted process (or some other next process) is restored.

f. The interrupted process (or some other next process) executes.

An interrupt may be specifically initiated by a running process (in which case it is often called a trap and said to be synchronous with the operation of the process). Or it may be caused by some event that may or may not be related to the running process. It is said to be asynchronous with the operation of the process.

3.13 Interrupt classes (types)

There are five interrupt classes. These are:

1. SVC (Supervisor call) interrupts

A running process that executes the SVC instruction such as initiates these:

- I/O request. - Obtaining more storage. - Communicating with user operator.

2. I/O interrupts

There are initiated by the I/O H/W. such as:

- An I/O operation completes.
- An I/O error occurs.
- When a device is made ready.

3. External interrupts:

These are caused by various events including:

- The expiration of a quantum (كمية) on an interrupting clock.
- Pressing of the console's interrupt key by the operator.
- Receipt of a signal from another processor.

Interrupt classes (types) Cont.

4. Restart interrupts:
These occur when the operator:
- Presses the console's restart bottom.
- When a restart signal processor instruction arrives from another processor on a multi-processor system.

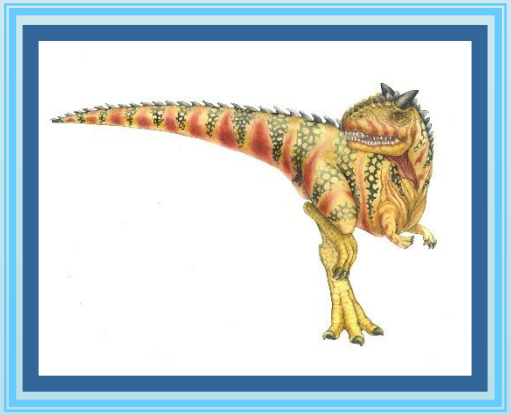
5. Program checks interrupt:
These are caused by many problems such as:
- Divide by zero.
- Arithmetic overflow.
- Data is in the wrong format.
- Attempt to execute invalid operation code.
- Attempt to reference a memory location beyond the limits of main memory.
- Attempt to execute a privileged instruction.
- Attempt to reference a protected resource.

End of Chapter Three

Mustansiriyah University
College of Education
Computers Science Department

Operating Systems Concepts
Chapter 4: CPU Scheduling

جدولة وحدة المعالجة المركزية



Assist. Prof. Dr. Hesham Adnan ALABBASI

4.1. CPU Scheduling

- تستخدم جدولة وحدة المعالجة المركزية في الانظمة التي تعمل بمبدأ الـ Multiprogramming.
- في نظام الحاسبة البسيط ، الـ CPU يكون خامل (Ideal), لذلك سيكون هنالك ضياع هو وقت الانتظار. لا يتم إنجاز أي عمل مفيد.
- مع الـ Multiprogramming، نحاول استخدام هذا الوقت بشكل منتج, فيكون هنالك بعض الـ Processes في التنفيذ في جميع الأوقات وذلك لتحقيق أقصى قدر من الاستفادة من الـ CPU.
- يتم الاحتفاظ بعدد من الـ Processes في الذاكرة في وقت واحد, ويتم تنفيذ الـ Process الى ان يحدث عليها مقاطعة (Interrupt) وتدخل في حالة الـ Waiting كأن يكون نوع الـ Interrupt هو I/O Request, فيجب أن تنتظرً لاستكمال هذا الـ Request فيأخذ نظام التشغيل الـ CPU من هذه الـ Process ويعطي الـ CPU الى Process أخرى و يستمر بهذا النمط.

4.2. Process Concept CPU-I/O Burst Cycle

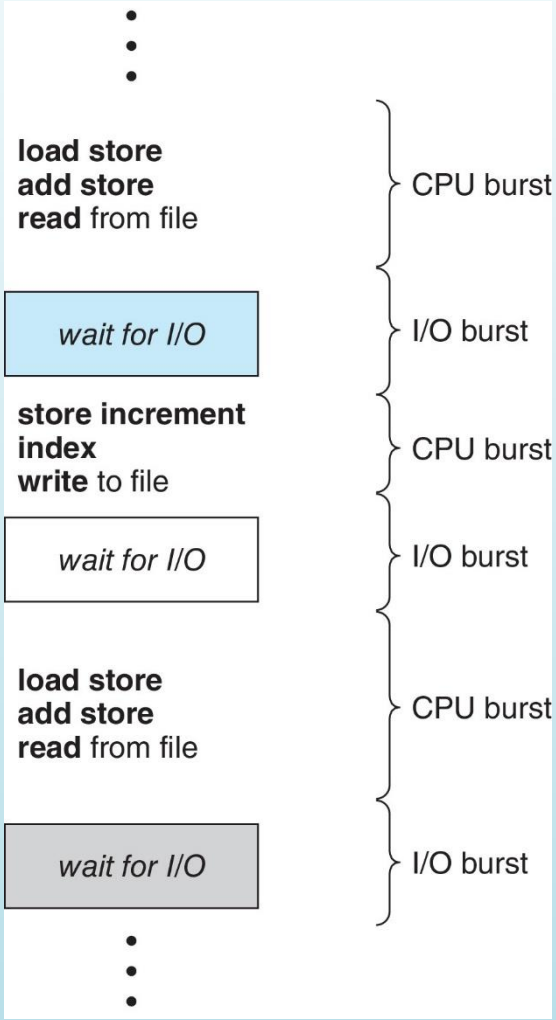
- يعتمد نجاح جدولة الـ CPU على خاصية الـ Processes التي تمت ملاحظتها.
- تنفيذ الـ Process يكون دورة من حالتين:

1- الـ CPU execution

2- I/O wait وتسمى بالاندفاع (Burst)

الـ Processes تتبادل بين هاتين الحالتين CPU burst يتبعها I/O burst ومن ثم يتبعها burst CPU وهكذا.

- أخيرًا ، ينتهي الـ burst النهائي لـ CPU بطلب إنتهاء التنفيذ.



4.3. CPU Scheduler (مجدول)

- عندما تصبح الـ CPU في وضع الخمول (Idle) ، يجب على نظام التشغيل أن يختار إحدى الـ Processes الموجودة في الـ ready queue لكي تنفذ.
- عملية الاختيار هذه الـ Process يتم تنفيذ عن طريق **short-term scheduler** او **CPU Scheduler**
- يقوم الـ Scheduler بأختيار Process واحدة من الـ Processes الموجودة في الذاكرة (في الـ ready queue) والتي تكون جاهزة للتنفيذ ، ويخصص او يحدد الـ CPU لتلك الـ Process .

4.5. Dispatcher(المرسل)

- الـ Dispatcher (المرسل) هو module يعطي السيطرة على الـ CPU للـ Process المحددة بواسطة short-term scheduler هذه العملية تشمل

1. Switching context.
2. Switching to user mode.
3. Jumping to the proper location in the user program to restart that program.

- يجب أن يكون المرسل أسرع ما يمكن لانه يتم استدعاؤه أثناء عملية تبديل الـ Process.

- يُعرف الوقت المستغرق من قبل الـ Dispatcher لإيقاف أحدى الـ Process وبدء تشغيل آخرى بأسم **Dispatch latency**

4.6. Scheduling Criteria معايير الجدولة

تحتوي خوارزميات جدولة الـ (CPU)المختلفة على خصائص متعددة، وقد يؤدي اختيار خوارزمية معينة إلى تفضيل فئة واحدة من العمليات على أخرى.
تم اقتراح العديد من المعايير لمقارنة خوارزميات جدولة الـ (CPU) تشمل المعايير ما يلي:

1. **CPU utilization**: الاستفادة من الـ CPU: أبقاء الـ CPU مشغولة بقدر المستطاع
2. **Throughput** : الانتاجية: عدد الـ Processes التي اكتمل تنفيذها خلال وحدة زمن.
3. **Turnaround time:** كمية الوقت المستغرق لتنفيذ process محدد.
4. **Waiting time:** وقت الانتظار: كمية الوقت الذي يقضيه الـ Process للانتظار في الـ ready queue .
5. **Response time:** وقت الاستجابة: كمية الوقت المحسوب من تقديم (دخول) الـ Process الى الوقت الذي تتم اول استجابة له

Scheduling Algorithm Optimization Criteria are: معايير تحسين خوارزمية الجدولة

- Max CPU utilization, Max throughput
- Min turnaround time, Min waiting time and Min response time

4.7 Scheduler Algorithms

خوارزميات الجدولة قد تكون:

1- استباقية **Preemptive** بمعنى يمكن استقطاع او ايقاف عمل الـ Process قيد التنفيذ والانتقال بالتنفيذ الى Process اخرى.

2- غيراستباقية **Non-Preemptive** بمعنى لا يمكن استقطاع او ايقاف عمل الـ Process قيد التنفيذ والانتقال بالتنفيذ الى Process اخرى.

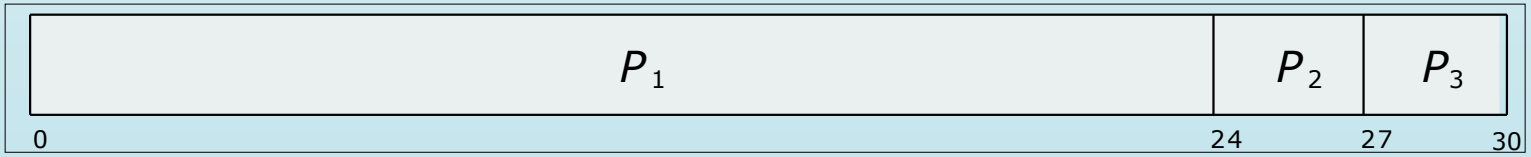
ملاحظة: كل Process يحتاج الى وقت محدد لاكمال تنفيذه يدعى بالـ Burst time

4.7.1. First- Come, First-Served (FCFS) Scheduling

عمل الخوارزمية : يتم التنفيذ اعتمادا على وصول الـ Processes وايضا وفق ترتيب وصولها, بمعنى تنفيذ الاول ثم الثاني الى الاخير وفق ترتيب الوصول المحدد في السؤال, وهذه الخوارزمية هي Nonpreemptive.

Example 1: Suppose that the processes arrive at time 0, in the order: P_1 , P_2 , P_3 Draw the Gantt chart and calculate the average waiting?

Processes	Burst time
P1	24
P2	3
P3	3



- Waiting time:
  P_1 = (0-0)=0
  P_2 = (24-0)=24
  P_3 = (27-0)=27

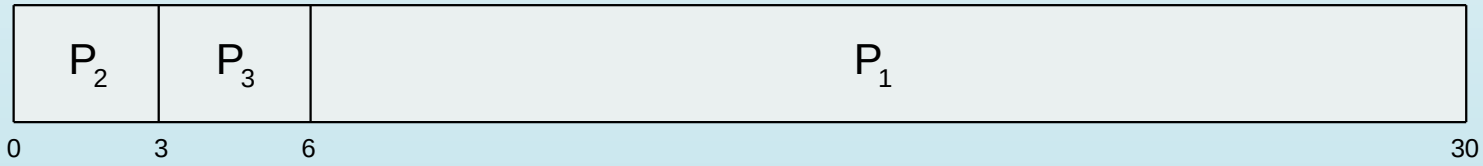
Waiting time= start time – arrival time ▪
وقت الانتظار= وقت بدأ التنفيذ – وقت الوصول

- Average waiting time: (0 + 24 + 27)/3 = 17 Ms.

FCFS Scheduling (Cont.)

Example 2: Suppose that the processes arrive at time 0, in the order: P_2 , P_3 , P_1 Draw the Gantt chart and calculate the average waiting?

Processes	Burst time
P1	24
P2	3
P3	3



- Waiting time:
  $P_1 = 6$
  $P_2 = 0$
  $P_3 = 3$

- Average waiting time: (6 + 0 + 3)/3 = 3 Ms.

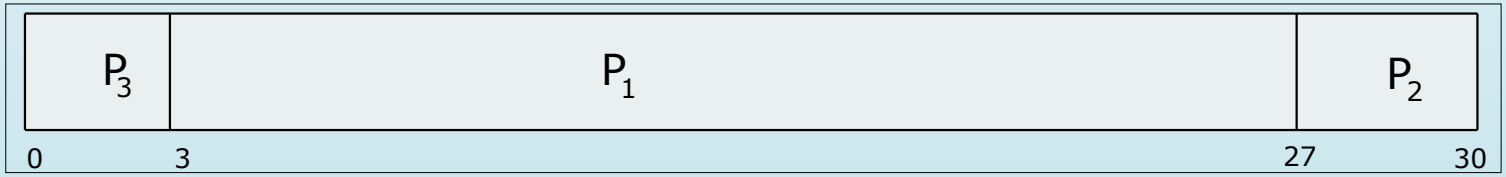
Much better than previous case

FCFS Scheduling (Cont.)

Example 3: Draw the Gantt chart and calculate the average waiting for the processes as in the given table ?

Processes	Burst time	Arrival time
P1	24	2
P2	3	5
P3	3	0

Processes	Burst time	Arrival time
P3	3	0
P1	24	2
P2	3	5



Waiting time: start time – arrival time ▪

- Waiting time:

  P_1 = (3-2)=1
  P_2 = (27-5)=22
  P_3= (0-0)=0

- Average waiting time: (1 + 22 + 0)/3 = (23/3)= 7.666 Ms.

4.7.2 Shortest-Job-First (SJF) Scheduling

عمل الخوارزمية : يتم التنفيذ اعتمادا على الوقت الذي يستغرقة الـ Process في التنفيذ (Burst time), فيتم ترتيب الـ Processes حسب الوقت من الاقل للاعلى ويبدأ بالتنفيذ.

هذه الخوارزمية تكون:

1- **Preemptive**

2- **Non-preemptive**.

*** الـ SJF هي الأمثل – حيث تعطي الحد الأدنى لمتوسط وقت الانتظار لمجموعة معينة من الـ processes**

A- (SJF) Scheduling (Non-preemptive)

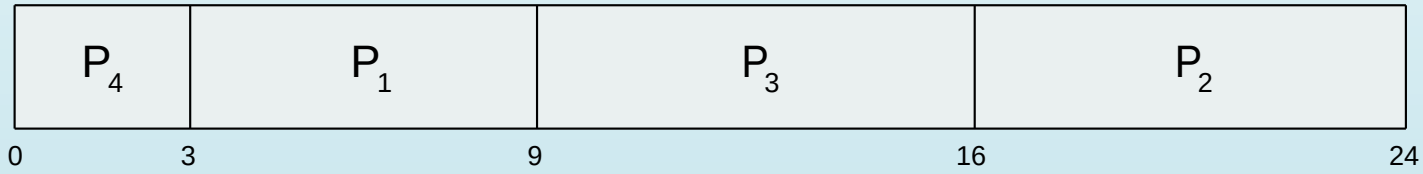
Example 1:Draw the Gantt chart and calculate the average waiting for the processes as in the given table (**all processes arrived at time 0**)?

في هذا المثال جميع الـ processes وصلت في نفس الوقت (0) .

Processes	Burst time
P1	6
P2	8
P3	7
P4	3

A- (SJF) Scheduling (Non-preemptive)

Processes	Burst time
P4	3
P1	6
P3	7
P2	8



- Waiting time:
  $P_1 = 3$
  $P_2 = 16$
  $P_3 = 9$
  $P_4 = 0$

- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7 Ms.

(SJF) Scheduling (Non-preemptive) Cont.

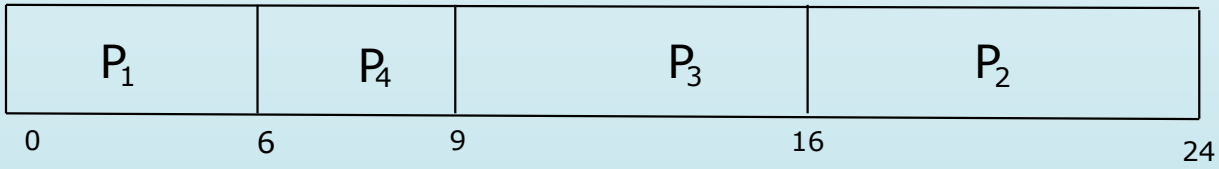
- Now we add the concepts of varying arrival times (اضافة وقت الوصول)
- Example 2:Draw the Gantt chart and calculate the average waiting for the processes as in the given table?

