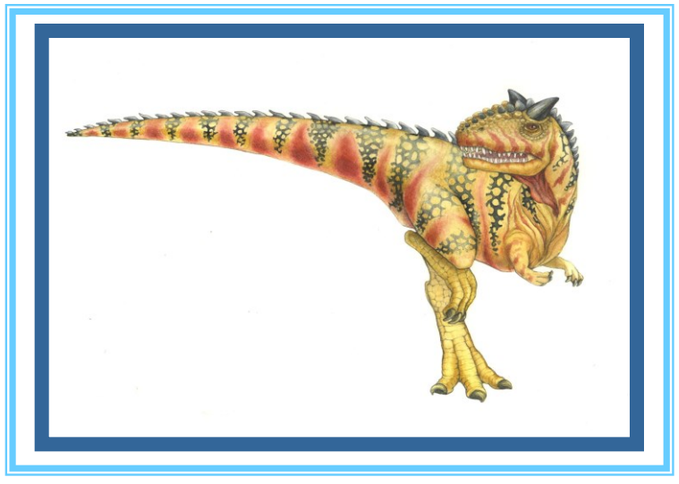
Chapter 3: Processes



Silberschatz, Galvin and Gagne ©2013



Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks or Active program
- Textbook uses the terms job and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts

  Process = برنامج قيد التنفيذ

  يحتوي على عدة أجزاء :
  - The program code, also called text section
  - Current activity including program counter, processor registers
  - Stack containing temporary data
    - Function parameters, return addresses, local variables
  - Data section containing global variables
  - Heap containing memory dynamically allocated during run time
- Program is ***passive*** entity stored on disk (executable file), process is ***active***
  - Program becomes process when executable file loaded into memory {Multi}
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes {how example}
  - Consider multiple users executing the same program

كل Process يحتوي عدة اجزاء في الذاكرة:

1 Text Section

يسمى ايضاً Program Code

وهو الكود نفسه.

مثال:

تعليمات البرنامج المكتوبة بلغة C او Java.

2 Program Counter + Registers

Program Counter

يحتفظ بعنوان التعليمة التالية التي سيتم تنفيذها.

Registers

هي مسجلات المعالج التي تخزن معلومات مؤقتة اثناء التنفيذ.

3 Stack

يحتوي بيانات مؤقتة مثل:

Function parameters Return addresses Local variables

مثال:

void sum(int a,int b)

القيم a و b تخزن في الـ Stack.

4 Data Section

تحتوي المتغيرات العامة (Global Variables).

مثال: int total = 0;

5 Heap

الـ Heap يستخدم للذاكرة التي يتم حجزها اثناء تشغيل البرنامج.

يسمى: Dynamic Memory Allocation

مثال:

الفرق بين Program و Process

Program

برنامج ساكن (Passive)

يعني مجرد ملف موجود على القرص (Disk).

مثال: word.exe

هو مجرد ملف.

Process

برنامج نشط (Active)

يعني يتم تنفيذه داخل الذاكرة.

البرنامج يتحول إلى Process عندما:

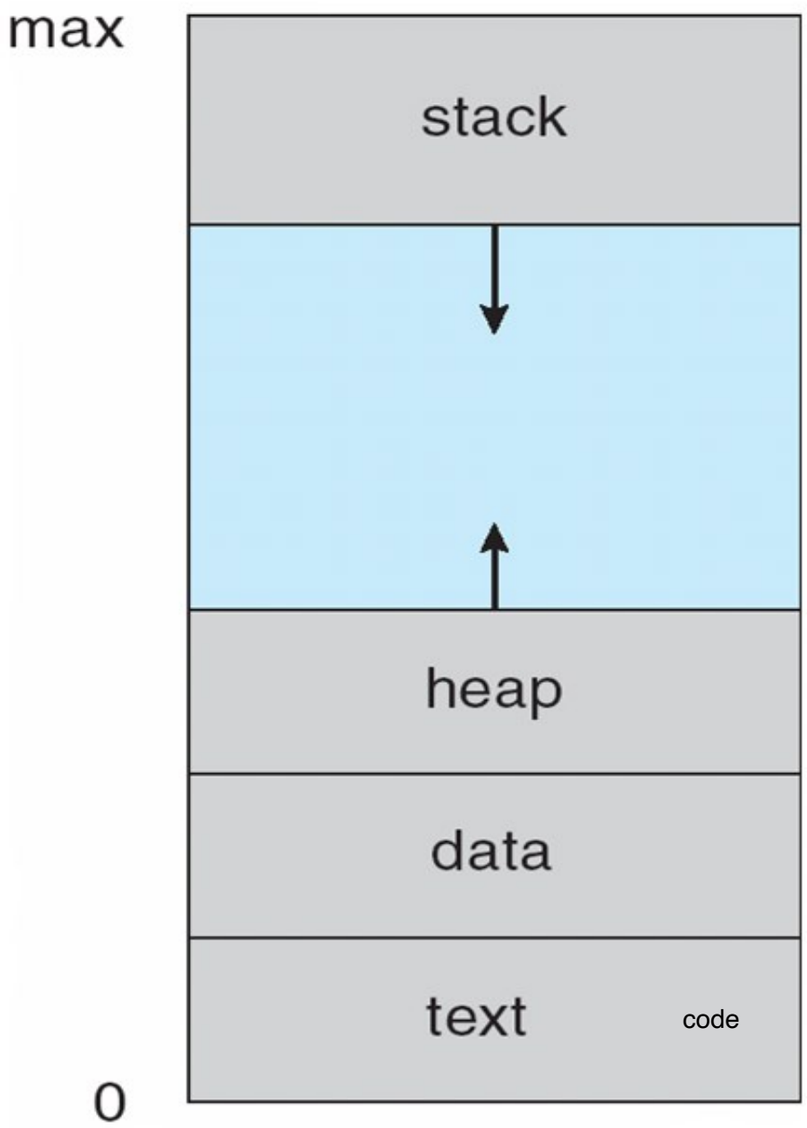
نقوم بتشغيله (load into memory)





Process in Memory

شكل الـ Process في الذاكرة :









Process State

- As a process executes, it changes **state**
  - **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 {Memory}
  - **terminated**: The process has finished execution (Passive/program}

يعني أن الـ Process يتم إنشاؤها الآن.
📌 ماذا يحدث هنا؟
المستخدم يشغل برنامج
النظام يبدأ بإنشاء Process له
يتم تخصيص الذاكرة له
مثال:
عندما تضغط على برنامج Word
نظام التشغيل يبدأ إنشاء Process جديدة.

الـ Process جاهزة للتنفيذ لكن تنتظر المعالج (CPU).
📌 هنا:
البرنامج موجود في الذاكرة (Memory)
لكنه ينتظر دوره ليأخذ CPU
لأن المعالج لا يستطيع تنفيذ أكثر من Process في نفس اللحظة.
مثال:
عندك مفتوح:
Chrome
Word
Spotify
كلهم في Ready Queue ينتظرون المعالج.

هنا الـ Process فعلاً يتم تنفيذها بواسطة المعالج.
📌 يعني:
الـ CPU الآن ينفذ تعليمات البرنامج.
مثال:
عندما تكتب في Word
المعالج ينفذ أوامر البرنامج.

