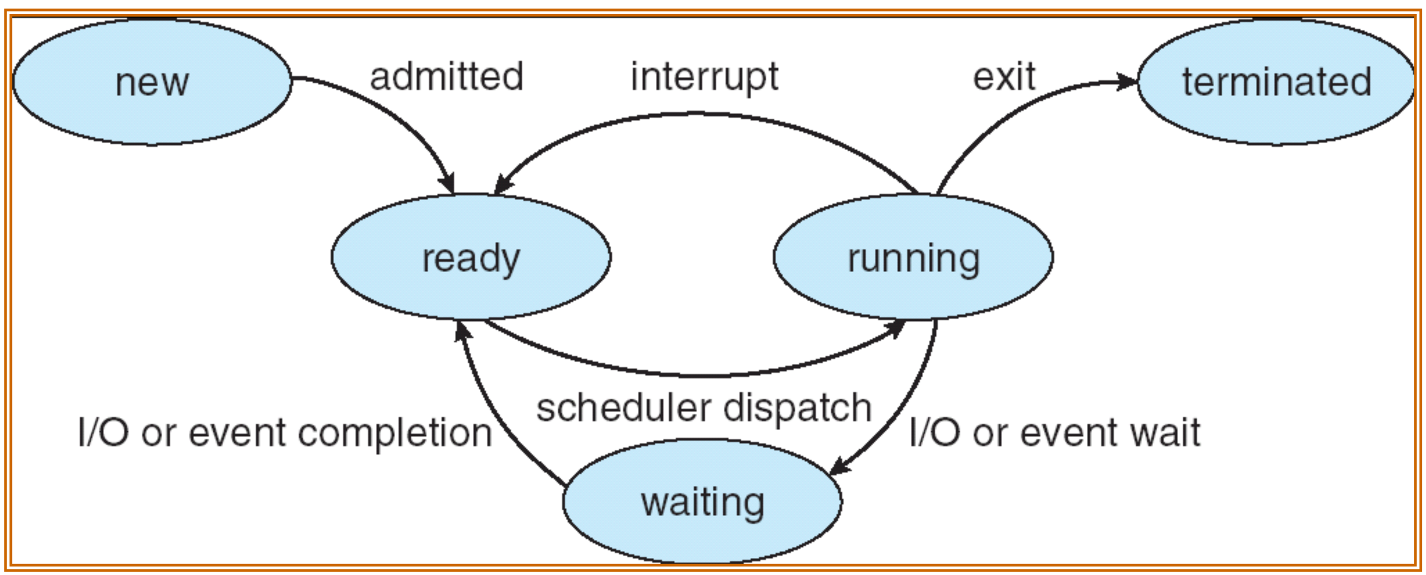
Operating Systems Fundamentals

Introduction to Processes



Introduction

- A **Process** is an **application / program in execution.**

In other words, it is the execution of a sequence of instructions (program), whose net result is the provision of some system function or user function.

We say that:

A processor runs a process,

or

A process runs on a processor.

Processes

- Processes are created whenever you start to run an application, i.e. when the operating System loads a program (e.g. a browser)

  A program becomes a process when it is loaded into memory.

- In the next few lectures we will look at the operations and aspects of the Operating System that support Processes and user applications

Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

Process Concept

- An OS executes a variety of programs & services
  - Batch system Jobs
    - e.g. cron jobs at night time, processing of payrolls, electricity bills not concerned with interacting with users , jobs are Q'ed up and run
  - Time-shared systems – User Programs, Tasks
    - Many exe's/windows running in parallel at same time
    - How does an OS do that??

- We know that a process is an application / program in execution
  - Process execution must progress in **sequential fashion**
    - Recall Assembly language and the Program Counter