Processes	Arrival time	Burst time
P1	0	6
P2	2	8
P3	3	7
P4	4	3

- يبدأ التنفيذ باختيار اول Process وصولاً حتى وان كان الـ Burst time له عالي مقارنة بالـ processes الاخرى (في السؤال) لانه الوحيد الواصل في ذلك الوقت ويستمر بتنفيذه الى حين اكماله لانها (Nonpreemptive).
- عند اكمال تنفيذه نلاحظ وصول جميع الـ processes الاخرى, هنا يكمل التنفيذ من الاقل Burst time الى الاعلى.

(SJF) Scheduling (Non-preemptive) Cont.

Processes	Arrival time	Burst time
P1	0	6
P4	4	3
P3	3	7
P2	2	8



Waiting time= start time – arrival time

- Waiting time:

  P_1 = (0-0)=0
  P_2 = (16-2)=14
  P_3= (9-3)=6
  P_4= (6-4)= 2

- Average waiting time = (0 + 14 + 6 + 2) / 4 = 5.5 Ms.

B- (SJF) Scheduling (Preemptive) Cont.

- Example 3: Draw the Gantt chart and calculate the average waiting for the processes as in the given table?

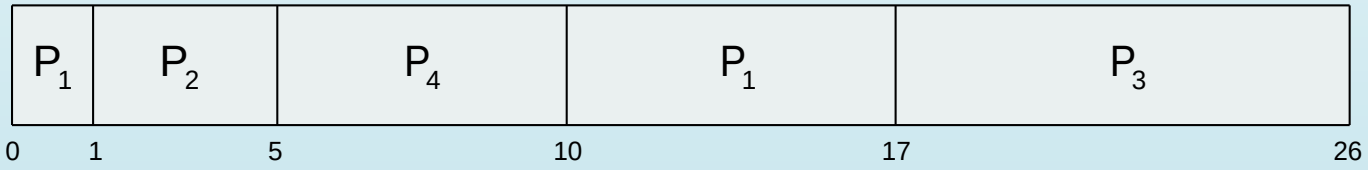
Processes	Arrival time	Burst time
P1	0	8 / 7
P2	1	4
P3	2	9
P4	3	5

- يبدأ التنفيذ باختيار اول Process (P1) وصولاً وبعد 1 Ms. سوف يستقطع لان الـ Bust time للـ Process (P2) الواصل بعده اقل منه

- يستمر بالتنفيذ لنفس الـ Process مع ملاحظة الـ burst time للـ Processes اذا كانت اقل منه يستقطع وينتقل للاخر واذا اقل يبقى بتنفيذه.

B- (SJF) Scheduling (Preemptive) Cont.

Processes	Arrival time	Burst time
P1	0	8 / 7
P2	1	4
P3	2	9
P4	3	5



Waiting time= (start time – arrival time)+(2nd/3rd start – 1st/2nd execute time)

- Waiting time:

  P_1 = [(0-0)+(10-1)]=9
  P_2 = (1-1)=0
  P_3= (17-2)=15
  P_4= (5-3)= 2

- Average waiting time = (9 + 0 + 15 + 2) / 4 = 6.5 Ms.

4.7.3 Priority Scheduling

* رقم الأولوية (الأهمية) عبارة عن رقم صحيح مرتبط بكل الـ Processes ضمن المدى (0-127), الاقل هو أعلى اولوية.

* الـ CPU تحجز الى الـ Process ذو الاعلى أولوية (اهمية) بمعنى الذي لدية أصغر رقم.

* عمل الخوارزمية : يتم التنفيذ اعتمادا على هذا الرقم Priority , فيتم ترتيب الـ Processes حسب الأولوية من الأصغر رقماً (أعلى أولوية) الى الاعلى (أقل أولوية) ويبدأ بالتنفيذ.

هذه الخوارزمية تكون:

1- **Preemptive**

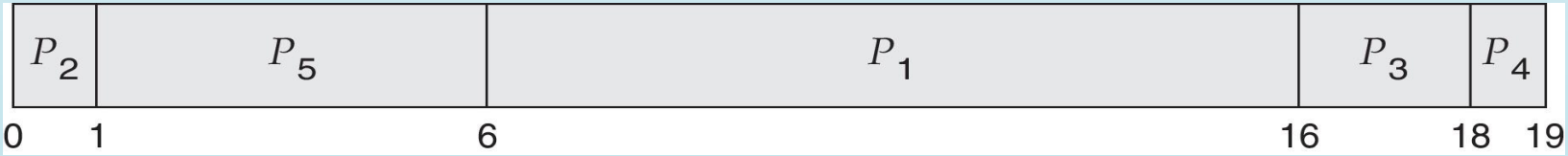
2- **Non-preemptive**.

A-Priority Scheduling (Non-preemptive)

- Example 1:Draw the Gantt chart and calculate the average waiting for the processes as in the given table, **All processes arrived at the same time (time 0)**?

Processes	Burst time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Processes	Burst time	Priority
P2	1	1
P5	5	2
P1	10	3
P3	2	4
P4	1	5



- Waiting time:

$P_1 = 6$
$P_2 = 0$
$P_3 = 16$
$P_4 = 18$
P5= 1

Average waiting time = (6+0+16+18+1) / 5= 8.2 Ms.

Priority Scheduling (Non-preemptive) Cont.

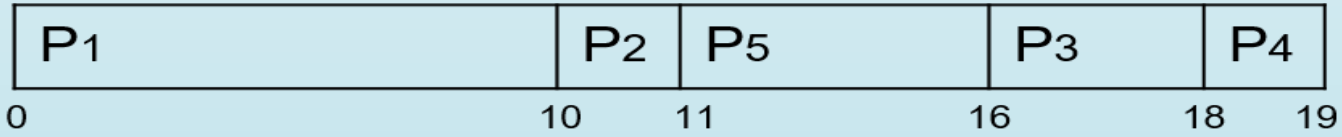
- Example 2:Draw the Gantt chart and calculate the average waiting for the processes as in the given table?

Processes	Burst time	Priority	Arrival time
P1	10	3	0
P2	1	1	1
P3	2	4	2
P4	1	5	3
P5	5	2	4

Processes	Burst time	Priority	Arrival time
P1	10	3	0
P2	1	1	1
P5	5	2	4
P3	2	4	2
P4	1	5	3

Priority Scheduling (Non-preemptive) Cont.

Processes	Burst time	Priority	Arrival time
P1	10	3	0
P2	1	1	1
P5	5	2	4
P3	2	4	2
P4	1	5	3



- Waiting time:

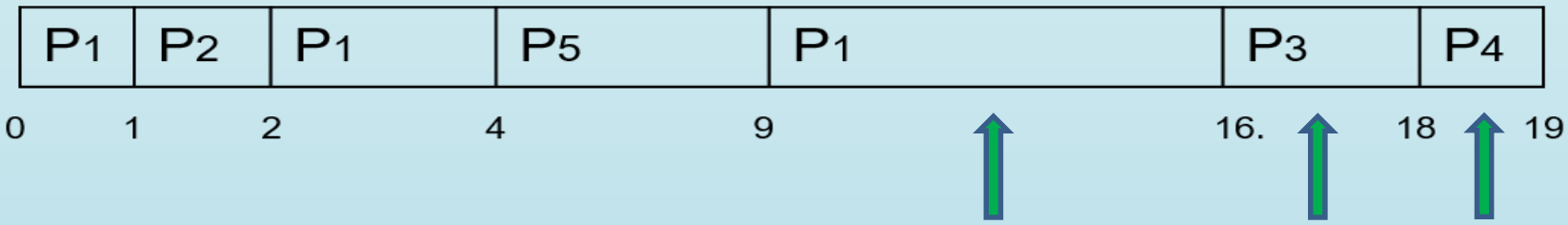
P_1 = [(0-0)=0
P_2 = (10-1)=9
P_3 = (16-2)=14
P_4 = (18-3)=15
P_5 = (11-4)=4

Average waiting time = (0+9+14+15+4) / 5 = 78.4 Ms.

B-Priority Scheduling (Preemptive)

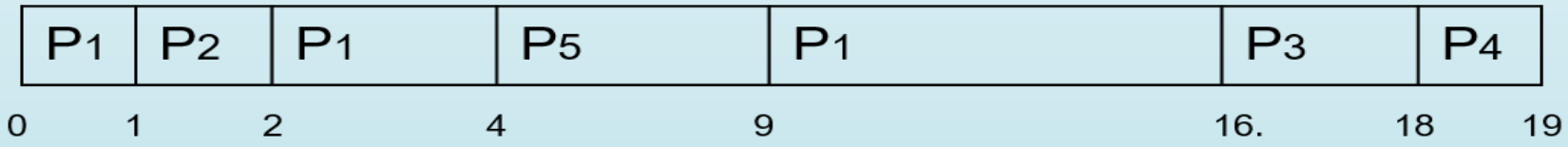
- Example 3:Draw the Gantt chart and calculate the average waiting for the processes as in the given table?

Processes	Burst time	Priority	Arrival time
P1	10 9 8 7	3	0
P2	1	1	1
P3	2	4	2
P4	1	5	3
P5	5	2	4



Priority Scheduling (Preemptive)

Processes	Burst time	Priority	Arrival time
P1	10	3	**0**
P2	1	1	**1**
P3	2	4	**2**
P4	1	5	**3**
P5	5	2	**4**



- **Waiting time:**

  P_1 = [(0-0)+(2-1)+(9-4)]=6

  P_2 = (1-1)=0

  P_3= (16-2)=14

  P_4= (18-3)=15

  P_5= (4-4)=0

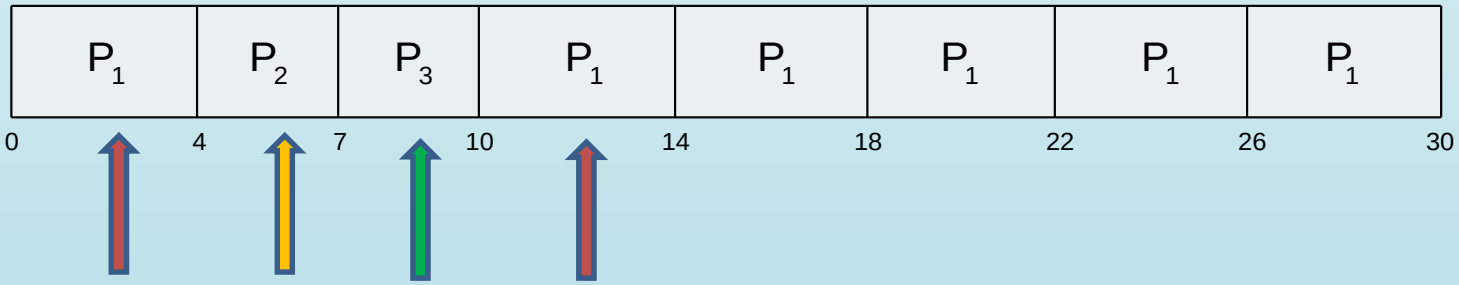
Average waiting time = (6+0+14+15+0) / 5 = 7 Ms.

4.7.4 Round Robin Scheduling (RR)

- كل Process يحصل على وحدة صغيرة من وقت الـ CPU تدعى بالـ (**time quantum *q***) عادةً تكون بين **(100-10) ms.** بعد ان ينقضي هذا الوقت اذا كانت الـ Process لم تنهي عملها ضمنه فيتم استقاع عملها وتضاف الى نهاية الـ Ready queue .

- Example 1: Draw the Gantt chart and calculate the average waiting for the processes as in the given table with **time quantum** *q=4*?

Processes	Burst time
P1	24
P2	3
P3	3



- Waiting time:

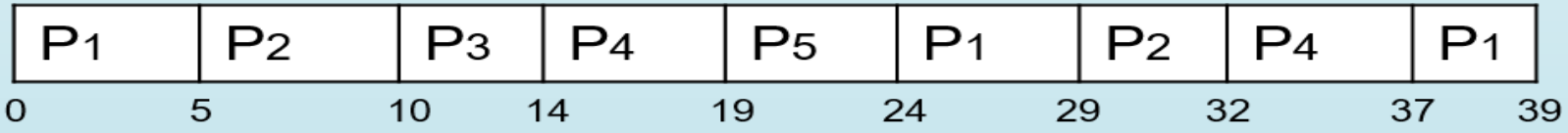
P_1 = [0+(10-4)]=6
P_2 = 4
P_3 = 7

The average waiting time is (6+4+7)/3 = 5.66 Ms.

Round Robin Scheduling (RR)Cont.

- Example: Draw the Gantt chart and calculate the average waiting time for the processes as in the given table with **time quantum** *q=5*?

Processes	Burst time
P1	12
P2	8
P3	4
P4	10
P5	5



- Waiting time:

P_1 = [0+(24-5)+(37-29)]=27
P_2 = [(5+(29-10)]=24
P_3= 10
P_4= [14+(32-19)=27
P_5= 19

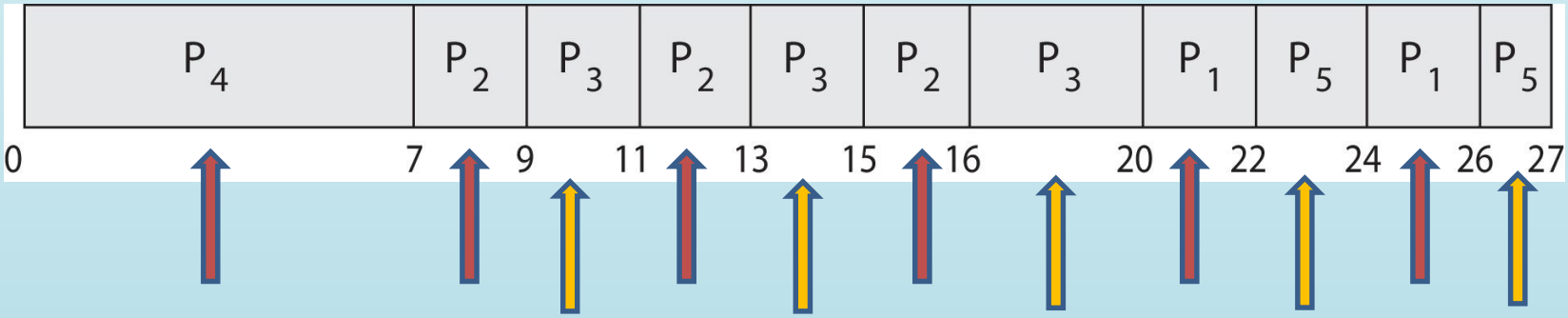
Average waiting time = (27+24+10+27+19) / 5 = 21.4 Ms.

Priority Scheduling with / Round-Robin

- Example: Draw the Gantt chart and calculate the average waiting for the processes as in the given table with **time quantum** *q=2,* using **Priority** and **Round Robin** algorithms?

Processes	Burst time	Priority
P1	4	3
P2	5	2
P3	8	2
P4	7	1
P5	3	3

- الحل: يتم تنفيذ الـ Process ذو أعلى أولوية . الـ Processes التي تكون متساوية في الاولوية يتم تنفيذها باستخدام RR .



- Calculate the average waiting time?

4.7.5 Multilevel Queue Scheduling

- تم إنشاء فئة أخرى من خوارزميات الجدولة للحالات التي يتم فيها تصنيف الـ Processes بسهولة إلى مجموعات مختلفة.