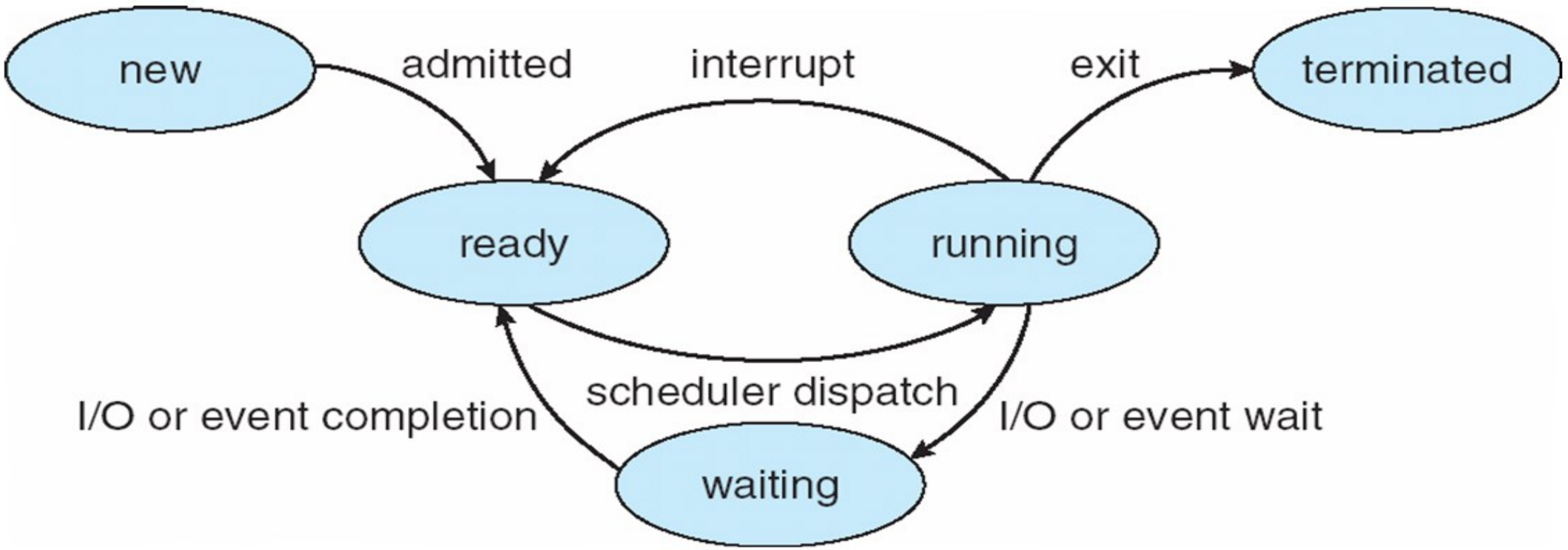
الـ Process تنتظر حدث معين قبل أن تكمل التنفيذ.
مثل:
قراءة ملف
إدخال من المستخدم
وصول بيانات من الشبكة
مثال:
برنامج ينتظر أن تضغط زر Enter.
هنا ينتقل إلى Waiting.

هنا البرنامج انتهى من التنفيذ.
📌 ماذا يحدث بعدها؟
النظام يحرر الذاكرة
يتم حذف الـ Process
ويعود البرنامج مجرد Program على القرص وليس Process.





Diagram of Process State





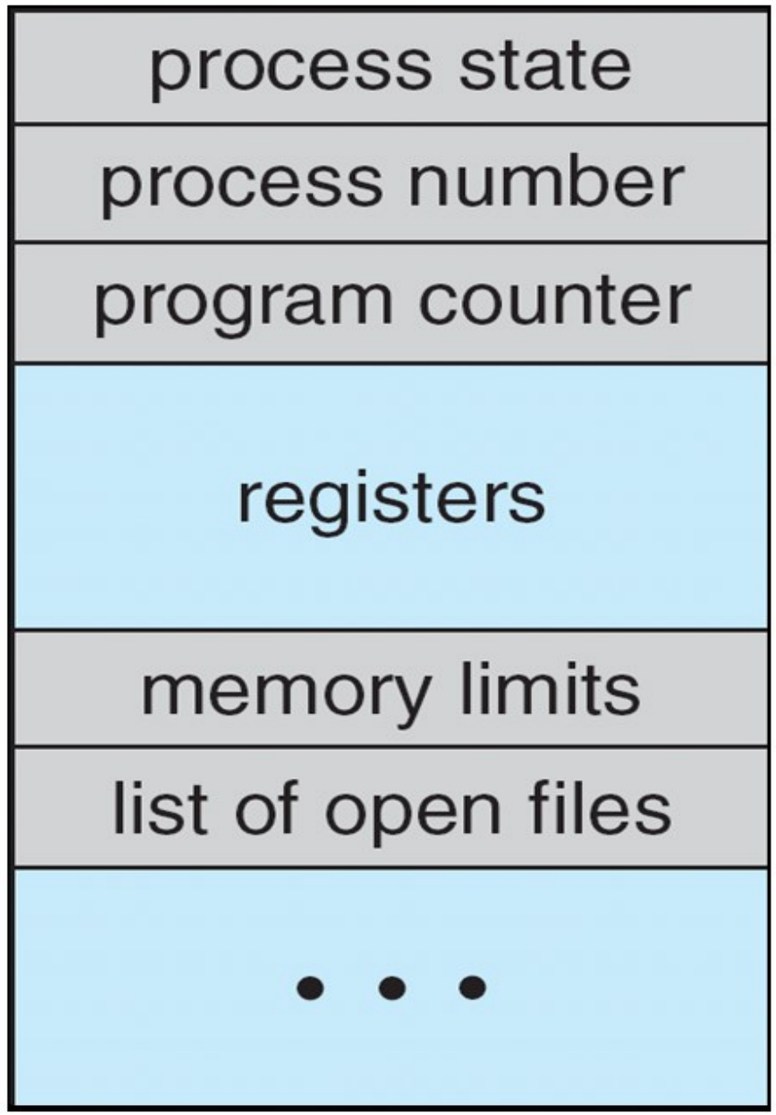


Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



Process Control Block (PCB)
الـ PCB هو هيكل بيانات (Data Structure) يستخدمه نظام التشغيل ليخزن كل المعلومات الخاصة بكل Process.
يسمى أحياناً أيضاً: Task Control Block (TCB)

📌 الفكرة الأساسية:
عندما ينشئ النظام Process جديدة فإنه ينشئ لها PCB خاص بها يحتوي كل المعلومات التي يحتاجها النظام لإدارة هذه العملية.

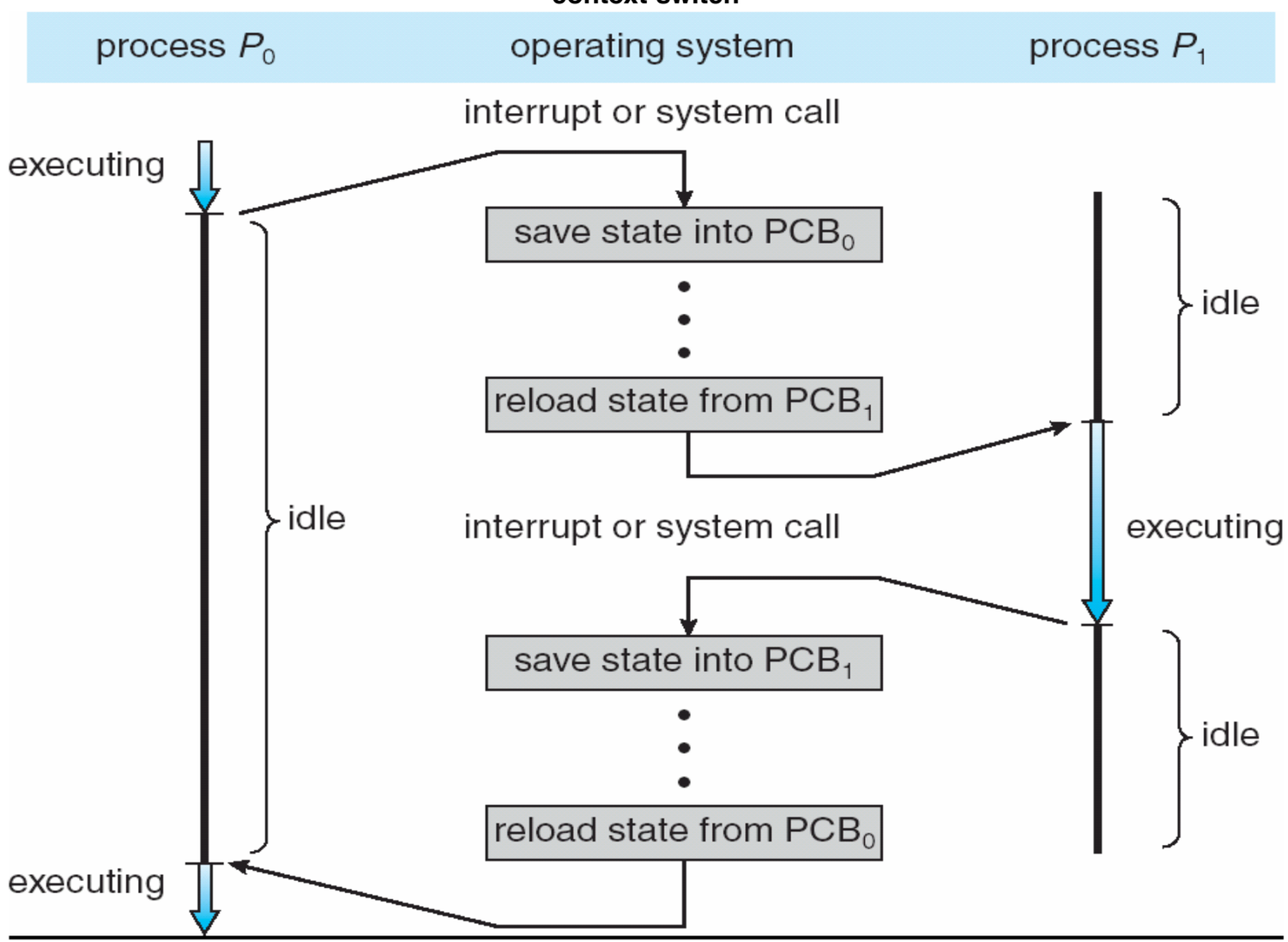
لماذا نحتاج PCB؟
نظام التشغيل قد يشغل عشرات أو مئات العمليات في نفس الوقت.
لكي يستطيع تتبع كل Process يحتاج مكان يخزن فيه معلوماتها، هذا المكان هو PCB.