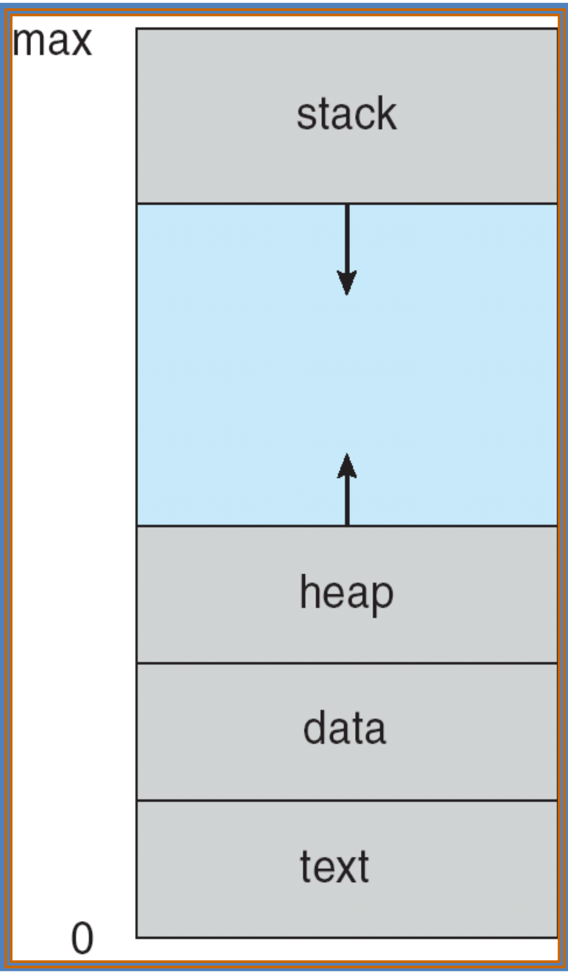
Process Concept, continued

- A Process must be self contained
  - All information a process needs to operate correctly … needs to be under the process's control

- To achieve this, a process includes:
  - Program Counter Specifies the next instruction to execute
  - Stack Used to store temporary data
  - Text/Data Sections This is the program code and global data

Recall your computer architecture from the current semester

A Process in Memory is more than the program code



The **process stack** contains temporary data, for example function parameters, return addresses and local variables.

The **heap** is memory that is dynamically allocated, i.e. allocated during run time

The **data section** contains global variables.

The **text** is the program code

https://www.sciencedirect.com/topics/engineering/process-stack

Question?

If a user invokes multiple copies of the same web browser,

do you think this is the same process?

Question?

If a user invokes multiple copies of the same web browser,

do you think this is the same process?

ANSWER

No - each is a separate process.

The text sections are the same

but the data, heap and stack sections are different for each process

Process States

- A Multi-tasking Operating System
  supports many processes

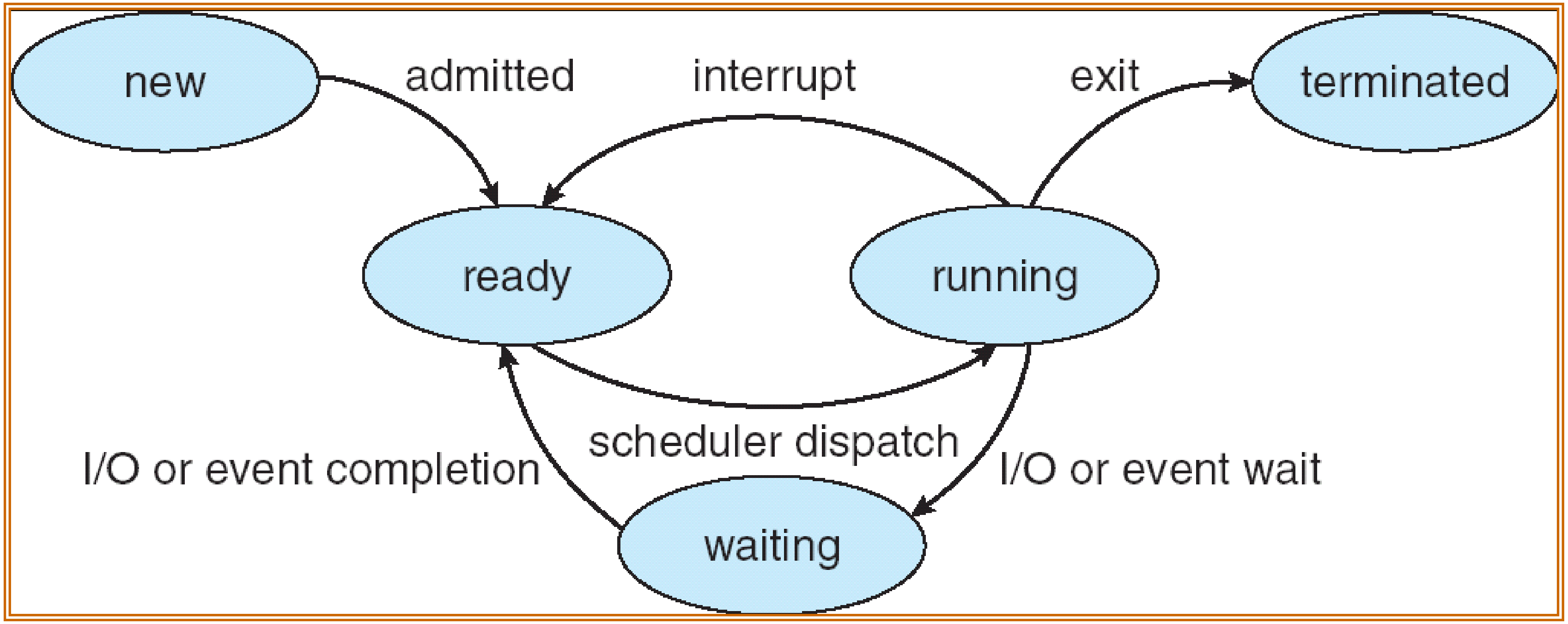
- Each process must be
  manageable and controllable