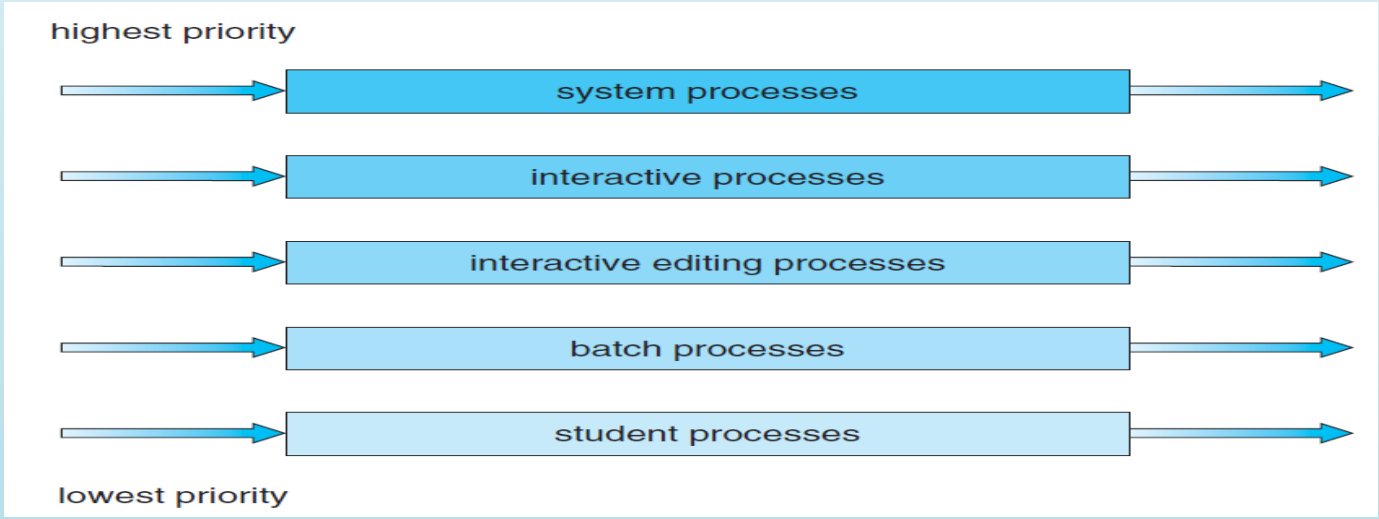
1- Foreground (or interactive)
2- Background (or batch)

لهذين النوعين من الـ Processes متطلبات وقت استجابة مختلفة ، وبالتالي قد يكون لها احتياجات جدولة مختلفة. بالإضافة إلى ذلك ، قد يكون للـ Foreground Processes الأولوية على الـ Foreground Processes.

Multilevel Queue Scheduling Cont.

- تُقسم خوارزمية الجدولة الـ Ready queue إلى عدة queues منفصلة. يتم تعيين الـ Processes بشكل دائم إلى واحد من الـ queue بناءً على بعض خصائص الـ Process ، مثل حجم الذاكرة أو الـ Priority أو نوع الـ Process.
- لا تنتقل الـ Processes بين الـ queues، لذلك فأنها لا تغير طبيعتها (تصنيفها) **Foreground** أو **Background**
- لكل queue خوارزمية جدولة خاصة بها

- يمكن جدولة **Foreground queue** بأستخدام خوارزمية **RR**
- يتم جدولة **Foreground Background** بأستخدام خوارزمية **FCFS**



This setup has the advantage of low scheduling overhead, but it is inflexible.

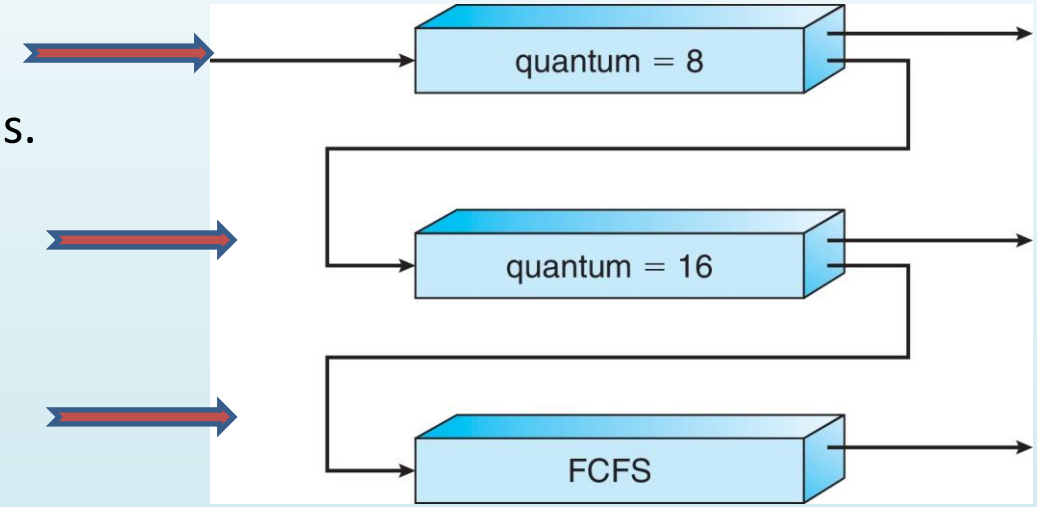
هذا الإعداد لديه ميزة هي تقليل نفقات الجدولة العامة ، لكنه غير مرن.

4.7.6 Multilevel Feedback Queue Scheduling

- تسمح هذه الجدولة للـ Processes بالانتقال بين الـ queues. تتمثل الفكرة في فصل الـ Processes وفقًا لخصائص CPU bursts بها.
- إذا كانت الـ Process تستهلك الكثير من وقت الـ CPU، فسيتم نقلها إلى queue ذو Priority أقل.
- وبالمثل ، قد يتم نقل الـ Process التي تنتظر طويلاً في queue ذو Priority أقل, إلى queue ذو Priority أعلى.

Example of Multilevel Feedback Queue

- **Three queues:**
- Q_0 – RR with time quantum 8ms.
- Q_1 – RR time quantum 16ms
- Q_2 – FCFS

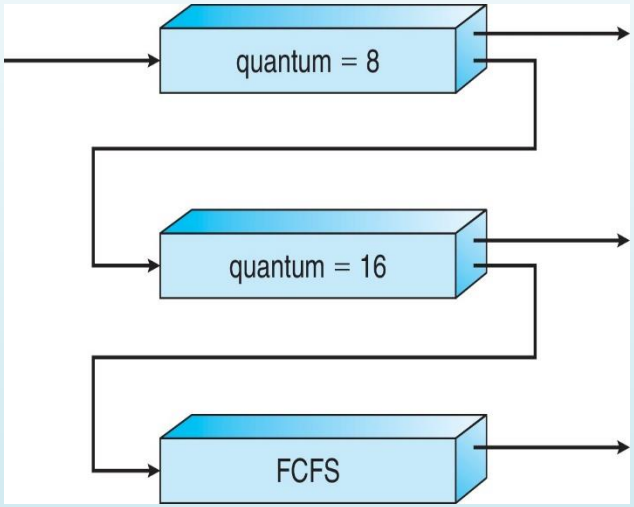


- يقوم الـ scheduler اولاً بتنفيذ جميع الـ Processes في الـ queue Q_0 .
- فقط عندما يكون الـ queue Q_0 فارغ، سيتم تنفيذ الـ Processes في queue Q_1.
- وبالمثل ، سيتم تنفيذ الـ Processes في queue Q_2 فقط إذا كان الـ queue Q_0 و queue Q_1 فارغين.

Example of Multilevel Feedback Queue Cont.

- **Scheduling**

- A new job enters queue Q_0 which is served ***RR** with **quantum=8 Ms.*** If it does not finish in ***8*** milliseconds, job is moved to queue Q_1
- At Q_1 job is again served ***RR** with* additional ***quantum=16 Ms.*** If it still does not complete, it is preempted and moved to queue Q_2
- At queue Q_2, processes are run on an **FCFS**



- نلاحظ ان الجدولة بهذه الخوارزمية تعطي أولوية قصوى لأي Process الـ CPU Burst <= 8ms وبذلك ستحصل هذه الـ Process بسرعة على الـ CPU ، وتنهي عملها.
- الـ Processes التي تحتاج الى اكثر من 8ms وأقل من 24ms ايضا سوف تنفذ بسرعة.
- الـ Processes التي تحتاج الى اكثر من هذا الوقت سوف تكون موجودة في queue 2 وتنفذ بـ FCFS.

Example:

Draw the Gantt chart and find the A.W.T (Average Waiting Time) using **Multilevel feedback Scheduling**? **NOTE: all processes arrive at time 0.**

Processes	Burst time
P1	30
P2	8
P3	42
P4	20
P5	4
P6	14

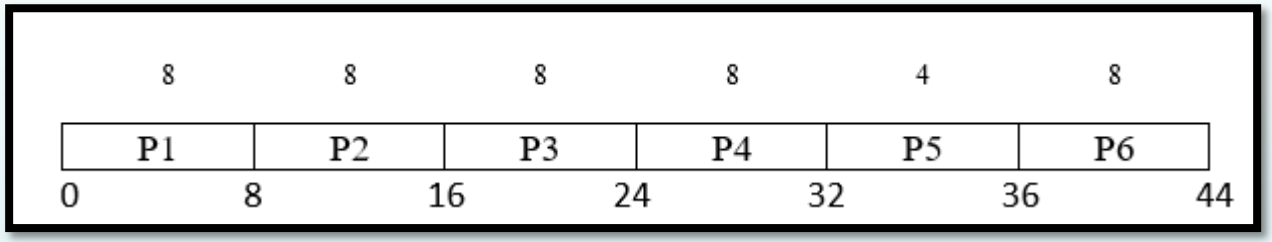
الحل:

جميع الـ processes واصلة الى النظام في نفس الوقت time0.

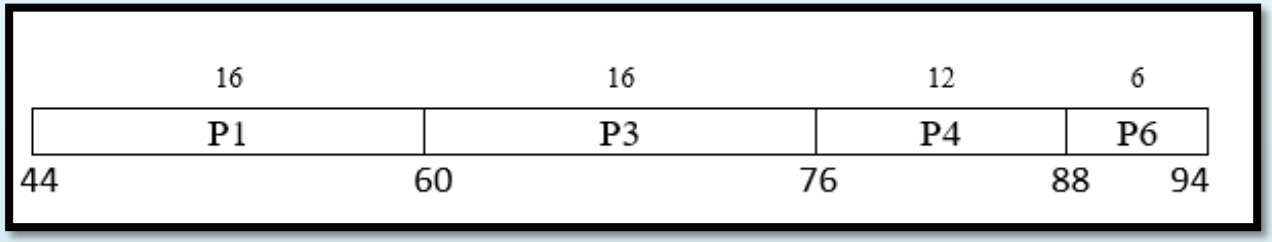
1. الـ queue 0 ينفذ بـ RR بمقدار q=8
2. الـ queue 1 ينفذ بـ RR بمقدار q=16 للـ processes التي لم ينتهي تنفيذها في 1
3. الـ queue 2 ينفذ بـ FCFS للـ processes التي لم ينتهي تنفيذها في 2

Example:

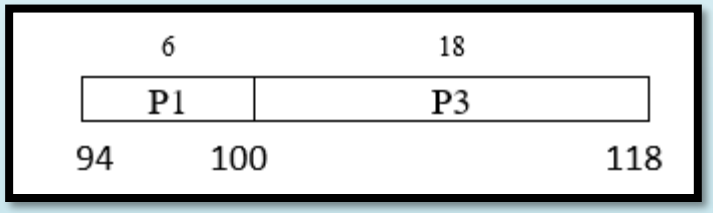
Queue O, q=8



Queue 1, q=16



Queue 2, FCFS



A.W.T.: (start-arrival)

P1= 0+(44-8)+(94-60)=70 , P2= 8, P3=16+(60-24)+(100-76)=76
P4 =24+(76-32)=68 , P5=32, P6= 36+(88-44)=80

A.W.T = (70+8+76+68+32+80)/6= 334/6= 55.666ms.

4.7.8 Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?

قد يكون من الصعب تحديد الخوارزمية لعدة اسباب:

- المشكلة الأولى هي تحديد المعايير التي ستستخدم في اختيار خوارزمية .غالبًا ما يتم تحديد المعايير من حيث الـ CPU utilization أو response time أو throughput.
- لتحديد خوارزمية ، يجب علينا أولاً تحديد الأهمية النسبية لهذه المقاييس .قد تتضمن المعايير عدة مقاييس، مثل:

- تعظيم الـ CPU utilization تحت القيد بأن الحد الأقصى للـ response time هو ثانية واحدة.

- تعظيم الـ Throughput الى الحد الأقصى بحيث يكون الـ response time في المتوسط يتناسب خطياً مع إجمالي الـ execution time.

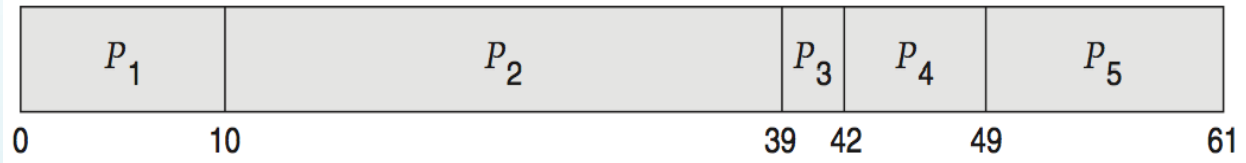
Example: Algorithm Evaluation

Assume that we have the workload shown. All five processes arrive at time 0, in the order given, with the length of the CPU-burst time given in milliseconds:

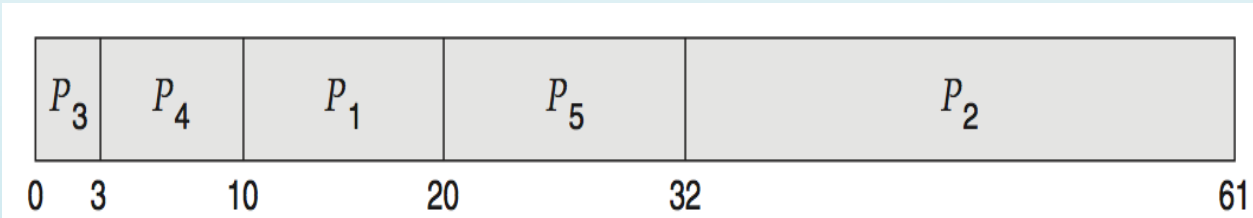
Processes	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

Consider the **FCFS, SJF (non-preemptive),** and **RR** (quantum = 10 ms) scheduling algorithms for this set of processes. Which algorithm would give the minimum average waiting time?

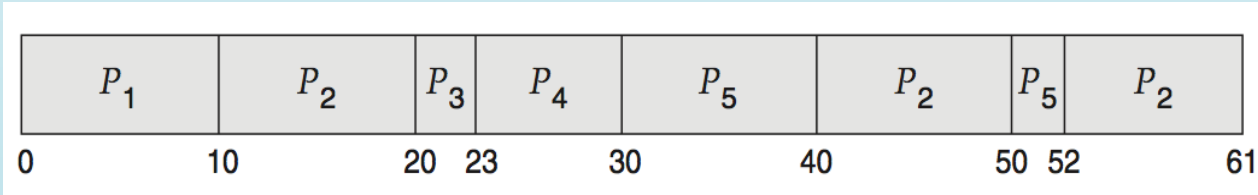
Processes	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12



- FCFS, average waiting time is 28ms.



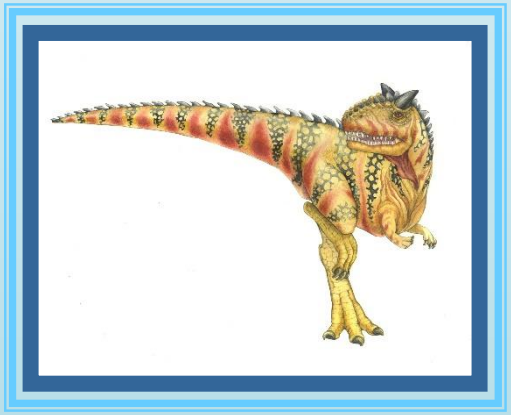
- Non-preemptive SFJ average waiting time is 13ms.