المعلومات	وظيفتها
Process State	حالة العملية
Program Counter	التعليمة القادمة
CPU Registers	قيم المسجلات
Scheduling Info	معلومات الجدولة
Memory Info	معلومات الذاكرة
Accounting Info	إحصائيات الاستخدام
I/O Status	الأجهزة والملفات





CPU Switch From Process to Process



States Q

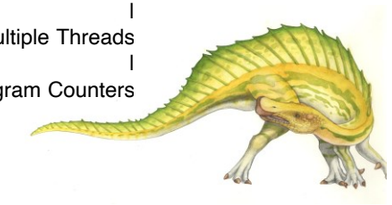




Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control -> threads
- Must then have storage for thread details, multiple program counters in PCB
- See next chapter

ما هي Thread؟ جزء من الـ Process يمثل مسار تنفيذ (Execution Path) داخل البرنامج.
بمعنى آخر: Thread = وحدة تنفيذ داخل الـ Process.
الحالة التي درسناها سابقاً
في الشابتر إلى الآن:
كل Process تحتوي:
Process
|
Thread
|
Execution
يعني: تنفيذ واحد في نفس الوقت
ماذا لو كان لدينا أكثر من Thread؟
يمكن أن تحتوي الـ Process على عدة Threads.
وهذا يعني:
عدة Program Counters
عدة أماكن تنفيذ في نفس الوقت
النص يقول:
Multiple locations can execute at once
يعني أجزاء مختلفة من البرنامج تعمل معاً.
مثال بسيط
متصفح الإنترنت.
مثلاً لو فتحت: تحميل ملف، تشغيل فيديو، فتح صفحة جديدة
المتصفح يستخدم Threads متعددة.
مثال:
Process: Browser
تحميل ملف → Thread 1
تشغيل الفيديو → Thread 2
عرض الصفحة → Thread 3
كل Thread تنفذ جزء من البرنامج.
لماذا نستخدم Threads؟
لأنها تجعل البرنامج:
✅ أسرع
✅ أكثر كفاءة
✅ يستطيع تنفيذ عدة مهام معاً
مثال:
برنامج Word
الكتابة → Thread 1
التدقيق الإملائي → Thread 2
الحفظ التلقائي → Thread 3
كلها تعمل في نفس الوقت.
ماذا يحدث داخل الـ PCB؟ لأن لدينا عدة Threads داخل Process واحدة:
يجب أن نخزن معلومات لكل Thread مثل:
Program Counter, Registers, Thread state
لذلك يصبح لدينا:
Process
|
PCB
|
Multiple Threads
|
Program Counters





Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler (S/W):** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute // {belong to}
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues // (State diagram /Process life cycle)

هدف Process Scheduling هو:
استخدام CPU بأكبر كفاءة
التبديل بسرعة بين العمليات (Time Sharing)
Process Scheduler
برنامج في نظام التشغيل يقوم بـ: اختيار أي Process تعمل على الـ CPU.

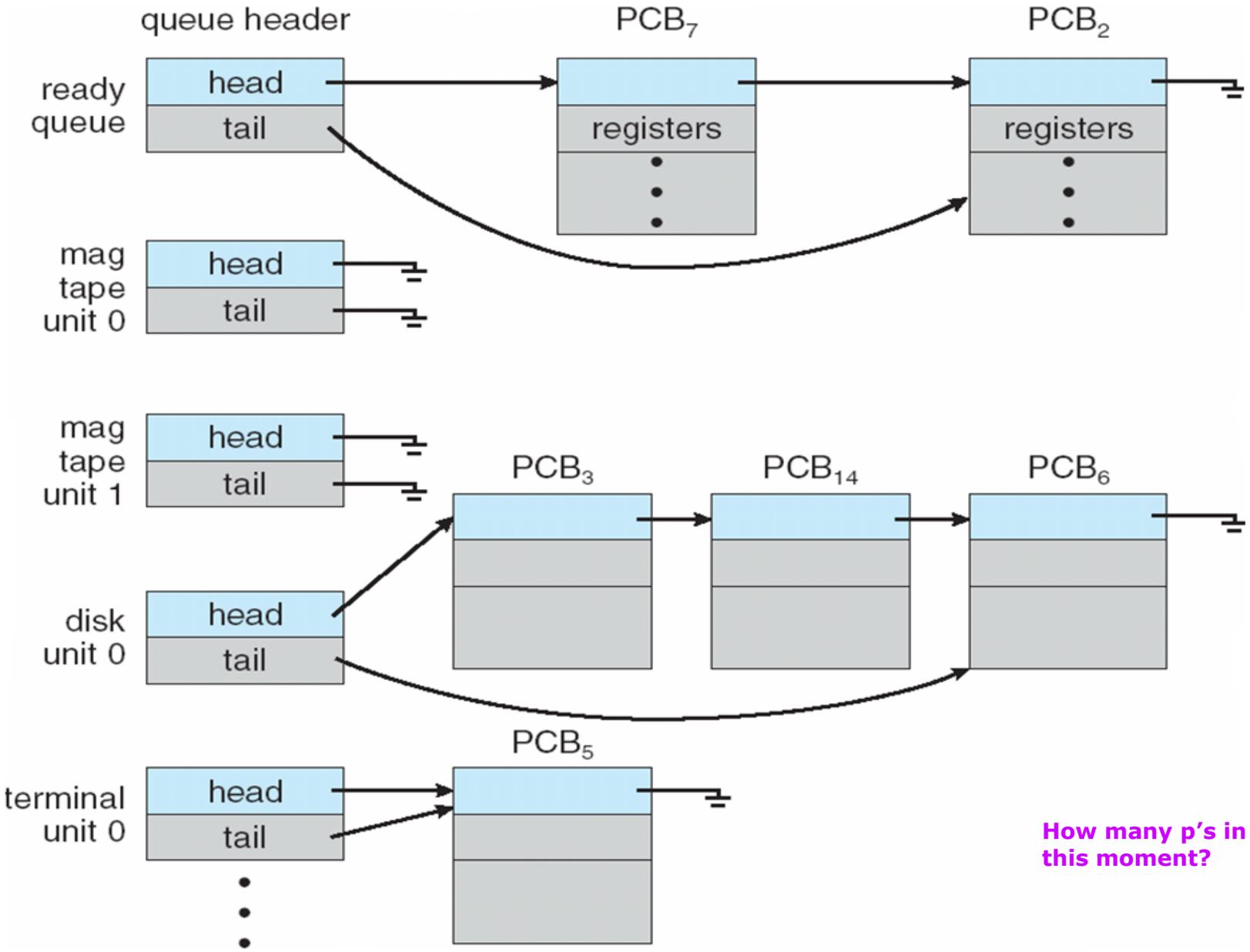
Scheduling Queues:
1 Job Queue
جميع العمليات الموجودة في النظام.
2 Ready Queue
العمليات الموجودة في الذاكرة وجاهزة للتنفيذ لكنها تنتظر CPU.
3 Device Queue
العمليات التي تنتظر أجهزة I/O مثل:
القرص
الطابعة