- To achieve this,
  processes have different **states**
  that control how they operate and behave

Process State

- As a process executes, it changes *state*
  - **New:** The Process is being created / started.
  - **Running**: Instructions are being executed / run on the CPU.
  - **Waiting**: The Process is waiting for some event to occur. e.g. Hard disk access, Keyboard operation
  - **Ready**: The Process is waiting to be assigned to a CPU / It is ready to be run.
  - **Terminated**: The Process has finished execution.

Process State Transitions



Process State Transitions, continued

- Progression between the states is controlled by the Operating System
  - An **Interrupt** can be generated by two things
    - An external application/hardware event or
    - the time slice assigned by the scheduler expires
  - **I/O or wait** event arises if the process must pause to gather extra data.
    - As this can take some time, CPU is freed to allow other processes to run
  - **Scheduler Dispatch** means the process can be run on the CPU based on the OS schedule scheme

    (more on OS schedules later…)

Process Data

- To allow an Operating System support many processes at once, each process needs specific data

- This data allows the OS to easily find out about each process without having to remember every process that is running

- The data is stored in a **Process Control Block**, also called **Process Descriptor**

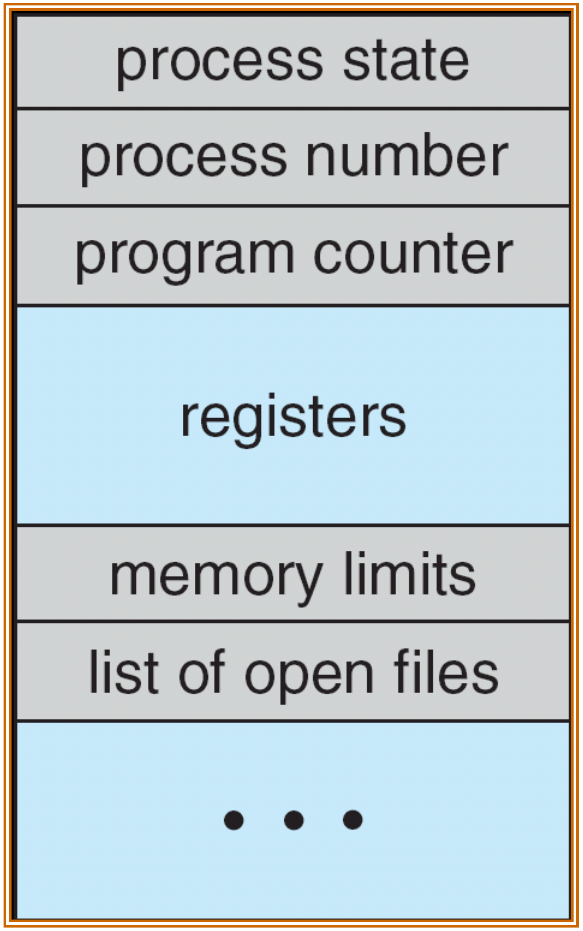
Process Control Block (PCB) Features

- The **Process Control Block** is an area in memory that stores the following process information that **uniquely characterises a particular process**.
  - **Process state**
  - **Program counter**
  - **CPU register contents**
  - **CPU scheduling information**
  - **Memory-management information**
  - **Accounting information**
  - **I/O status information**

Process Control Block (PCB) Features

- Process state
  - *Ready, running, waiting, etc*
- Program counter
  - *Next instruction to be executed for that process*
- CPU register contents
  - *Values*
- CPU scheduling information
  - *Used by short-term scheduler/ CPU scheduler. Includes a process priority etc*
- Memory-management information
  - *Memory limits.*
- Accounting information
  - *e.g. How much CPU used, when it was last run, etc*
- I/O status information
  - *e.g. List of I/O devices allocated to process, list of open files…*

PCB, continued



Context Switch

A <u>context switch</u> is the switching of the CPU from one process (or thread) to another.