- RR, average waiting time is 23ms.

نلاحظ ان الـ SJF تعطي معدل انتظار اقل من نصف معدل انتظار الـ FCFS
ومعدل انتظار الـ RR هو متوسط.
لذلك فان افضل خوارزمية اعتماداً على معدل وقت الانتظار هي الـ SJF .

End of Chapter 4



Chapter Five

5.1. Process Synchronization

5.1.1. Background

A cooperating process is one that can affect or be affected by other processes executing in the system. Cooperating processes can either directly share a logical address space (that is, both code and data) or be allowed to share data only through files or messages.

We've already seen that processes can execute concurrently or in parallel. The role of process scheduling and has been described how the CPU scheduler switches rapidly between processes to provide concurrent execution. This means that one process may only partially complete execution before another process is scheduled. In fact, a process may be interrupted at any point in its instruction stream, and the processing core may be assigned to execute instructions of another process.

• Concurrent access to shared data may result in data inconsistency

• Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes

الـ process المتعاونة هي الـ process التي يمكن أن تؤثر أو تتأثر با الـ processes الأخرى المنفذة في النظام. يمكن للـ process المتعاونة إما مشاركة مساحة عنوان منطقية مباشرة (أي كل من الكود والبيانات) أو السماح لها بمشاركة البيانات فقط من خلال الملفات أو الرسائل.

لقد رأينا بالفعل أن ا الـ processes يمكن تنفيذها بشكل متزامن أو بالتوازي. وقد وصف دور جدولة الـ process وكيفية تبديل جدولة وحدة المعالجة المركزية بسرعة بين ا الـ processes لتوفير التنفيذ المتزامن. هذا يعني أن process واحدة قد تكتمل التنفيذ جزئيا فقط قبل جدولة عملية أخرى.

في الواقع ، قد تتم مقاطعة الـ process في أي نقطة في تدفق التعليمات الخاص بها ، وقد يتم تعيين نواة المعالجة لتنفيذ تعليمات عملية أخرى.

*قد يؤدي الوصول المتزامن إلى البيانات المشتركة إلى عدم تناسق البيانات

*يتطلب الحفاظ على اتساق البيانات آليات لضمان التنفيذ المنظم للـ process المتعاونة

5.1.2. Race Condition

If there are several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition**. To guard against the race condition above, we need to ensure that only one process at a time can be manipulating the variable counter. To make such a guarantee, we require that the processes be synchronized in some way.

Situations such as the one just described occur frequently in operating systems as different parts of the system manipulate resources. Furthermore, as we have emphasized in earlier chapters, the growing importance of multicore systems has brought an increased emphasis on developing multithreaded applications. In such applications, several threads which are quite possibly sharing data are running in parallel on different processing cores. Clearly we want any changes that result from such activities not to interfere with one another.

إذا كان هناك وصول للعديد من ال processes والتلاعب في نفس البيانات في وقت واحد ونتائج التنفيذ يعتمد على ترتيب معين الذي يعتمد وصولها, ويسمى شرط سباق. للحماية من حالة السباق أعلاه ، نحتاج إلى التأكد من أن process واحدة فقط في كل مرة يمكن أن تعالج عداد متغير. لتقديم مثل هذا الضمان ، نطلب مزامنة ال processes بطريقة ما.

تحدث حالات مثل الحالة الموصوفة للتو بشكل متكرر في أنظمة التشغيل حيث تتلاعب أجزاء مختلفة من النظام بالموارد. علاوة على ذلك ، كما أكدنا في الفصول السابقة ، أدت الأهمية المتزايدة للنظم متعددة النواة إلى زيادة التركيز على تطوير تطبيقات متعددة مؤشرات الترابط. في مثل هذه التطبيقات ، يتم تشغيل العديد من مؤشرات الترابط التي من المحتمل جدا مشاركة البيانات بالتوازي على نوى معالجة مختلفة. من الواضح أننا نريد ألا تتداخل أي تغييرات ناتجة عن مثل هذه الأنشطة مع بعضها البعض.

5.1.3. The Critical-Section Problem

We begin our consideration of process synchronization by discussing the so called critical-section problem. Consider a system consisting of n processes $\{P0, P1, ..., Pn-1\}$. Each process has a segment of code, called a **critical section**, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is allowed to execute in its critical section.

That is, no two processes are executing in their critical sections at the same time. The ***critical section problem*** is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the **entry section**. The critical section may be followed by an **exit section**. The remaining code is the **remainder section**. The general structure of a typical process *Pi* is shown in Figure 5.1. The entry section and exit section are enclosed in boxes to highlight these important segments of code.

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual exclusion. If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded waiting. There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

نبدأ نظرنا في تزامن الـ process من خلال مناقشة ما يسمى مشكلة القسم الحرج. لنفرض نظام يتكون من n العمليات {P0, P1, ..., Pn−1} تحتوي كل process على جزء من التعليمات البرمجية ، يسمى القسم الحرج ، حيث قد تقوم الـ process بتغيير المتغيرات الشائعة ، وتحديث جدول ، وكتابة ملف ، وما إلى ذلك. الميزة المهمة للنظام هي أنه عندما يتم تنفيذ process واحدة في قسمها الحرج ، لا يسمح بتنفيذ أي process أخرى في قسمها الحرج. أي أنه لا يتم تنفيذ اثنان من الـ processes في أقسامهما الحرجة في نفس الوقت. تكمن مشكلة القسم الحرجة في تصميم بروتوكول يمكن للـ processes استخدامه للتعاون. يجب أن تطلب كل process إذنا لدخول قسمها الحرج. قسم التعليمات البرمجية التي تنفذ هذا الطلب هو قسم الإدخال. قد يتبع القسم الحرج قسم الخروج. الكود المتبقي هو قسم الباقي. يظهر الهيكل العام لprocess نموذجية بي في الشكل 5.1. يتم تضمين قسم الدخول وقسم الخروج في مربعات لتسليط الضوء على هذه الأجزاء الهامة من التعليمات البرمجية.

الحل لمشكلة القسم الحرج يجب أن يفي بالمتطلبات الثلاثة التالية:

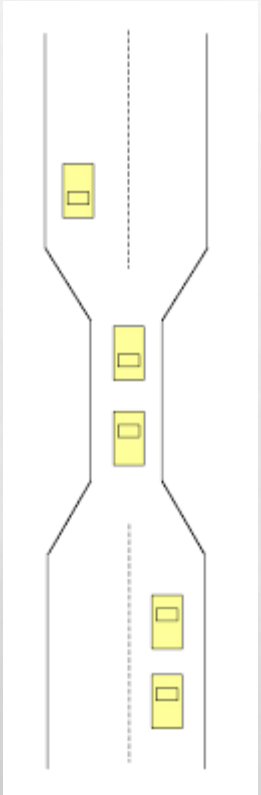
1. الاستبعاد المتبادل. إذا كانت process Pi تنفذ في قسمها الحرج ، فلا يمكن تنفيذ أي processes أخرى في أقسامها الحرجة.

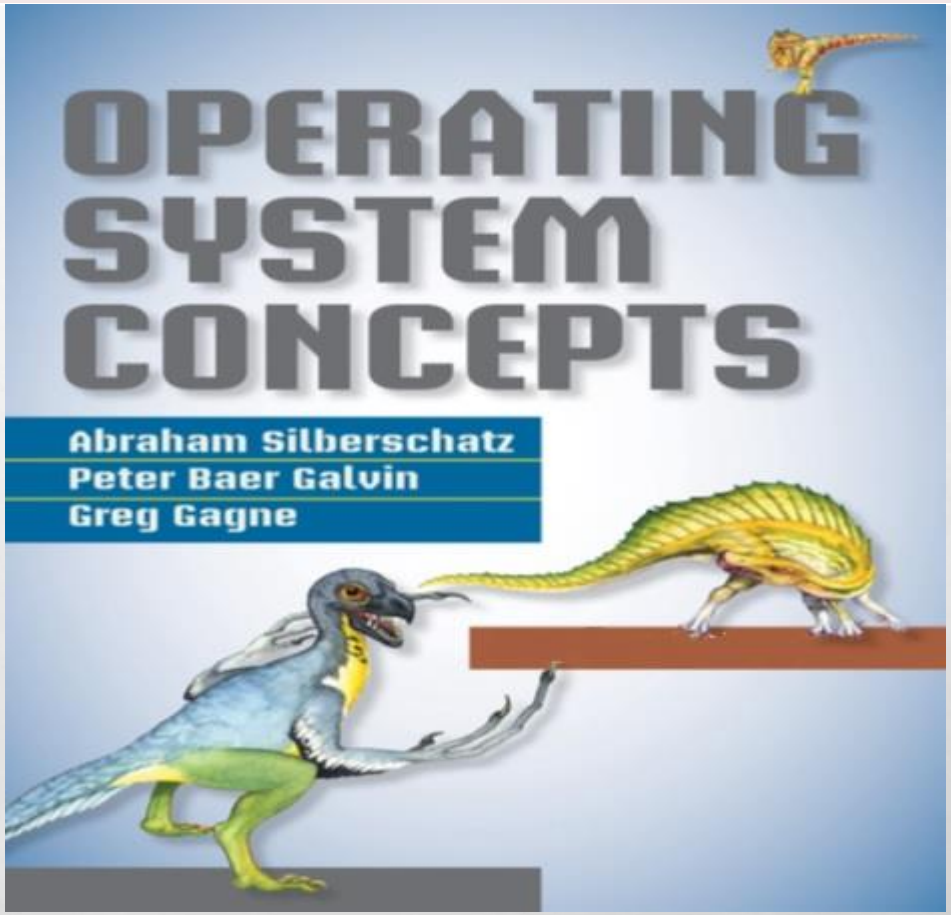
2. التقدم. إذا لم يتم تنفيذ أي process في قسمها الحرج وترغب بعض الـprocesses في إدخال أقسامها الحرجة ، فعندئذ فقط تلك ا الـprocesses التي لا يتم تنفيذها في أقسامها المتبقية يمكن أن تشارك في تحديد أي منها سيدخل قسمها الحرج بعد ذلك ، ولا يمكن تأجيل هذا الاختيار إلى أجل غير مسمى.

3. حدود الانتظار. يوجد حد أو تحديد لعدد المرات التي يسمح فيها للـprocesses الأخرى بدخول أقسامها الحرجة بعد أن تقدم الـ process عملية طلبا لدخول قسمها الحرج وقبل منح هذا الطلب.

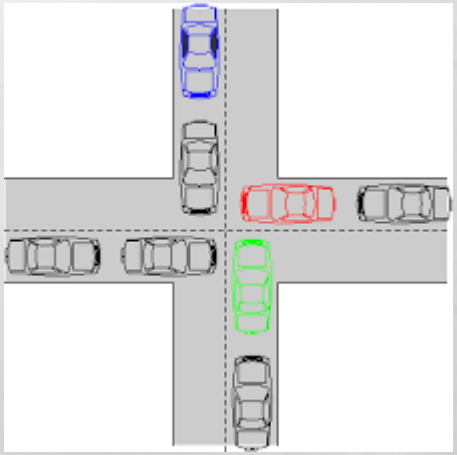
Mustansiriayah University
Collage of Education
Computers Science Department

Chapter Six
Deadlock I





Fourth Class



Assist. Prof. Dr. Hesham Adnan ALABBASI

2021-2022

6.1.Deadlock

- In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state.

  في بيئة البرمجة المتعددة ، قد تتنافس عدة processes على عدد محدود من الموارد. الـ processesتطلب الموارد؛ إذا كانت الموارد غير متوفرة في ذلك الوقت ، فإن ال process تدخل حالة انتظار.

- Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

- في بعض الأحيان ، لا يمكن لل process التي تنتظر ان تغير حالتها ، لأن الموارد التي طلبتها يتم الاحتفاظ بها بواسطة waiting processes أخرى. تسمى هذه الحالة Deadlock

- operating systems typically do not provide deadlock-prevention facilities, and it remains the responsibility of programmers to ensure that they design deadlock-free programs.

- لا توفر أنظمة التشغيل عادةً تسهيلات منع الDeadlock ، ويبقى من مسؤولية المبرمجين التأكد من تصميم برامج خالية من الDeadlock .

6.1.1. System Model

- System consists of a finite number of resources
- Resource types R1, R2, . . ., Rm
  CPU cycles, memory space, I/O devices
- Each resource type Ri has Wi instances.

- A process must request a resource before using it and must release the resource after using it.
  - يجب ان يطلب الـ process الموارد قبل استخدامها ويجب ان يحررها بعد استخدامها

- A process may request as many resources as it requires to carry out its designated task. The number of resources requested may not exceed the total number of resources available in the system.
  - قد تطلب الprocess ما تحتاج إليه من موارد لتنفيذ المهمة المحددة لها. بحيث لا يتجاوز عدد الموارد المطلوبة عن عدد الموارد المتاحة في النظام.

System Model Cont.

- Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. Request: The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

- تطلب الـ process المورد. إذا كان لا يمكن منح الطلب على الفور (على سبيل المثال ، إذا كان المورد قيد الاستخدام من قبل process أخرى) ، فيجب أن تنتظر process التي قامت بالطلب حتى تتمكن من الحصول على المورد.

2. Use: The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

- تستطيع الـ process أستخدام المورد. (كمثال: اذا كان المورد هو printer , فتستطيع الـ process الطباعة على هذه الـ Printer .

3. Release: The process releases the resource.

- الـ process يحرر المورد

To illustrate a deadlocked state:

- consider a system with **three CD R/W drives**. Suppose each of **three processes** holds one of these CD R/W drives.
- If each process is now requesting another drive, the three processes will be in a deadlocked state.
- Each is waiting for the event "**CD RW is released**," which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.

- لتوضيح حالة الDeadlock :

- النظام به مورد واحد من الـ محركات أقراص CD R / W. به ثلاثة من هذا النوع مع ثلاثة processes.
- افترض أن كل process من الثلاث يستخدم او يمسك أحد محركات أقراص CD R / W.
- إذا كانت كل process تطلب الآن محرك أقراص آخر ، فستكون ال processes الثلاث في حالة Deadlock.
- كل منها ينتظر الحدث "تم تحرير CD RW" والذي يمكن أن يحدث فقط بسبب إحدى waiting processes الأخرى. يوضح هذا المثال حالة Deadlock لنظام يتضمن نفس نوع المورد.

- Deadlocks may also involve different resource types.
  - For example, consider a system with one **printer,** and one **DVD drive**.
  - Suppose that process ***Pi*** is holding the DVD and process ***Pj*** is holding the printer.
  - If ***Pi*** requests the printer and ***Pj*** requests the DVD drive, a deadlock occurs.

- قد تشمل الDeadlock أيضا أنواع مختلفة من الموارد.
على سبيل المثال ، النظام به طابعة واحدة ومحرك أقراص DVDواحد.
افترض أن process ***Pi*** يستخدم او يمسك قرص DVD وأن process ***Pj*** ت يستخدم او يمسك الطابعة.
إذا طلبت Pi الطابعة وطلبت Pj محرك أقراص DVD، يحدث Deadlock.

6.2. Deadlock Characterization

In a deadlock, processes never finish executing, and system resources are tied up (مقيد), preventing other jobs from starting

6.2.1. Necessary Conditions

Deadlock can arise if four conditions hold simultaneously (في نفس الوقت).

1. **Mutual exclusion:** At least one resource must be held in a **non-sharable mode**; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

   الاستبعاد المتبادل: يجب الاحتفاظ بمورد واحد على الأقل في وضع غير قابل للمشاركة ؛ بمعنى ، process واحدة فقط في كل مرة يمكن ان يستخدام المورد. إذا طلبت process أخرى ذلك المورد ، فيجب تأجيل الـ process الطالب حتى يتم تحرير المورد.