الفكرة الأساسية:
الـ Process تنتقل بين هذه الـ Queues حسب حالتها أثناء التنفيذ.





Ready Queue And Various I/O Device Queues

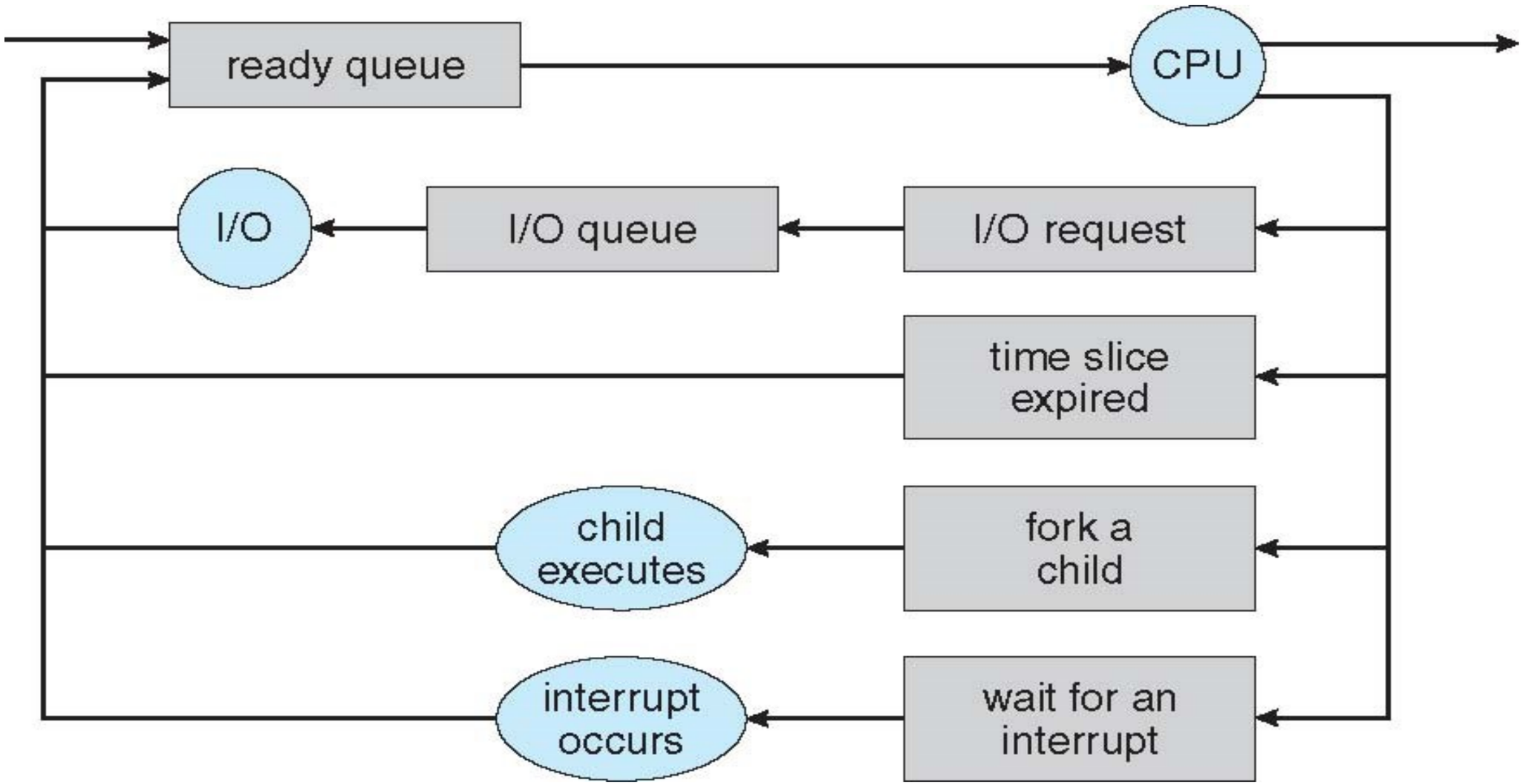


How many p's in the system this moment?



Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows





Schedulers

- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue

اختيار أي Process تدخل إلى Ready Queue من الـ Job Queue.

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU

اختيار أي Process من Ready Queue ستعمل على الـ CPU.

  - Sometimes the only scheduler in a system

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)

متكرر جداً — يجب ان تكون سريعة

- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)

غير متكرر — ممكن ان تكون بطيئة

- The long-term scheduler controls the **degree of multiprogramming**

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

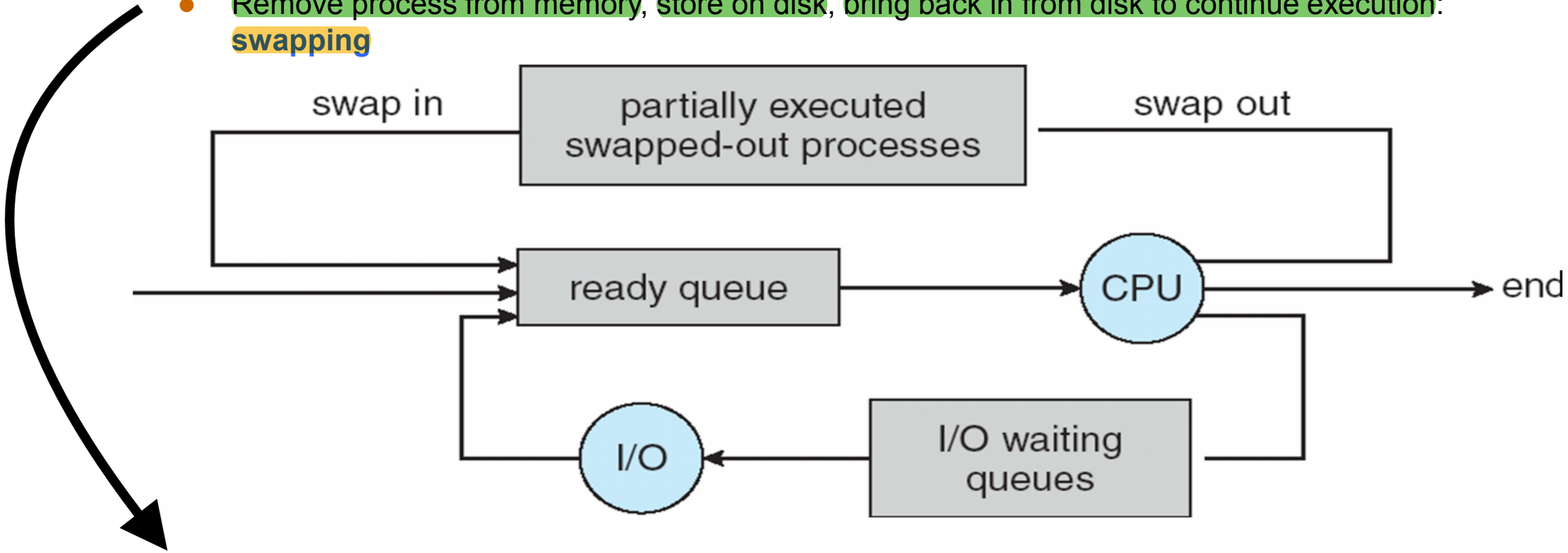
أنواع العمليات
I/O-Bound Process
تقضي وقتها أكثر في عمليات الإدخال والإخراج.
خصائصها:
قصيرة CPU bursts
تنتظر I/O كثير
مثال: برنامج قراءة ملفات.
CPU-Bound Process
تقضي وقتها أكثر في الحسابات.
خصائصها:
طويلة CPU bursts
عمليات I/O قليلة
مثال: برنامج حسابات رياضية ثقيلة.





Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



يتم استخدام Medium-Term Scheduler عندما يريد نظام التشغيل تقليل عدد العمليات الموجودة في الذاكرة.
وظيفته
يقوم بـ:
إخراج Process من الذاكرة (Memory)
حفظها في القرص (Disk)
ثم إعادتها لاحقاً لتكمل التنفيذ
هذه العملية تسمى: Swapping

Merge F13& F15 in one single Figure





Multitasking in Mobile Systems

IOS

- Some systems / early systems allow only one process to run, others suspended
- Due to screen real estate, user interface limits iOS provides for a
  - Single **foreground** process- controlled via user interface
  - Multiple **background** processes– in memory, running, but not on the display, and with limits
  - Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback.

{ refers back to page 37}

Android

- Android runs foreground and background, with fewer limits
  - Background process uses a **service** to perform tasks
  - Service can keep running even if background process is suspended
  - Service has no user interface, small memory use

Two operating systems currently dominate mobile computing: Apple iOS and Google Android. iOS was designed to run on Apple iPhone and iPad mobile devices. Android powers smartphones and tablet computers available from many manufacturers. We examine these two mobile operating systems in further detail in Chapter 2.

بعض الأنظمة القديمة كانت تسمح بتشغيل عملية واحدة فقط، بينما تكون العمليات الأخرى موقوفة (Suspended).
بسبب صغر شاشة الجوال وقيود واجهة المستخدم، يعمل نظام iOS كالتالي:
العملية الأمامية (Foreground Process)
يوجد برنامج واحد فقط يظهر على الشاشة ويتحكم به المستخدم.
العمليات الخلفية (Background Processes)
يمكن أن توجد عدة برامج تعمل في الخلفية داخل الذاكرة لكنها لا تظهر على الشاشة، ولها بعض القيود مثل:
تنفيذ مهام قصيرة
استقبال الإشعارات
تشغيل مهام طويلة محددة مثل تشغيل الصوت

أما نظام Android فهو أكثر مرونة:
يسمح بوجود عمليات أمامية وخلفية بقيود أقل.
ويستخدم مكوّن يسمى Service.
Service
هو برنامج يعمل في الخلفية:
بدون واجهة مستخدم
يستهلك ذاكرة قليلة
يمكن أن يستمر في العمل حتى لو تم إيقاف البرنامج الأساسي
مثال:
تشغيل الموسيقى في الخلفية أثناء استخدام تطبيق آخر.





Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**

يحدث Context Switch عندما يقوم المعالج CPU بالانتقال من تنفيذ Process إلى تنفيذ Process أخرى.
عند حدوث ذلك يجب على نظام التشغيل:
حفظ حالة العملية الحالية
تحميل حالة العملية الجديدة
وذلك حتى تستطيع العملية الجديدة أن تكمل التنفيذ بشكل صحيح.

- **Context** of a process represented in the PCB

يتم حفظ حالة العملية (Context) داخل PCB (Process Control Block).
تشمل المعلومات مثل:
Program Counter
CPU Registers
Process State

- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch

وقت تبديل السياق (Context Switch Time)
الوقت الذي يستغرقه النظام أثناء التبديل يسمى Context Switch Time.
هذا الوقت يعتبر Overhead لأن النظام:
لا ينفذ أي عمل مفيد أثناء التبديل.
كلما كان:
نظام التشغيل أكثر تعقيدًا
و محتوى PCB أكبر
فإن وقت التبديل يزداد.
ماذا يؤثر على سرعة التبديل؟

- Time dependent on hardware support

تعتمد سرعة Context Switch على:
قدرات العتاد (Hardware Support)
فبعض المعالجات تدعم عمليات التبديل بشكل أسرع.





Operations on Processes

- **System must provide mechanisms for process creation, termination, and so on as detailed next.**

- **Process Creation** : **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes

- Generally, process identified and managed via a **process identifier (pid)**

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

- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

العمليات على الـ Process (Operations on Processes)

يجب على نظام التشغيل أن يوفر آليات لـ:

إنشاء Process

إنهاء Process

وإدارة العمليات بشكل عام.

إنشاء العملية (Process Creation)

عند إنشاء Process جديدة:

تسمى العملية الأصلية Parent Process

والعملية الجديدة تسمى Child Process

ويمكن لـ Child Process أن تنشئ عمليات أخرى.

لذلك تتشكل شجرة من العمليات (Process Tree).

معرّف العملية (Process Identifier)

كل Process يتم التعرف عليها باستخدام رقم خاص يسمى:

PID (Process Identifier)

يستخدمه نظام التشغيل لإدارة العمليات.

مشاركة الموارد (Resource Sharing)

هناك عدة طرق لمشاركة الموارد بين Parent و Child:

مشاركة جميع الموارد بين الأب والأبناء.

مشاركة جزء من الموارد فقط.

عدم مشاركة أي موارد.

خيارات التنفيذ (Execution Options)

هناك طريقتان لتنفيذ العمليات:

1. **التنفيذ المتزامن (Concurrent Execution)**

يعمل Parent و Child في نفس الوقت.