2. **Hold and wait:** a process holding at least one resource and waiting to acquire additional resources held by other processes

   اللامساك والانتظار: الـ process يمسك موردًا واحدًا على الأقل وتنتظر الحصول على موارد إضافية تمسك بها الـ processes الأخرى

Deadlock Characterization Cont.

3. **No preemption: Resources cannot be preempted**; that is, a resource can be released only by the process holding it, after that process has completed its task.

لا يوجد استباق: لا يمكن استباق (استقطاع) الموارد ؛ أي أنه لا يمكن تحرير المورد إلا من خلال الـ process التي تحتفظ بها ، بعد أن تكمل هذه الـ process مهمتها.

4. **Circular wait**: There exists a set {P0, P1, …, Pn} of waiting processes, such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, …., Pn-1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.

يوجد على الاقل مجموعة من الـ waiting processes $\{P_0, P_1, \ldots, P_n\}$,
P_0 ينتظر مورد يمسك به P_1 , P_1 ينتظر مورد يمسك به P_2 , P_{n-1} ينتظر مورد يمسك به P_n
P_n ينتظر مورد يمسك به P_0 .

6.2.2. Resource-Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system **resource-allocation graph** consists of:

- This graph consists of a set of vertices **V** and a set of edges **E**.

- The set of vertices **V** is partitioned into two different types of nodes:

  $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system

  $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system

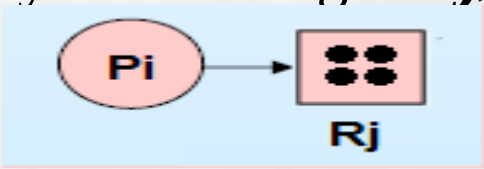
- **Request edge:** A directed edge from process ***Pi*** to resource type ***Rj*** is denoted by **Pi → Rj**; it signifies that process Pi, has requested an instance of resource type Rj, and is currently waiting for that resource.

- **Assignment edge : edge** from resource type ***Rj*** to process ***Pi*** is denoted by **Rj → Pi**; it signifies that an instance of resource type ***Rj*** has been allocated to process ***Pi***

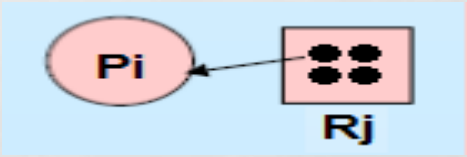
•We represent each process ***Pi***, as a circle 

•We represent each resource type ***Rj*** as a rectangle. Since resource type ***Rj*** may have more than one instance, we represent each such instance as a dot within the rectangle. 

•Note that a request edge points to only the rectangle ***Rj***



•An assignment edge must also designate one of the dots in the rectangle (instance).



- When process ***Pi***, requests an instance of resource type ***R***j, a **request edge** is inserted in the resource-allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an **assignment edge**.
- When the process no longer needs access to the resource, it releases the resource; as a result, the assignment edge is deleted.

RESOURCE ALLOCATION GRAPH EXAMPLE

The resource-allocation graph shown in the figure 6.1 depicts the following situation

- The sets P, R, and E:

- P = {P1, P2, P3}

- R = {R1, R2 ,R3, R4}

- E = {P1 →R1, P2 → R3} request edges

  ={ R1 → P2, R2 → P2, R2 → P1, R3 →P3 } assignment edges

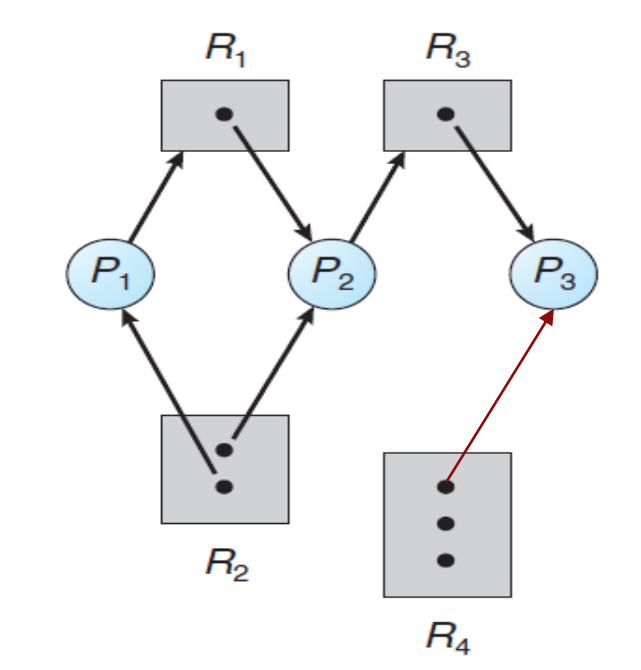


Figure 6.1

- Resource instances:

  - One instance of resource type R1

  - Two instances of resource type R2

  - One instance of resource type R3

  - Three instances of resource type R4

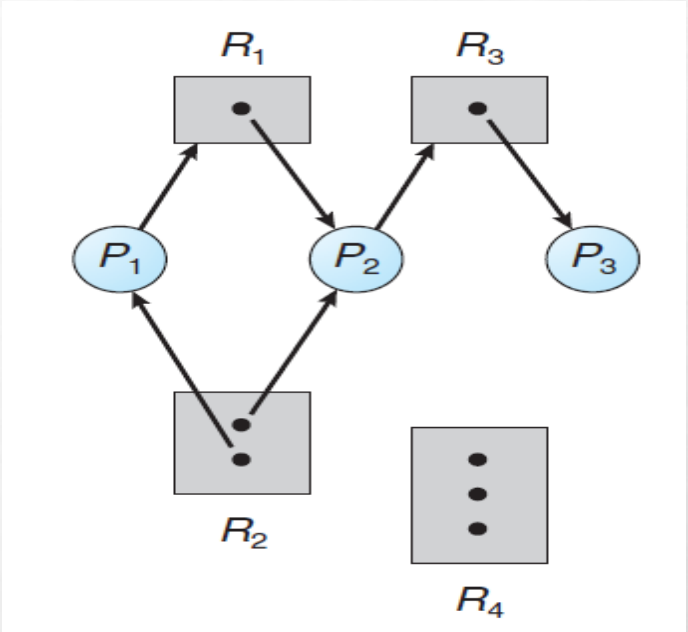
- Process states:

- Process P1 is holding (assignment) an instance of resource type R2 and is waiting (request)for an instance of resource type R1.
- Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
- Process P3 is holding an instance of R3.

Given the definition of a resource-allocation graph, it can be shown that, if the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. Each process involved in the cycle is deadlocked.

If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred.



من الشكل اعلاه يمكن ملاحظة:

- إذا لم يحتوي الرسم على cycles ، فلن تكون هناك process في النظام في حالة الـ Deadlock .
إذا كان الرسم يحتوي على cycles ، فقد يكون هناك Deadlock.
- إذا كان لكل نوع من الموارد حالة واحدة بالضبط one instance ، فإن الcycle تشير إلى حدوث Deadlocked. كل process تشارك في ال cycle وصلت إلى Deadlock.
- إذا كان لكل نوع من الموارد عدة حالات several instances ، فإن الcycle لا تعني بالضرورة حدوث Deadlock

EXAMPLE 1: To illustrate this concept, we return to the resource-allocation graph depicted in Figure 6.1.

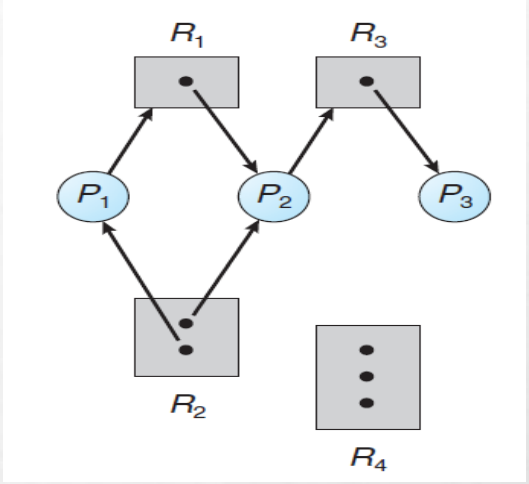


Figure 6.1

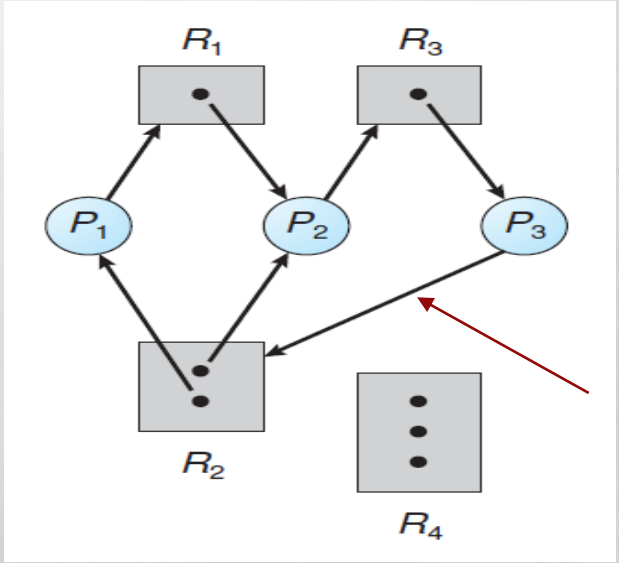
- Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3 → R2 is added to the graph (Figure 6.2)

نفرض ان الـ process P3 تطلب عدد واحد من المورد R2 . وبما ان لا يتوفر في هذه اللحظة هذا المورد (محجوز من PROCESSES اخرى, فيتم اضافة request edge P3 → R2 كما في الى الشكل 6.2

- At this point, two minimal cycles exist in the system:

1- P1 →R1→ P2 →R3→ P3 →R2→ P1

2. P2 →R3→ P3 →R2→ P2



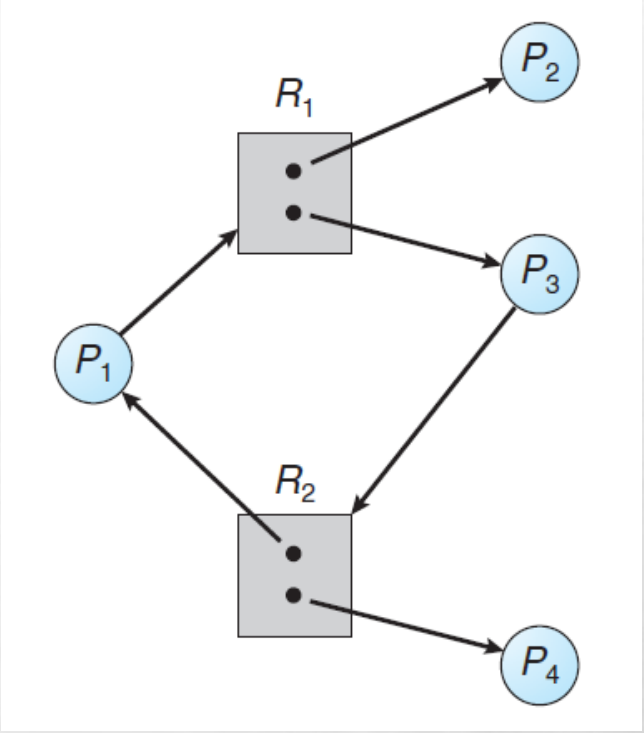
Processes P1, P2, and P3 are Deadlocked.

- Process P2 is waiting for the resource R3, which is held by process P3.
- Process P3 is waiting for either process P1 or process P2 to release resource R2.
- In addition, process P1 is waiting for process P2 to release resource R1.

EXAMPLE 2

- Now consider the resource-allocation graph in Figure 6.3. In this example, we also have a cycle: **P1 →R1→ P3 →R2→ P1**
- However, there is no deadlock. Observe that process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

هنا لا يوجد Deadlock. لاحظ ان الـ process P4 ممكن ان تحرر المورد R2 , وهذا المورد ممكن ان يحجز الى P3 فيتم كسر الـ cycle.



BASIC FACTS

- If graph contains no cycles ⇒ no deadlock.
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock

6.3 METHODS FOR HANDLING DEADLOCKS

- We can deal with the deadlock problem in one of three ways:

1. Ensuring that the system will **never** enter a deadlock state.
   - Deadlock **prevention (منع)**
   - Deadlock **avoidance** (تجنب)
2. Allow the system to enter a deadlock state and then recover.
3. Ignore the problem and pretend that deadlocks never occur in the system.

يمكننا التعامل مع مشكلة الdeadlock بإحدى الطرق الثلاث:

1. التأكد من أن النظام لن يدخل إلى حالة الdeadlock .
   - منع الـ deadlock
   - تجنب الـ deadlock
2. السماح للنظام بدخول حالة الـ deadlock ثم معالجته.
3. تجاهل المشكلة وتظاهر بأن الـ deadlock لا يحدث أبداً في النظام.

1- To ensure that deadlocks never occur, the system can use either:

- a deadlock prevention or
- a deadlock avoidance scheme.

- Deadlock prevention provides a set of methods to ensure that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how requests for resources can be made.

يوفر منع حالة الـ deadlock مجموعة من الطرق لضمان عدم توفر واحد على الأقل من الشروط الضرورية .تمنع هذه الطرق الـ deadlock عن طريق تقييد كيفية تقديم طلبات الموارد.

- Deadlock avoidance requires that the operating system be given additional information in advance concerning which resources a process will request and use during its lifetime. With this additional knowledge, the operating system can decide for each request whether or not the process should wait.

يتطلب تجنب الـ Deadlock إعطاء نظام التشغيل معلومات إضافية مقدمًا بشأن الموارد التي ستطلبها وتستخدمها الـ process خلال فترة حياتها .من خلال هذه المعرفة الإضافية ، يمكن لنظام التشغيل أن يقرر لكل طلب ما إذا كانت الـ process ستنتظر أم لا

To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

لتحديد ما إذا كان يمكن تلبية الطلب الحالي أو تأجيله ، يجب على النظام مراعاة الموارد المتاحة حاليًا ، والموارد المخصصة حاليًا لكل process، والطلبات والإصدارات المستقبلية لكل process .

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may arise. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock (if a deadlock has indeed occurred).

إذا كان النظام لا يستخدم إي من طرق منع أو تجنب حالة الـ deadlock ، فقد ينشأ أو يحدث الـ deadlock

في هذه البيئة ، يمكن للنظام توفير خوارزميات:

- تفحص حالة النظام لتحديد ما إذا كان قد وصل إلى الـ deadlock
- وخوارزمية لمعالجة او حل الـ deadlock (إذا حدث بالفعل الـ deadlock).

In the absence of algorithms to detect and recover from deadlocks, we may arrive at a situation in which the system is in a deadlocked state yet has no way of recognizing what has happened.

في حالة عدم وجود خوارزميات للكشف عن حالات الـ deadlock والتعافي منها ، قد نصل إلى وضع يكون فيه النظام في حالة الـ deadlock ولكن لا توجد طريقة للتعرف على ما حدث.

6.4. DEADLOCK PREVENTION

■ For a deadlock to occur, each of the four necessary conditions must hold. **By ensuring that at least one of these conditions cannot hold**, we can **prevent** the occurrence of a deadlock. We elaborate on this approach by examining each of the four necessary conditions separately.

من أجل حدوث الـ deadlock ، يجب تحقق كل من الشروط الأربعة الضرورية معاً .
ومن خلال التأكد من أن شرطًا واحدًا على الأقل من هذه الشروط لا يمكن أن يحدث ، يمكننا منع حدوث الـ deadloc .
نقوم بتوضيح هذا النهج من خلال فحص كل من الشروط الأربعة الضرورية بشكل منفصل.

- **6.4.1 MUTUAL EXCLUSION:** only one process at a time can use a resource. The mutual-exclusion condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are essentially non-sharable.