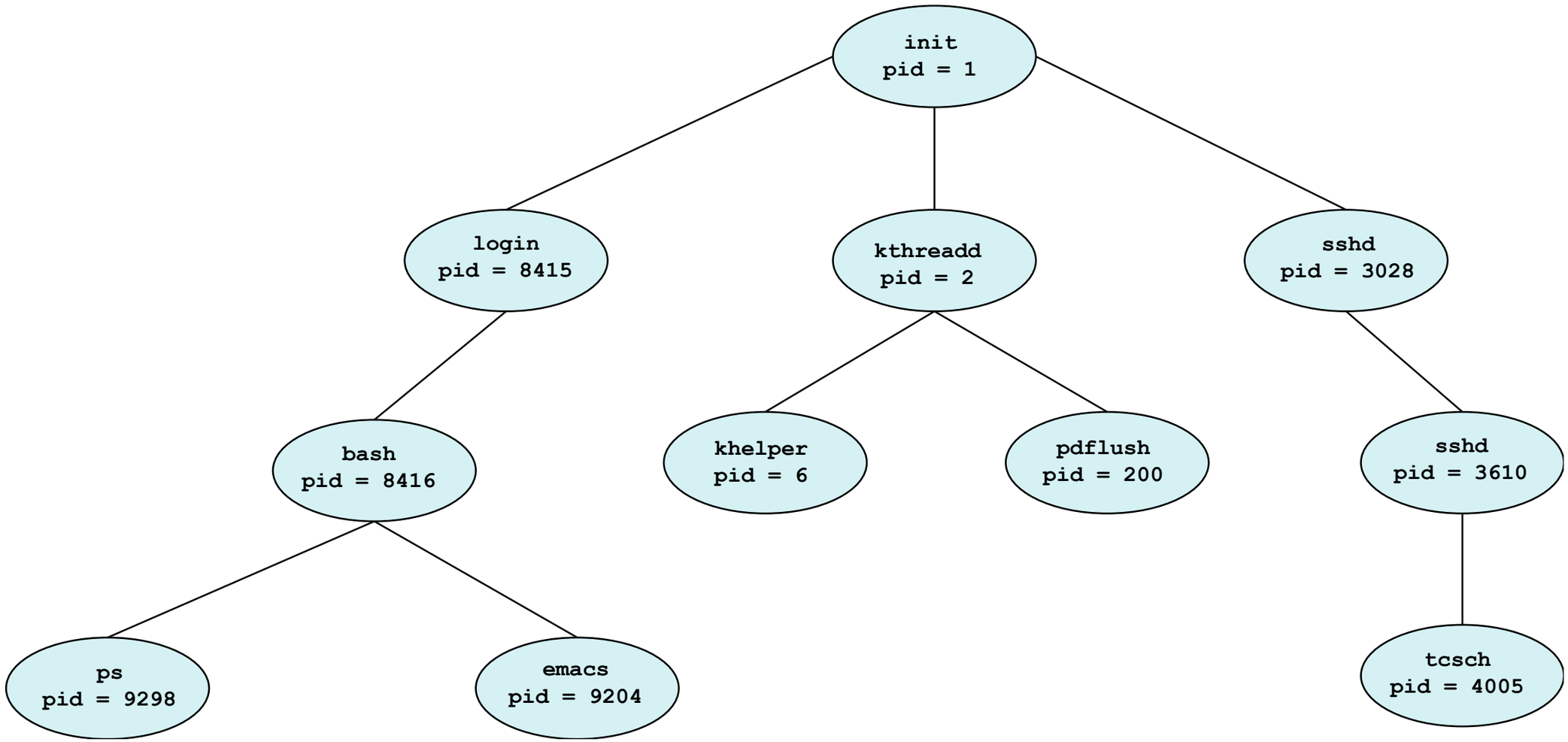
2. **الانتظار (Parent Wait)**

ينتظر Parent Process حتى تنتهي Child Process من التنفيذ.





A Tree of Processes in Linux





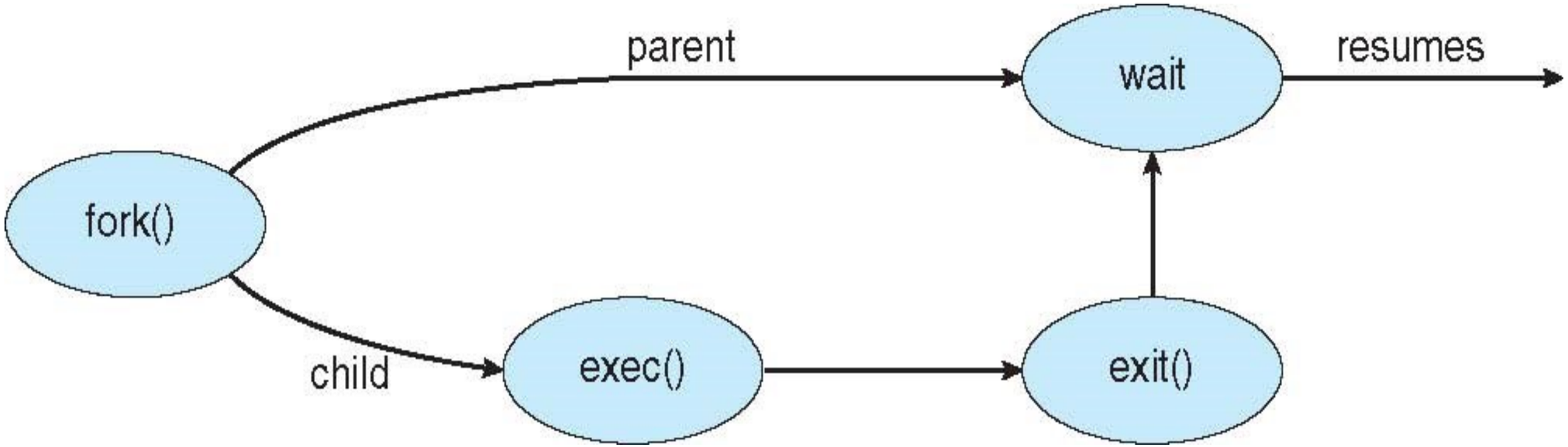


Process Creation (Cont.)

مساحة العناوين (Address Space)
عند إنشاء Child Process يمكن أن يحدث أحد الخيارين:
الـ Child تكون نسخة من الـ Parent
أي أن مساحة الذاكرة للطفل تكون مطابقة للأب.
تحميل برنامج جديد داخل الـ Child
أي أن العملية الجديدة تنفذ برنامج مختلف.

في أنظمة UNIX يتم إنشاء العمليات باستخدام أوامر تسمى System Calls.
fork()
هي System Call تستخدم لإنشاء Process جديدة.
العملية الجديدة تسمى Child Process
تكون نسخة من Parent Process في البداية.
موجودة في أنظمة:
UNIX
Linux
macOS
exec()
تستخدم بعد fork().
وظيفتها:
استبدال ذاكرة العملية ببرنامج جديد.
أي أن العملية ستبدأ بتنفيذ برنامج مختلف.

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - `fork()` system call creates new process{ what is fork and in which os is found}
  - `exec()` system call used after a `fork()` to replace the process' memory space with a new program







C Program Forking Separate Process

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```





Creating a Separate Process via Windows API

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    /* allocate memory */
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    /* create child process */
    if (!CreateProcess(NULL, /* use command line */
     "C:\\WINDOWS\\system32\\mspaint.exe", /* command */
     NULL, /* don't inherit process handle */
     NULL, /* don't inherit thread handle */
     FALSE, /* disable handle inheritance */
     0, /* no creation flags */
     NULL, /* use parent's environment block */
     NULL, /* use parent's existing directory */
     &si,
     &pi))
    {
       fprintf(stderr, "Create Process Failed");
       return -1;
    }
    /* parent will wait for the child to complete */
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    /* close handles */
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```





Process Termination

- Process executes last statement and asks the operating system to delete it (`exit()`) what is the commend?
  - Output data from child to parent (via `wait()`)
  - Process' resources are deallocated by operating system

عندما تنتهي Process من تنفيذ آخر تعليمة، تطلب من نظام التشغيل حذفها باستخدام الأمر: exit()
وهو System Call يستخدمه البرنامج ليخبر نظام التشغيل أنه انتهى من التنفيذ.
بعد ذلك يقوم نظام التشغيل بـ:
تحرير الموارد الخاصة بالعملية (الذاكرة، الملفات، وغيرها).

- Parent may terminate execution of children processes (`abort()`) when the abort command happened by parent?
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating systems do not allow child to continue if its parent terminates
      - All children terminated - **cascading termination**

يمكن لـ Child Process إرسال نتيجة التنفيذ إلى Parent Process باستخدام:
()wait
الأب ينتظر انتهاء الابن ثم يستقبل:
PID
حالة التنفيذ (status)

يمكن لـ Parent Process إنهاء Child Process باستخدام: abort()
يحدث ذلك في الحالات التالية:
إذا تجاوز الابن الموارد المسموح بها
إذا لم تعد المهمة المطلوبة منه ضرورية
إذا انتهى Parent Process

Cascading Termination
بعض أنظمة التشغيل لا تسمح لـ Child Process بالاستمرار إذا انتهى الأب.
لذلك يتم: إنهاء جميع العمليات الأبناء تلقائيًا.
يسمى هذا: Cascading Termination

- Wait for termination, returning the pid:

```
pid t_pid; int status;
pid = wait(&status);
```

- If no parent waiting, then terminated process is a **zombie***
- If parent terminated, processes are **orphans** page 121

Zombie Process
تحدث عندما:
تنتهي Child Process
لكن Parent Process لم ينفذ wait() بعد
فتبقى العملية في النظام بحالة تسمى: Zombie
أي عملية انتهت لكنها ما زالت مسجلة في النظام.

Orphan Process
تحدث عندما:
ينتهي Parent Process
بينما Child Process ما زالت تعمل
فتصبح العملية:
Orphan Process (عملية يتيمة).
عادة يقوم النظام بإسنادها إلى عملية أخرى في النظام.