الاستبعاد المتبادل: يمكن ل process واحدة فقط في كل مرة استخدام المورد. يجب أن يكون شرط الاستبعاد المتبادل ساريًا على الموارد غير القابلة للمشاركة .على سبيل المثال ، لا يمكن مشاركة الطابعة في وقت واحد من خلال عدة processes .وعلى النقيض من ذلك ، لا تتطلب الموارد القابلة للمشاركة الوصول الحصري المتبادل وبالتالي لا يمكن أن تشارك في الـ deadlock .بشكل عام ، ومع ذلك ، لا يمكننا منع ال deadlock من خلال رفض شرط الاستبعاد المتبادل ، لأن بعض الموارد غير قابلة للمشاركة بشكل أساسي.

6.4.2. Hold and Wait: To ensure that the hold-and-wait condition, never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

- One protocol that can be used requires **each process to request and be allocated all its resources before it begins execution**.
- An alternative protocol **allows a process to request resources only when it has none.** A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

للتأكد من أن شرط **Hold and Wait** ، لا يحدث أبدًا في النظام ، يجب أن نضمن أنه ، كلما طلبت الـ process مورد ما ، فإنها لا تحتفظ بأي موارد أخرى.

- بروتوكول واحد يمكن استخدامه يتطلب من كل process ان يطلب وتخصيص جميع مواردها قبل بدء التنفيذ.
- بروتوكول بديل يسمح للـ process بطلب الموارد فقط عندما لا يكون لديه أي مورد آخر. قد تطلب الـ process بعض الموارد وتستخدمها . قبل أن يتمكن من طلب أي موارد إضافية ، يجب عليه الإفراج عن جميع الموارد التي يتم تخصيصها حاليًا.

ملاحظة: قراءة الامثلة الموجودة في المحاضرات

6.4.3. NO PREEMPTION

- The third necessary condition for deadlocks is that there is **no preemption of resources that have already been allocated**. To ensure that this condition does not hold, we can use the following protocol.
- If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- The released resources (preempted) are added to the list of resources for which the process is waiting.
- The process will be restarted only when it can regain (استرجاع)its old resources, as well as the new ones that it is requesting.

الشرط الثالث الضروري للخروج من الطريق المسدود هو أنه لا يوجد استباق (قطع) للموارد التي تم تخصيصها بالفعل .لضمان عدم استمرار هذا الشرط ، يمكننا استخدام البروتوكول التالي.

- إذا كانت الـ process تحتفظ ببعض الموارد وتطلب موردًا آخر لا يمكن تخصيصه لها على الفور ، فسيتم تحرير جميع الموارد المحتفظ بها حاليًا.
- تضاف الموارد المفرج عنها (المستقطعة) إلى قائمة الموارد التي تنتظرها الـ process .
- سيتم إعادة تشغيل الـ process فقط عندما تتمكن من استعادة (استرجاع) مواردها القديمة ، بالإضافة إلى الموارد الجديدة التي تطلبها.

6.4.4. CIRCULAR WAIT: One way to ensure that this condition never holds is to impose (فرض) a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration

إحدى الطرق للتأكد من عدم استمرار هذا الشرط أبدًا هي فرض ترتيب كامل لجميع أنواع الموارد ويتطلب من كل process بطلب موارد ان يكون بترتيب متزايد للعد.

To illustrate, we let *R = {R1, R2, ..., Rm}* be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering.

للتوضيح ، نفرض *R = {R1, R2, ..., Rm}* هي مجموعة أنواع الموارد لكل نوع من الموارد يتم تعيين رقمًا صحيحًا فريدًا ، مما يسمح لنا بمقارنة مصدرين وتحديد ما إذا كان أحدهما يسبق آخر في الترتيب.

We define a one-to-one function *F: R→N*, where *N* is the set of natural numbers. For example, if the set of resource types *R* includes tape drives, disk drives, and printers, then the function F might be defined as follows:

يتم تعريف دالة one-to-one (F: R → N) ، حيث N هي مجموعة الأعداد الطبيعية. على سبيل المثال ، إذا كانت مجموعة أنواع الموارد R تتضمن محركات الأشرطة ومحركات الأقراص والطابعات, فان الدالة F تكون:

F (tape drive) = 1

F (disk drive) = 5

F (printer) = 12

We can now consider the following protocol to prevent deadlocks: **Each process can request resources only in an increasing order of enumeration**. That is, a process can initially request any number of instances of a resource type —say, *Ri.* After that, the process can request instances of resource type *Rj* if and only if *F(Rj) > F(Ri).* For example, using the function defined previously, a process that wants to use the tape drive and printer at the same time must first request the tape drive and then request the printer. .

يمكننا الآن النظر في البروتوكول التالي لمنع حالات الـ deadlocks : **يمكن لكل process طلب الموارد فقط بترتيب متزايد من التعداد** .أي أن الـ process يمكن أن تطلب مبدئيًا أي عدد من حالات نوع المورد - مثل Ri . بعد ذلك ، يمكن أن تطلب الـ process مثيلات من نوع المورد Rj إذا وفقط إذا كان F (Rj)> F (Ri) على سبيل المثال ، باستخدام الوظيفة المحددة مسبقًا ، الـ process التي تريد استخدام محرك الأشرطة والطابعة في نفس الوقت يجب ان تطلب محرك الأشرطة أولاً ثم طلب الطابعة

Alternatively, we can require that a process requesting an instance of resource type *Rj* must have released any resources *Ri* such that *F(Ri) ≥ F(Rj).* Note also that if several instances of the same resource type are needed, a single request for all of them must be issued.
If these two protocols are used, then the circular-wait condition cannot hold

بدلاً من ذلك ، يمكننا أن نحدد أن الـ process الذي يطلب مثيل من نوع المورد Rj يجب أن تكون قد أطلق أي موارد مثلRi . بحيث يكون F (Ri) ≥ F (Rj) . لاحظ أيضًا أنه في حالة الحاجة إلى عدة حالات من نفس نوع المورد ، يجب إصدار طلب واحد لهم جميعًا.
إذا تم استخدام هذين البروتوكولين ، فلا يمكن أن يستمر شرط الـ circular-wait

6.5. Deadlock Avoidance

- Deadlock avoidance requires that the system has some additional a **prior** information available

يتطلب تجنب الـ Deadlock أن يتوفر لدى النظام بعض المعلومات الإضافية (المسبقة) المتاحة

- Simplest and most useful model requires that each process declare the ***maximum number*** of resources of each type that it may need

يتطلب النموذج الأبسط والأكثر فائدة أن تعلن كل process عن العدد الأقصى لكل نوع قد تحتاجه من الموارد

- The deadlock-avoidance algorithm dynamically examines **the resource-allocation state** to ensure that there can never be a circular-wait condition

تفحص خوارزمية تجنب الـ Deadlock ديناميكيًا حالة تخصيص الموارد لضمان عدم وجود حالة شرط circular-wait

- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

يتم تعريف حالة تخصيص الموارد من خلال عدد الموارد المتاحة والمخصصة ، والحد الأقصى المطلوب من الـ processes

6.5.1. Safe State

A state is **safe** if **the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock**. More formally, a system is in a safe state **only if there exists a safe sequence**.

تكون الحالة آمنة إذا كان بإمكان النظام تخصيص الموارد لكل process (حتى الحد الأقصى) بترتيب أو تسلسل محدد بحيث يبقى النظام يستطيع تجنب الـ Deadlock . بشكل آخر، يكون النظام في حالة آمنة فقط إذا كان هناك تسلسل آمن.

- The system is in a safe state if there exist a sequence of processes <*P1*, *P2*, ..., *Pn*> of all the processes in the systems such that for each *Pi*, the resource requests that *Pi*, can still request, can be satisfied by the currently available resources plus the resources held by all *Pj*, with *j* < *i*.

يكون النظام في حالة آمنة إذا كان هناك تسلسل للprocesses < P1، P2، ...، Pn>
لجميع الـ processes في الأنظمة وكما في التالي:
لكل الـ process Pi يستطيع ان يبقى يطلب موارد والتي يمكن تلبيتها من الموارد المتوفرة حالية اضافة الى الموارد التي يحتفظ او يمسك بها الـ process Pj على ان يكون i> j .

6.5.1. Safe State Cont.

That is:

- If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished

اذا لم تكن الموارد التي تحتاجها Pi متوفرة حالياً, فيجب عليها الانتظار (Pi) الى ان تنهي كل الـ Pj .

- When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate

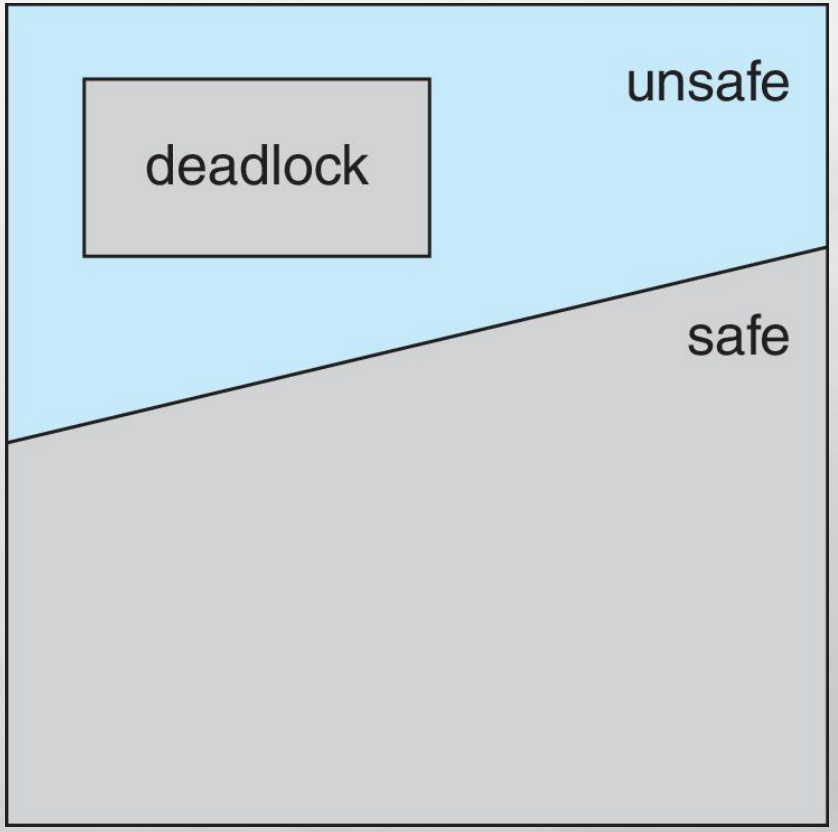
عندما تنتهي الـ Pj, الـ Pi تستطيع الحصول على الموارد التي تحتاجها وتنفذ عملها ومن ثم تطلق او تحرر جميع الموارد المحجوزة لها وتنتهي.

- When P_i terminates, P_{i+1} can obtain its needed resources, and so on

عندما تنتهي الـ Pj, الـ Pi+1 تستطيع الحصول على الموارد التي تحتاجها ويستمر التنفيذ بهذه الصيغة

Basic Facts

- If a system is in a safe state ➔ no deadlock
- If a system is in unsafe state ➔ possibly of deadlock
- Not all unsafe states are deadlocks.



- To illustrate, Consider a system with **12 magnetic tape drives** and **3 processes** (P_0, P_1, *and* P_2)
  - Process P_0 requires **10 tape drives**, (require= يطلب او يحتاج)
  - Process P_1 may need as many as **4 tape drives**, and
  - Process P_2 may need up to **9 tape drives**.

- Suppose that, at time t_0:
  - Process P_0 is holding 5 tape drives, (holding= يحتفظ أو يمسك)
  - Process P_1 is holding 2 tape drives, and
  - Process P_2 is holding 2 tape drives.

(Thus, there are **3 free tape drives**.)

Available resources = total resources in the system − total holding resources

= 12 – 9 = 3

Current Needs= Maximum Needs- Allocation.

	Maximum Needs (requires)	Current Needs	Allocation (holding)
P_0	10	5	5
P_1	4	2	2
P_2	9	7	2

	Maximum Needs (requires)	Current Needs	Allocation (holding)
P_0	10	5	5
P_1	4	2	2
P_2	9	7	2

- At time t_0, the system is in a **safe state**.

The sequence < P_1, P_0, P_2> satisfies the safety condition.

- Process P_1 can immediately be allocated all its tape drives (because the system still have 3 available tape drives) and then return them (the system will then have **(3+2) = 5** available tape drives;(2+2+1)=5.

- Then process P_0 can get all its tape drives and return them (the system will then have **(5+5) =10** available tape drives);

- And finally process P_2 can get all its tape drives and return them (the system will then have all **12** tape drives available).

- **A system can go from a safe state to an unsafe state.**
- Suppose that, at time t_1, process P_2 requests and is allocated **one more** tape drive.

	Maximum Needs (requires)	Current Needs	Allocation (holding)
P_0	10	5	5
P_1	4	2	2
P_2	9	6	3

New requests

The system is no longer in a safe state.

At this point, only process P_1, can be allocated all its tape drives. When it returns them, the system will have only **4** available tape drives. Since process P_0, is allocated **5** tape drives, but has a maximum of 10, it may request 5 more tape drives. Since they are unavailable, process P_0 must wait. Similarly, process P_2 may request an additional **6** tape drives and have to wait, resulting in a deadlock.

عند هذه النقطة ، فقط الـ process P1 يمكن ان يخصص لها ما تحتاجه من المورد tape drives وعندما يعيدها ، سيكون لدى النظام 4 من tape drives متاحة فقط. الـ process P0 كان مخصص له 5 من tape drives ولكنه يحتاج كحد أقصى 10 من tape drives فانه سيطلب 5 من tape drives و نظرًا لعدم توفرها ، يجب أن تنتظر وبالمثل ، قد تطلب الـ process P2 , 6 من tape drives وايضاً هي غير متوفرة فيجب عليه الانتظار، مما يؤدي إلى حدوث الـ Deadlock .

AVOIDANCE ALGORITHMS

- خوارزميات تجنب حدوث الـ DEADLOCK تتبع التالي:

1- SINGLE INSTANCE OF A RESOURCE TYPE

اذا كان النظام يحتوي على مثيل (حالة) واحد من نوع المورد فيتم استخدام خوارزمية

RESOURCE-ALLOCATION GRAPH

2- MULTIPLE INSTANCES OF A RESOURCE TYPE

اذا كان النظام يحتوي على مثيلات (حالت) متعددة لنوع المورد فيتم استخدام خوارزمية

USE THE BANKER'S ALGORITHM

6.5.2. Resource-Allocation Graph Algorithm

If we have a resource-allocation system with only one instance of each resource type, we can use a variant of the resource-allocation graph for deadlock avoidance. In addition to the request and assignment edges already described, we introduce a new type of edge, called a **claim edge**

- إذا كان لدينا نظام تخصيص الموارد مع مثيل واحد فقط لكل نوع من أنواع الموارد ، فيمكننا استعمال resource-allocation graph لتجنب حالة الـ Deadlock.

إضافة إلى الـ assignment edges التي تم شرحها سابقاً فيتم استخدام نوع جديد من الـ edge يسمى **Claim edge**.

Claim edge ***Pi*** → ***Rj*** indicates that process ***Pi***, may request resource ***Rj***, at some time in the future.

- **Claim edge** ***Pi*** → ***Rj*** تشير إلى أن الـ process ***Pi*** ، قد تطلب المورد ***Rj***، في وقت ما في المستقبل.

This edge resembles a request edge in direction, but is represented in the graph by a **dashed line.**

تشبه هذه الـ edge الجديدة الـ request edge في الاتجاه ، ولكن يتم تمثيلها في الرسم البياني **بخط متقطع**.

6.5.2. Resource-Allocation Graph Algorithm Cont.

1- When process ***Pi*** requests resource ***Rj***, the claim edge ***Pi*** → ***Rj*** is converted to a request edge.

1- عندما يطلب الـ ***Pi*** مصدر ***Ri*** فيتم تحويل الـ claim edge الى request edge

2- Similarly, when a resource ***Rj*** is released by ***Pi***, the assignment edge ***Rj*** → ***Pi*** is reconverted to a claim edge ***Pi ---> Rj***.

- وعندما يطلق أو يحرر المصدر ***Ri*** من قبل الـ ***Pi*** فيتم اعادة تحويل assignment edge الى claim edge

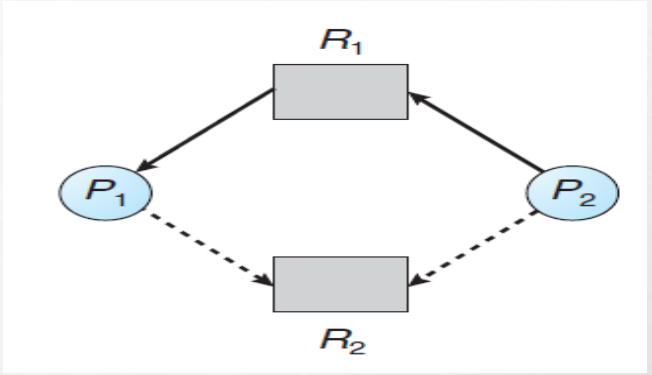
- Suppose that process ***Pi*** requests resource ***Rj***. The request can be granted only if converting the request edge ***Pi*** → ***Rj*** to an assignment edge ***Rj*** → ***Pi*** does not result in the formation of a cycle in the resource-allocation graph.

If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state.

افترض أن الـ process Pi تطلب المورد Ri . يمكن منح الطلب فقط إذا لم ينتج cycle عند تحويل الـ request edge ***Pi*** → ***Rj*** إلى assignment edge ***Rj*** → ***Pi*** في الرسم البياني.
في حالة عدم وجود cycle ، فإن تخصيص المورد سيترك النظام في حالة safe. إذا تم العثور على cycle ، فإن التخصيص سيضع النظام في حالة unsafe state.

EXAMPLE: To illustrate this algorithm, we consider the resource-allocation graph of Figure 6.5.

Figure 6.5 Resource-allocation graph for deadlock avoidance



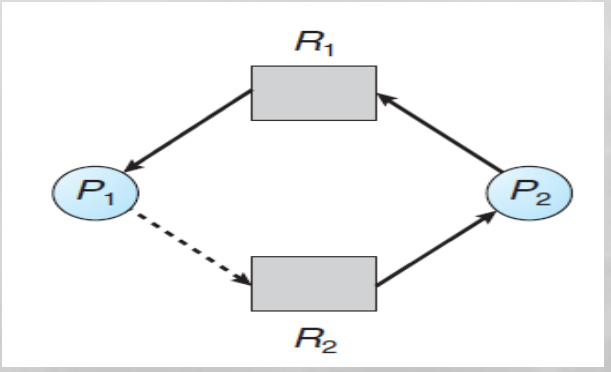
Suppose that ***P_2*** requests ***R_2***. Although ***R2*** is currently free, we cannot allocate it to ***P2***, since this action will create a cycle in the graph (Figure 6.6). A cycle indicates that the system is in an unsafe state.

If ***P1*** requests ***R2***, and ***P2*** requests ***R1***, then a deadlock will occur

افترض أن الـP2 تطلب مورد R2 وعلى الرغم من أن الـR2 متاح حالياً(غير محجوز) لا يمكننا تخصيصه إلى الـ P2، حيث سيؤدي هذا الإجراء إلى إنشاء cycle في الرسم البياني (الشكل 6.6) . تشير ال cycle إلى أن النظام في حالة unsafe.

إذا طلبت الـ P1 مورد R2 وp2 تطلب مورد R1،فسيحدث الـ deadlock

Figure 6.6 An unsafe state in a resource-allocation graph



AVOIDANCE ALGORITHMS
BANKER'S ALGORITHM

- Used with Multiple instances of resources
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

- تستخدم مع الموارد التي تحوي مثيلات (حالات) متعددة
- يجب على كل Process ان يعلن مسبقاً عن الاستخدام الأقصى من الموارد التي يحتاجها.
- عندما تطلب Process مورد ما قد تضطر إلى الانتظار
- عندما تحصل الـ Process على كل مواردها ، يجب أن تعيدها في فترة زمنية محدودة

Data Structures for the Banker's Algorithm

هيكلة بيانات خوارزمية الـ Banker

Let *n* = number of processes, and *m* = number of resources types.

n تمثل عدد الـ processes, و m تمثل عدد الموارد. الهيكلية تتكون من الأربعة التالية:

- **Available***:* A vector of length ***m*** indicates the number of available resources of each type. If ***Available [j]*** = ***k***, there are ***k*** instances of resource type ***Rj*** are available.

- **Available : (المتوفر),** متجه طوله **m** يشير الى عدد الموارد المتوفرة لكل نوع. اذا كان **Available [j] = k** هذا يعني توفر k من عدد المثيلات (حالات) المصدر **Rj**

- **Max***:* An ***n × m*** matrix defines the maximum demand of each process. I***f Max[i][j]*** =***k***, then process ***Pi*** may request at most ***k*** instances of resource type ***R j***.

- **Max** *(اقصى)*: مصفوفة من **n × m** يعرف الطلب الاقصى لكل process من الموارد.
- اذا كان **Max[i][j] =k** هذا يعني ان الـ **process Pi** ربما يطلب على الاغلب بعدد **k** من حالات المورد **Rj** .

Data Structures for the Banker's Algorithm Cont.

- **Allocation**: ***An n × m matrix*** defines the number of resources of each type currently allocated to each process. If ***Allocation[i][j] = k***, then process ***Pi*** is currently allocated ***k*** instances of resource type ***R j***.

- **Allocation**(المحجوز): مصفوفة من **n × m** تشير الى عدد الموارد المحجوزة من كل نوع لكل process. اذا كان **Allocation[i][j] = k** هذا يعني ان الـ process **Pi** يحجز في الوقت الحالي **k** من حالات المورد Rj.

- **Need**: An ***n × m*** matrix indicates the remaining resource need of each process. If ***Need[i][j] equals k***, then process ***Pi*** may need ***k*** more instances of resource type ***R j*** to complete its task.

- **Need** *(الاحتياج): مصفوفة من* **n × m** *تشير الى عدد الموارد التي يبقى يحتاجها كل نوع لكل process. اذا* **Need[i][j] = k** *هذا يعني ان الـ process* **Pi** *ربما يحتاج الى k من حالات المورد* **Rj** *لكي يكمل عمله*

Need [i][j] = Max [i][j] – Allocation [i][j]

Data Structures for the Banker's Algorithm

- To simplify the presentation of the banker's algorithm, we next establish some notation.
- Let ***X*** and ***Y*** be vectors of length ***n.*** We say that ***X*** ≤ ***Y*** if and only if ***X[i]*** ≤ ***Y[i]*** for all ***i = 1, 2, ..., n.***
- For example, if ***X = (1,7,3,2)*** and *Y* = ***(0,3,2,1)***, then ***Y*** ≤ ***X.*** In addition, *Y* < *X if* *Y* ≤ *X* and *Y* ≠ *X*.

- تبسيط التمثيل للخوارزمية:
- نفرض ***X*** and ***Y*** ***هما متجهين بالطول n . نستطيع القول ان*** ***X*** ≤ ***Y*** فقط فقط في حالة ***X[i]*** ≤ ***Y[i]*** لكل ***i = 1, 2, ..., n.***
- كمثال : كان ***X = (1,7,3,2)*** if *Y* = ***(0,3,2,1)*** و ***اذن X*** ≤ ***Y*** .

Data Structures for the Banker's Algorithm

- We can treat each row in the matrices ***Allocation*** and ***Need*** as vectors and refer to them as ***Allocation i*** and ***Need i*** .
- نستطيع معاملة كل صف في المصفوفات ***Allocation*** and ***Need*** على انهم متجه يشير اليهم بـ ***Allocation i*** و ***Need i*** .

- The vector ***Allocation i*** specifies the resources currently allocated to process ***Pi***
- المتجه ***Allocation i*** *يشير الى الموارد المحجوزة للـ* ***process Pi***.

- The vector ***Need i*** specifies the additional resources that process ***P***i may still request to complete its task.
- المتجه ***Need i*** *يشير الى الموارد الاضافية التي يمكن للـ* ***process Pi*** يبقى يحتاجه لاكمال عمله.

6.5.3.1. Safety Algorithm

We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described, as follows:

هذه الخوارزمية تستخدم في ايجاد هل ان النظام في حالة Safe أو Unsafe

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

لنفرض لدينا المتجه Work بالطول m , والمتجه Finish بالطول n . قيمهم الاولية كم ادناه:

Work = Available and
Finish [i] = false for i = 0, 1, …, n- 1

2. Find an ***i*** such that both:
   (a) ***Finish [i] = false***
   (b) $Need_i \leq Work$
   If no such ***i*** exists, go to step 4
3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   go to step 2
4. If ***Finish [i] = true*** for all ***i***, then the system is in a safe state

6.5.3.2. Resource-Request Algorithm

We now describe the algorithm which determines if requests can be safely granted.

تستخدم هذه الخوارزمية لتحديد فيما اذا كان الطلب للموارد من اي process يكمن ضمانه او لا.

Let ***Request i*** be the request vector for process ***Pi***.

If ***Request i [j] == k***, then process ***Pi***, wants ***k*** instances of resource type ***Rj.***

When a request for resources is made by process Pi the following actions are taken:

عندما يكون هنالك طلب لموارد من اي process مثلا Pi فنتبع الخطوات التالية:

1. If ***Request i < Need i***, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim."The request can't be guaranty, because the process has exceeded its maximum claim"
2. If ***Request i <= Available***, go to step 3. Otherwise, ***Pi*** must wait, since the resources are not available.

"The request can't be guaranty, because the process request is grater than available resources"

1. Pretend to allocated the requested resources to process ***Pi*** by modifying the state as follows:

Available = Available – Request i

Allocation i= Allocation i + Request i

Need i = Need i- Request i

- If safe ⇒ the resources are allocated to P_i
- If unsafe ⇒ P_i must wait, and the old resource-allocation state is restored

Example of Banker's Algorithm

A system with:
-5 processes P_0 through P_4;
-3 resource types (A, B, C).
-*A* (10 instances), *B* (5 instances), and *C* (7 instances)
Snapshot at time T_0:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

A B C
10 5 7
7 2 5
3 3 2

Answer the following questions using banker algorithm:

1. What is the content of the need matrix?
2. Is the system in safe state or unsafe state? if it safe show the sequences with details?
3. If process P1 request ***one*** additional instance of resource type ***A*** and ***two*** instances of resource type ***C***, so *Request i* = (1,0,2). can the result be granted immediately?

Example (Cont.)

5 3 2

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

Need يتم ايجاد المصفوفة ***Max - Allocation*** باستخدام

P0 ➔ 753 – 010 = 743, P1 ➔ 322- 200 = 122, …

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

1 0 2

Process	Allocat
	A B
P0	0 1
P1	2 0
P2	3 0
P3	2 1
P4	0 0

2- The system is in a safe state since if we follow this s

$< P_1, P_3, P_4, P_2, P_0 >$

$< P_3, P_1, P_4, P_2, P_0 >$

Which satisfies safety criteria

النظام في حالة Safe اذا اتبعنا التسلسل اعلاه في التنفيذ.

- We can't start with ***P0***, because it needs (**7 4 3**) which is greater than available resources (**3 2 2**).
- لا نستطيع البدء باستخدام P0 لانه يحتاج الى موارد (**7 4 3**) والتي هي اكبر من الموارد المتوفرة (**3 2 2**)

- So, we start with **P1** because Need (1 2 2)<= Available (**332**), P1 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi))

- لذلك, نبدأ بالـProcess P1 لانه يحتاج الى (1 2 2) والتي هي اصغر او تساوي Available (**332**)
- فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available = 3 3 2
2 0 0 +
---
5 3 2

- We can't select **P2** because Need (6 0 0) > new available (**5 3 2**).
- ايضا لا نستطيع اختيار الـ process P2 لان الـ Need له (6 0 0) أكبر من new available (**5 3 2**)
- لذلك نختار **P3** لان الـNeed له (0 1 1) اصغر او تساوي (**5 3 2**) new available.

For this we select **P3** because Need (0 1 1)<= new available (**5 3 2**).
P3 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi))

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available = 532
211 +

743

- Then select **P4** because Need (4 3 1)<= new available (**7 4 3**).
- نختار **P4** لان ال Need له (4 3 1) اصغر او تساوي (**7 4 3**) new available

P4 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi))

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available = 7 4 3
0 0 2 +
---
7 4 5

- Then select **P2** because Need (6 0 0)<= new available (**7 4 5**).
- نختار **P2** لان ال Need له (6 0 0) اصغر او تساوي (**7 4 5**) new available
- P2 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi)).

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available = 7 4 5
3 0 2 +
---
10 4 7

الانتباه هنا عند الجمع يتم اعتبار كل مورد كمتجه منفرد عن الاخر فيتم جمعه بصورة معزولة عن المورد الاخر

- Then select **P0** because Need (7 4 3)<= new available (**10 4 7**).
  نختار **P0** لان ال Need له (743) اصغر او تساوي new available (**10 4 7**)
- P0 get its need resources, start execute, after finished return back all resources (new available + allocation(Pi)).

فيحصل على الموارد التي يحتاجها ويبدأ التنفيذ بعد التنفيذ يرجع جميع الموارد وتجمع الموارد التي كانت محجورة له مع الموارد القديمة المتوفرة في النظام. فيكون New available كما في ادناه:

So, the New available =

```
10 4 7
 0 1 0+
---------
10 5 7
```

- وبهذه الخطوة وصلنا الى نهاية الحل وتم التأكد من ان التسلسل Sequence التالي :

$< P_1, P_3, P_4, P_2, P_0 >$

يحقق لنا ان النظام في حالة Safe

- ملاحظات للحل:

1- الانتباه هنا عند الجمع يتم اعتبار كل مورد كمتجه منفرد عن الاخر فيتم جمعه بصورة معزولة عن المورد الاخر .
2- قد يكون هنالك اكثر من تسلسل لتنفيذ الـ processes لذلك قد يختلف الحل من طالب الى آخر.
3- للتاكد من ان الحل صحيح نلاحظ الموارد النهائية بعد اكمال الحل. يجب ان تساوي مجموع كل الموارد المحجوزة للـ processes زائد المتوفرة available في النظام قبل البدء بالحل

Note: No. of all resources in the system (Allocation for all resources + Available)

```
  0 1 0
  2 0 0
  3 0 2     +
  2 1 1
  0 0 2
 _____
  7 2 5   مجموع الموارد المحجوزة لكل الـ processes
        +
  3 3 2
 ------
 10 5 7
```

وهو مطابق للناتج النهائي من الموارد الموجودة في النظام بعد الحل

3- If process P1 request ***one*** additional instance of resource type ***A*** and ***two*** instances of resource type ***C***, so *Request i* = (1,0,2). can the result be granted immediately?

. To decide whether the request of ***P1*** (1,0,2) can be immediately granted, we use Resource-Request Algorithm in section 6.5.3.2.

المطلب الثالث من المثال: اذا طلب الـ process Pi موارد اضافية هي (1,0,2) هل نستطيع تلبية طلبه في الحال ام لا, غنتبع الخوارزمية الموجودة في المقطع 6.5.3.2

. To decide whether the request of ***P1*** (1,0,2) can be immediately granted, we use Resource-Request Algorithm in section 6.5.3.2.

1-If ***Request i < Need i***, so ***P1*** request (1,0,2) **<** (1,2,2) which is **True**. The go to step 2. If it is false raise an error condition, "the process has exceeded its maximum claim"

يتم فحص الطلب اذا كان اصغر من الـ Need فيتم الانتقال الى الخطوة 2 . واذا لا يظهر له رسالة " لا يمكن تلبية الطلب لان الـ request اكبر من الـ Need .