* A process that has terminated, but whose parent has not yet called wait(), is known as a **zombie** process.

* Now consider what would happen if a parent did not invoke wait() and instead terminated, thereby leaving its child processes as **orphans**.





Multiprocess Architecture – Chrome Browser

بنية تعدد العمليات في متصفح Chrome (Multiprocess Architecture)
في الماضي كانت بعض المتصفحات تعمل كـ Process واحدة فقط.
المشكلة:
إذا حدث خطأ في موقع واحد فقد يتوقف أو ينهار المتصفح بالكامل.
في متصفح Google Chrome
يستخدم Multiprocess Architecture
أي أن المتصفح يعمل باستخدام عدة Processes مختلفة.

- Many web browsers ran as single process (some still do)
  - If one web site causes trouble, entire browser can hang or crash
- Google Chrome Browser is multiprocess with 3 categories
  - **Browser** process manages user interface, disk and network I/O
  - **Renderer** process renders web pages, deals with HTML, Javascript, new one for each website opened
    - Renderer process runs in **sandbox** restricting disk and network I/O, minimizing effect of security exploits
  - **Plug-in** process for each type of plug-in page 123

العملية	وظيفته
Browser Process	إدارة واجهة المستخدم والقرص والشبكة
Renderer Process	عرض صفحات الويب وتشغيل JavaScript و HTML
Plug-in Process	تشغيل إضافات المتصفح







Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

العمليات داخل النظام قد تكون:
1 Independent Processes
عمليات مستقلة لا تؤثر على بعضها ولا تتشارك البيانات.

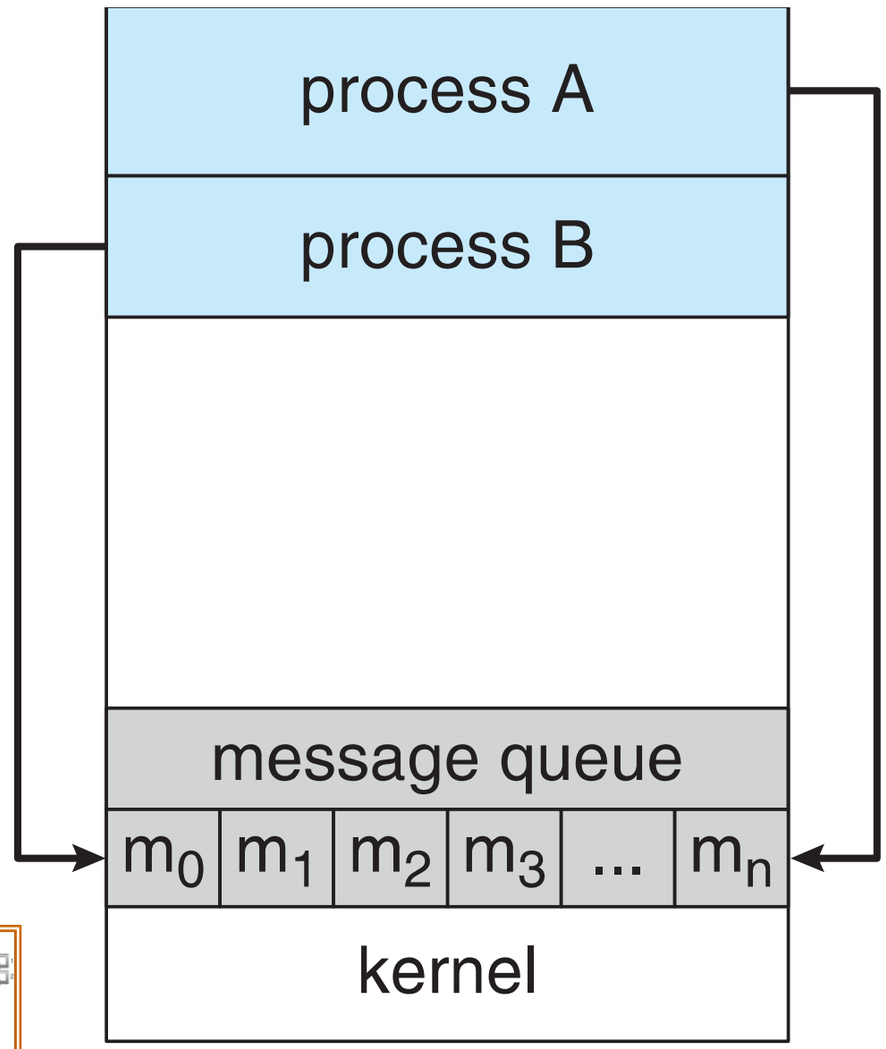
2 Cooperating Processes
عمليات متعاونة يمكن أن:
تؤثر على عمليات أخرى
تتأثر بها
تشارك البيانات معها.

لماذا نستخدم العمليات المتعاونة؟
هناك عدة أسباب:
Information Sharing
مشاركة المعلومات بين العمليات.
Computation Speedup
تسريع تنفيذ العمليات بتقسيم العمل بين عدة عمليات.
Modularity
تقسيم النظام إلى أجزاء أصغر وأسهل للإدارة.
Convenience
تشغيل عدة مهام معاً لراحة المستخدم.

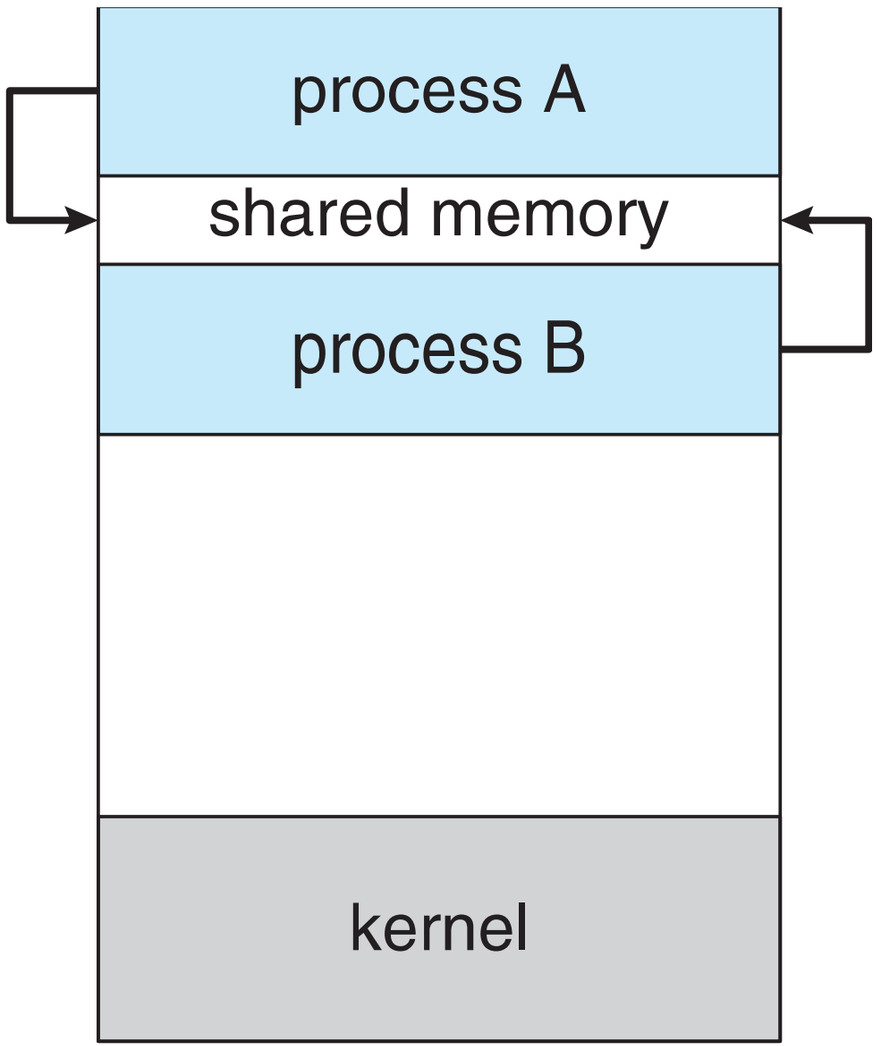
عندما تتعاون العمليات فهي تحتاج طريقة للتواصل تسمى:
IPC (Interprocess Communication)
أي التواصل بين العمليات.
نماذج الاتصال بين العمليات
يوجد طريقتان رئيسيتان:
1 Shared Memory
عدة عمليات تشترك في منطقة ذاكرة واحدة لتبادل البيانات.
2 Message Passing
العمليات تتواصل عن طريق إرسال واستقبال الرسائل بينها.