2- Check that ***Request*** $\leq$ ***Available*** (that is, (1,0,2) $\leq$ (3,3,2) $\Rightarrow$ **true**, and allocated the requested resources to process ***P1*** by modifying the state as follows:

الخطوة الثانية يتم فحص الطلب اذا كان اصغر او يساوي المتوفر من الموارد (available) فيتم اعطاءه الموارد التي طلبها ويتم تحديث الحالة للنظام كما يلي:

- ***Available = Available – Request I***
- ***Allocation i= Allocation i + Request I***
- ***Need i = Need i- Request i***

$<P_1, P_3, P_4, P_0, P_2>$

Process	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0 (3 3 2-1 0 2) 7 4 5+ 0 1 0=7 5 5
P1	3 0 2 (2 0 0 + 1 0 2)	0 2 0	2 3 0+ 3 0 2 5 3 2
P2	3 0 2	6 0 0	7 5 5+ 3 0 2= 10 5 7
P3	2 1 1	0 1 1	5 3 2+ 2 1 1=7 4 3
P4	0 0 2	4 3 1	7 4 3 + 0 0 2=7 4 5

الجديد بعد عمل الفقرات اعلاه

We must determine whether this new system state is safe.
To do so, we execute our safety algorithm and find that the sequence **<P_1, P_3, P_4, P_0, P_2>** satisfies the safety requirement.
Hence, we can immediately grant the request of process ***P_1***.

يجب ان نحدد فيما اذا كان حالة النظام الجديدة هي Safe او Unsafe فنستخدم نفس خوارزمية Safety للتأكد من ان التسلسل **<P_1, P_3, P_4, P_0, P_2>** يلبي المتطلبات الضرورية .
لذلك يستطيع النظام توفير او منح الموارد الجديدة للـ process الذي طلبها.

1. Can request for (3,3,0) by P4 be granted?
2. Can request for (0,2,0) by P0 be granted?

6.6. Deadlock Detection (كشف)

- If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.

- اذا كان النظام لا يوفر او يوظف خوارزميات منع أو تجنب الـ Deadlock فممكن ان يحدث الـ Deadlock .

- In this environment, the system may provide:

  - Detection algorithm: An algorithm that examines the state of the system to determine whether a deadlock has occurred

  - Recovery scheme: An algorithm to recover from the deadlock.

- في هذه البيئة فيمكن للنظام ان يوفر:

- - خوارزمية الكشف: والتي تقوم بفحص حالة النظام لتحديد فيما اذا تم حدوث الـ Deadlock .

- - طريقة للتخلص أو الاسترداد من الـ Deadlock .

6.6.1. Single Instance of Each Resource Type
الموارد التي تحتوي على حالة واحدة منها

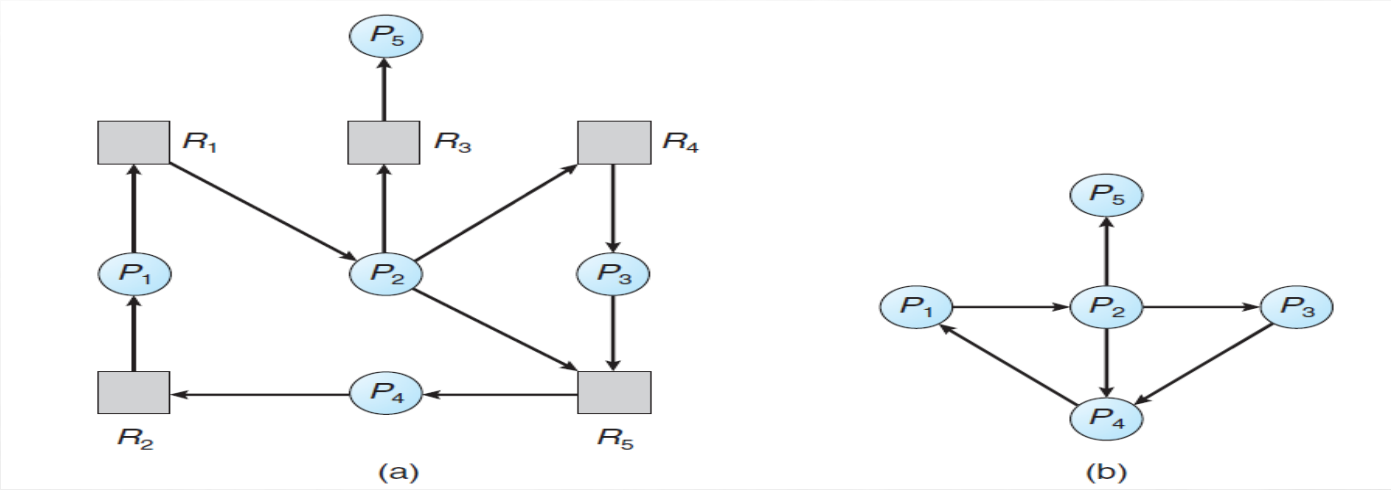
- If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a **wait-for graph**. We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

إذا كانت جميع الموارد تحتوي على مثيل واحد فقط ، فيمكننا تحديد خوارزمية كشف حالة الـ Deadlock التي تستخدم متغيرًا من resource-allocation graph يسمى **wait-for graph.** نحصل عليه عن طريق إزالة الـ **nodes** وطي الـ **edges** .

- More precisely, an edge from ***Pi*** to ***Pj*** in a wait-for graph indicates that process ***Pi*** is waiting for process ***Pj*** to release a resource that ***Pi*** needs.
- بتعبير أدق ، تشير الـedge من Pi إلى Pj في الـ wait-for graph إلى أن الـ process Pi تنتظر الـ process Pj لإطلاق المورد الذي تحتاجه Pi.
- An edge ***Pi*** → ***Pj*** exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges ***Pi*** → ***Rq*** and ***Rq*** → ***Pj*** for some resource ***Rq*** .
- توجد Pi → Pj edge في الـ wait-for graph إذا وفقط إذا كان الـ resource allocation graph contains يحتوي على أثنين من الـ edges Pi → Rq و Rq → Pj لبعض الموارد Rq.

Single Instance of Each Resource Type Cont.

In Figure 6.8, we present a resource-allocation graph and the corresponding wait-for graph.



Resource-Allocation Graph Corresponding wait-for graph

- As before, a deadlock exists in the system if and only if the wait-for graph ***contains a cycle***. To detect deadlocks, the system needs to maintain the wait for graph and periodically invoke an algorithm that searches for a cycle in the graph.

- كما في السابق ، يوجد الـ Deadlock النظام إذا وفقط إذا كان الـ wait-for graph يحتوي على ***cycle*** . لاكتشاف الـ Deadlock ، يحتاج النظام إلى الاحتفاظ بالـ wait-for graph واستدعاء خوارزمية بشكل دوري تبحث عن الـ cycle في الرسم .

6.6.2. Several Instances of a Resource Type
الموارد التي تحتوي عدة حالات

In a system with several instances of resources types, A deadlock detection algorithm that is applicable to such a system is used. This algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm.

تستخدم في النظام الذي يحتوي على عدة حالات من الموارد. هذه الخوارزمية وظف هياكل بيانات مشابهه الى خوارزمية الـ banker

- **Available***:* A vector of length ***m*** indicates the number of available resources of each type

متجه بالطول m يشير الى عدد الموارد المتوفرة من كل نوع

- **Allocation***:* An ***n*** **x** ***m*** matrix defines the number of resources of each type currently allocated to each process

مصفوفة ***n*** **x** ***m*** **تشير الى عدد الموارد المحجوزة من كل نوع لكل process**

- **Request***:* An ***n*** **x** ***m*** matrix indicates the current request of each process. If ***Request*** **[*i*][*j*] = *k***, then process P_i is requesting ***k*** more instances of resource type R_j.

مصفوفة ***n*** **x** ***m*** **تشير الى طلب كل process . اذا كان الطلب** ***Request*** **[*i*][*j*] = *k*,** هذا يعني ان الـ process Pi يطلب k من حالات المورد Rj .

Detection Algorithm

The detection algorithm described here simply investigates every possible allocation sequence for the processes that remain to be completed.

تبحث خوارزمية الكشف الموضحة هنا ببساطة في كل تسلسل تخصيص ممكن لل processes التي لم تكتمل بعد.

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n,* respectively. Initialize

لنفرض لدينا المتجه Work بالطول m , والمتجه Finish بالطول n . قيمهم الاولية كم ادناه:

Work = available *and*

Finish[i] = false for i=0, 1, ..., *n – 1.*

If Allocation ≠ 0, then and ***Finish[i]= false***, otherwise ***Finish[i]= true.***

2. Find an index i such that both

   a. ***Finish[i] ==false***

   b. ***Need i ≤ Work***

   If no such i exists, go to step 4.

3. ***Work = Work + Allocation i***

Finish[i] = true

Go to step 2.

4. If ***Finish[i]*** = ***true*** some i, 0 ***≤ i < n*** then the system is in a deadlocked state. Moreover, if ***Finish[i] == false***, then process ***Pi*** is deadlocked

Example of Deadlock detection

A system with:
-5 processes P_0 through P_4;
-3 resource types (A, B, C).
-*A* (7 instances), *B* (2 instances), and *C* (6 instances)
Suppose that, at time $\mathbf{T_0}$, we have the following resource-allocation state:

Process	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 2	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

ملاحظة : هنا في السؤال يعطى التسلسل $<P_0, P_2, P_3, P_1, P_4>$ **مثلاً ويطلب كشف هل ان النظام يكون في** حالة الـ Deadlock أو لا

Example of Deadlock detection Cont.

الحل:

عندما نلاحظ التسلسل $<P_0, P_2, P_3, P_1, P_4>$ فنبدأ ب Process P0 , نلاحظ ان الـ Request له (0 0 0) فيستطيع التنفيذ بالموارد المتوفرة لدية وبعد انتهاء التنفيذ يرجع الموارد التي كانت محجوزة لديه.

New available=Available + Allocation

0 1 0 = 0 0 0 + 0 1 0

نستمر بالتسلسل Process P2 , نلاحظ ان الـ Request له (0 0 0) فيستطيع التنفيذ بالموارد المتوفرة لديه وبعد انتهاء التنفيذ يرجع الموارد التي كانت محجوزة لديه.

New available=Available + Allocation

3 1 2 = 0 1 0 + 3 0 2

نستمر بالتسلسل Process P3 , نلاحظ ان الـ Request له (1 0 0) فيتم اعطاء الموارد التي يحتاجها ويبدا بالتنفيذ وبعد انتهاء التنفيذ يرجع الموارد التي كانت محجوزة لديه.

New available=Available + Allocation

5 2 3 = 3 1 2 + 2 1 1

نستمر بالتسلسل Process P1 , نلاحظ ان الـ Request له (2 0 2) فيتم اعطاء الموارد التي يحتاجها ويبدا بالتنفيذ وبعد انتهاء التنفيذ يرجع الموارد التي كانت محجوزة لديه.

New available=Available + Allocation

7 2 3 = 5 2 3+ 2 0 0

نستمر بالتسلسل Process P4 , نلاحظ ان الـ Request له (0 0 2) فيتم اعطاء الموارد التي يحتاجها ويبدا بالتنفيذ وبعد انتهاء التنفيذ يرجع الموارد التي كانت محجوزة لديه.

New available=Available + Allocation

7 2 5 = 7 2 3+ 0 0 2

Example of Deadlock detection Cont.

We claim that the system is not in a deadlocked state. Indeed, if we execute our algorithm, we will find that the sequence **<P_0, P_2, P_3, P_1, P_4>** results in ***Finish[i]*** = true for all *i*.

النظام هنا ليس في حالة الـ Deadlock .
اذا تم تنفيذ الخوارزمية نجد ان التسلسل **<P_0, P_2, P_3, P_1, P_4>** ينتج عنه ان الـ
Finish[i] = true لكل i

Example Cont.

Suppose now that process P_2 makes one additional request for an instance of type C (0 0 1). The Request matrix is modified as follows:

نفرض الان ان الـ process P_2 عمل طلب اضافي نوع واحد من المورد C بمعنى (0 0 1) فمصفوفة الـ Request سوف تحدث كما ادناه.

Process	Request
	A B C
P0	0 0 0
P1	2 0 2
P2	**0 0 1**
P3	1 0 0
P4	0 0 2

التسلسل او الترتيب هو نفس القديم $<P_0, P_2, P_3, P_1, P_4>$

هنا النظام سوف يدخل في حالة اللـ Deadlock, بالرغم من انه سوف ينفذ P1 وتصبح عدد الموارد الجديد(0 1 0) = New available , لكن الـ Request للـ process P2 هو (0 0 1) لايمكن توفير الموارد التي يحتاجها وبذلك يدخل النظام في حالة الـ Deadlock في الـ process P2 .

We claim that the system is now deadlocked. Although we can reclaim the resources held by process P_0, the number of available resources is not sufficient to fulfill the requests of the other processes.

Thus, a deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.

6.7. Recovery from Deadlock

- **When a detection algorithm determines that a deadlock exists, several alternatives are available.**

- One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually.

- Another possibility is to let the system **recover** from the deadlock automatically.

عندما تحدد خوارزمية الكشف وجود حالة Deadlock، تتوفر عدة بدائل.

-أحد الاحتمالات هو إبلاغ المشغل بحدوث حالة Deadlock والسماح للمشغل بالتعامل مع حالة Deadlock يدوياً .

-هناك إمكانية أخرى تتمثل في السماح للنظام بالتعافي من حالة الـ Deadlock تلقائيًا

- There are two options for breaking a deadlock (methods).

1. One is simply to abort one or more processes to break the circular wait.
2. The other is to preempt some resources from one or more of the deadlocked processes.

- هناك خياران لكسر ال Deadlock

1. واحد هو ببساطة تجاوز(abort) process واحدة أو أكثر لكسر الـ circular wait.
2. والآخر هو أستقطاع (استباق) بعض الموارد من واحدة أو أكثر من الـ processes التي حدث فيها الـ Deadlock

6.7.1. Process termination

- To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.
  1. Abort all deadlocked processes.
  2. Abort one process at a time until the deadlock cycle is eliminated.

لإزالة الـ Deadlock عن طريق تجاوز او اهمال process ، نستخدم إحدى الطريقتين .في كلتا الطريقتين ، يسترد النظام جميع الموارد المخصصة لل processes التي حدث فيها الـ Deadlock

1. تجاوز او اهمال جميع الـ processes التي حدث فيها الـ Deadlock .
2. تجاوز او اهمال process واحدة في كل مرة حتى يتم التخلص من دورة الـ Deadlock .

- Aborting a process may not be easy. If the process was in the midst of updating a file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the midst of printing data on a printer, the system must reset the printer to a correct state before printing the next job.

- تجاوز او اهمال process قد لا تكون سهلة. إذا كانت الـ process في خضم تحديث ملف ، فإن إنهاءه سيترك هذا الملف في حالة غير صحيحة. وبالمثل ، إذا كانت الـ process في خضم بيانات الطباعة على الطابعة ، فيجب على النظام إعادة تعيين الطابعة إلى الحالة الصحيحة قبل طباعة المهمة التالية.

6.7.1. Process termination Cont.

- If the partial termination method is used, then we must determine which deadlocked process (or processes) should be terminated. This determination is a policy decision, similar to CPU-scheduling decisions.
- إذا تم استخدام طريقة الإنهاء الجزئي ، فيجب علينا تحديد أي process أو processes التي حدث فيها الت Deadlock يجب أن يتم إنهاؤها. هذا التحديد هو قرار سياسة ، مشابه لقرارات جدولة وحدة المعالجة المركزية

- Many factors may affect which process is chosen, including:
- هنالك عدة معايير تؤثر على اختيار الـ process لغرض انهاءه

1. What the priority of the process is ?
2. How long the process has computed and how much longer the process will compute before completing its designated task
3. How many and what types of resources the process has used (for example, whether the resources are simple to preempt)
4. How many more resources the process needs in order to complete
5. How many processes will need to be terminated
6. Whether the process is interactive or batch

End of Chapter Six