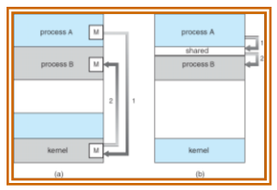
Communications Models



(a)



(b)





Cooperating Processes

- ***Independent*** process cannot affect or be affected by the execution of another process
- ***Cooperating*** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

مکرره





Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - **unbounded-buffer** places no practical limit on the size of the buffer
  - **bounded-buffer** assumes that there is a fixed buffer size

مشكلة المنتج والمستهلك (Producer–Consumer Problem)
هي مثال مشهور على العمليات المتعاونة (Cooperating Processes).
الفكرة:
ينتج بيانات → Producer Process
يستهلك هذه البيانات → Consumer Process
يتم تخزين البيانات في Buffer (مخزن مؤقت) بينهما.
أنواع الـ Buffer
1 Unbounded Buffer
لا يوجد حد عملي لحجم الـ Buffer.
يعني أن الـ Producer يمكنه الاستمرار في إنتاج البيانات بدون توقف.
2 Bounded Buffer
يوجد حجم محدد لـ Buffer.
إذا امتلأ الـ Buffer:
يجب أن ينتظر Producer حتى يستهلك Consumer بعض البيانات.





Bounded-Buffer – Shared-Memory Solution

Q. Explain the function of the following segment

- Shared data

```
#define BUFFER_SIZE 10

typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];

int in = 0;

int out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements





Bounded-Buffer – Producer

Q. Explain the function of the following segment?

```
item next_produced;
while (true) {
    /* produce an item in next produced */
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}
```





Bounded Buffer – Consumer

```
item next_consumed;

while (true) {
     while (in == out)
          ; /* do nothing */
     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;

     /* consume the item in next consumed */
}
```





Interprocess Communication – Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a ***communication link*** between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., direct or indirect, synchronous or asynchronous, automatic or explicit buffering)

هذه الطريقة تستخدم لتواصل العمليات المتعاونة وتنظيم تنفيذها (synchronization) بدون مشاركة الذاكرة.

آلية عمل النظام:

كل عملية تتواصل مع الأخرى عن طريق الرسائل.

لا تحتاج العمليات إلى متغيرات مشتركة.

العمليات الأساسية في IPC :

1️⃣ send(message)

إرسال رسالة إلى عملية أخرى

حجم الرسالة يمكن أن يكون ثابتًا أو متغيرًا.

2️⃣ receive(message)

استقبال رسالة من عملية أخرى.

خطوات التواصل بين عمليتين P و Q

إنشاء رابط اتصال (Communication Link) بينهما.

تبادل الرسائل باستخدام send/receive.

تنفيذ رابط الاتصال

يمكن أن يكون الرابط:

1️⃣ Physical (مادي)

مثل Shared Memory أو Hardware Bus

2️⃣ Logical (منطقي)

عملية إلى عملية مباشرة: Direct

Queue أو Mailbox عبر: Indirect

العملية تنتظر حتى تصل الرسالة: Synchronous

العملية تستمر دون انتظار: Asynchronous

يمكن أن يكون تلقائي أو صريح لتخزين الرسائل مؤقتًا: Buffering





Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

هذه الأسئلة تحدد كيفية تصميم وتنفيذ روابط الرسائل بين العمليات وتؤثر على أداء ومرونة IPC.





Direct Communication

- Processes must name each other explicitly:
  - **`send`** (*P, message*) – send a message to process P
  - **`receive`**(*Q, message*) – receive a message from process Q

- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

في هذا النوع من الاتصال، يجب على العمليات أن تعرف بعضها البعض بشكل صريح.
كيفية إرسال واستقبال الرسائل
send(P, message) → إرسال رسالة إلى العملية P
receive(Q, message) → استقبال رسالة من العملية Q
خصائص رابط الاتصال (Communication Link)
Links are established automatically
الرابط يتم إنشاؤه تلقائيًا عند الحاجة.
A link is associated with exactly one pair of processes
كل رابط يربط زوج واحد فقط من العمليات.
Between each pair there exists exactly one link
بين كل زوج من العمليات يوجد رابط واحد فقط.
Unidirectional or Bi-directional
قد يكون الرابط أحادي الاتجاه (من المرسل إلى المستقبل)
لكنه غالبًا ثنائي الاتجاه، أي يمكن للعمليتين تبادل الرسائل في كلا الاتجاهين.





Indirect Communication

- Messages are directed and received from mailboxes
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox

- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

في هذا النوع من الاتصال، الرسائل تُرسل وتُستقبل عبر صناديق بريد (Mailboxes)، وليس مباشرة بين العمليات.
خصائص الـ Mailboxes
كل Mailbox له معرف فريد (Unique ID).
يمكن للعمليات التواصل فقط إذا كان لديهم صندوق بريد مشترك.
خصائص رابط الاتصال
Link established only if processes share a common mailbox
الرابط يتم إنشاؤه فقط إذا تشارك العمليات نفس الـ Mailbox.
A link may be associated with many processes
يمكن أن يرتبط الرابط بعدة عمليات في نفس الوقت.
Each pair of processes may share several communication links
يمكن لكل زوج من العمليات مشاركة عدة روابط اتصال.
Link may be unidirectional or bi-directional
قد يكون أحادي الاتجاه
أو ثنائي الاتجاه حسب الحاجة.





Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox

- Primitives are defined as:

  `send`(*A, message*) – send a message to mailbox A

  `receive`(*A, message*) – receive a message from mailbox A

الاتصال غير المباشر بين العمليات – العمليات الأساسية (Indirect Communication – Operations)

العمليات على الـ Mailboxes
إنشاء Mailbox جديد
إرسال واستقبال الرسائل من خلال الـ Mailbox
حذف Mailbox عند عدم الحاجة

الدوال الأساسية (Primitives)
send(A, message) → إرسال رسالة إلى الصندوق A
receive(A, message) → استقبال رسالة من الصندوق A





Indirect Communication

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?

- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

مثال:
العمليات P1, P2, P3 تشترك في Mailbox A
يرسل رسالة P1
من سيستقبل الرسالة؟ P2 أم P3؟
الحلول الممكنة:
السماح لكل رابط أن يرتبط بحد أقصى عمليتين فقط
السماح لعملية واحدة فقط بتنفيذ receive في نفس الوقت
السماح للنظام باختيار المستقبل عشوائيًا، مع إعلام المرسل من استلم الرسالة





Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

}

عند استخدام Message Passing، يمكن أن يكون الاتصال:
(Synchronous) متزامن → (محظور) Blocking
(Asynchronous) غير متزامن → (غير محظور) Non-Blocking

1 Blocking / Synchronous
Blocking Send
العملية المرسلة تتوقف حتى يتم استلام الرسالة من قبل المستقبل.
Blocking Receive
العملية المستقبلة تتوقف حتى تصبح هناك رسالة جاهزة للاستلام.

2 Non-Blocking / Asynchronous
Non-Blocking Send
العملية المرسلة ترسل الرسالة وتستمر في التنفيذ دون انتظار المستقبل.
Non-Blocking Receive
العملية المستقبلة تحصل على رسالة صالحة أو null إذا لم تكن هناك رسالة جاهزة.