Also known as **CPU Switch**

Context Switch

The OS kernel:

- stops *execution of the process* currently using the CPU.
- resumes *execution of some other process* that had previously been queued up waiting to use the CPU.

Process information stored in PCBs

Context Switch

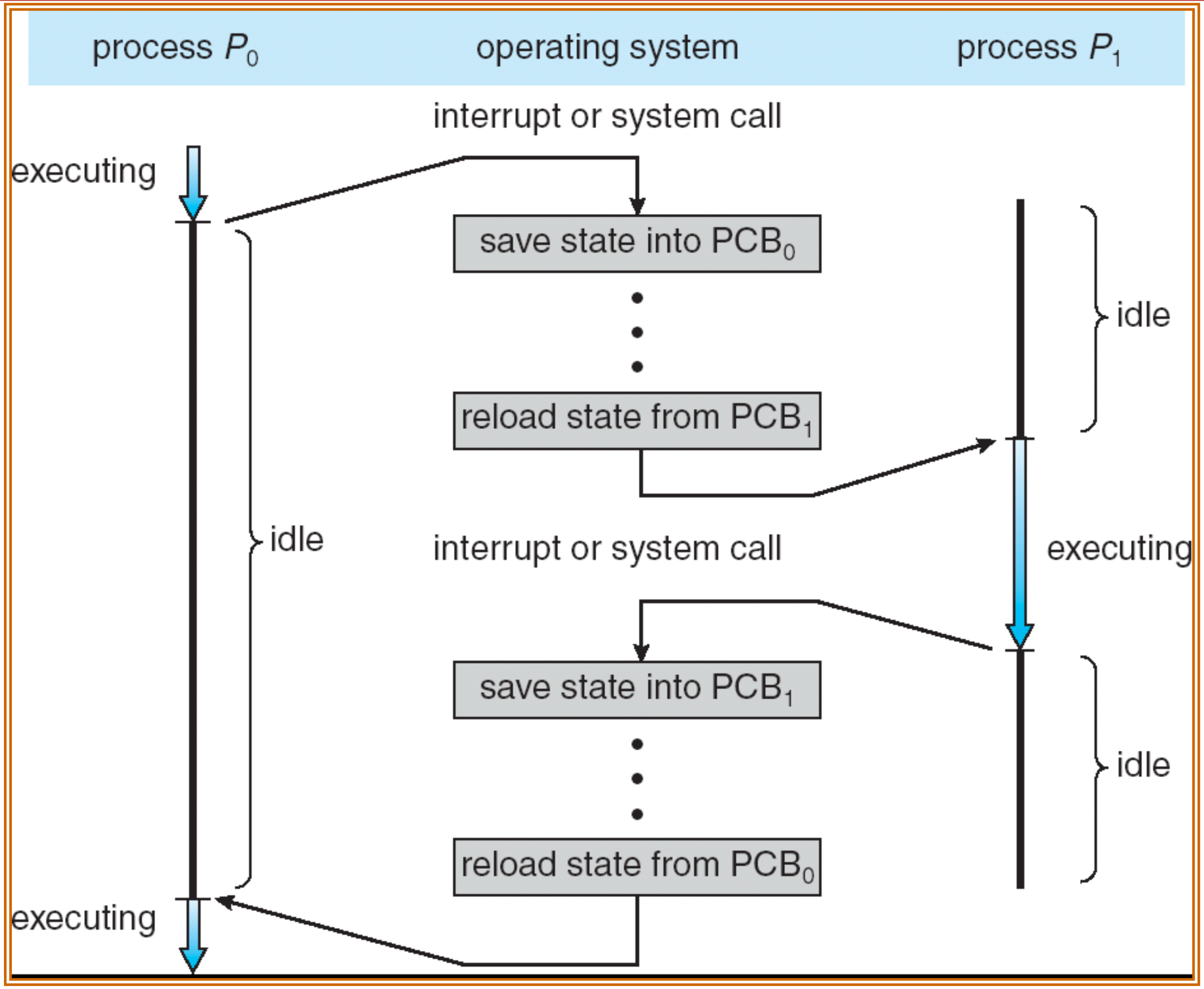
- It is not possible to read all registers and write all registers without some time overhead
  – The registers can be read quickly but only 1 at a time
  – There will be many registers to access and check
  – So …. This loses us time … but ….

Context Switch, continued

- The advantages of Multi-tasking outweigh the overheads of a context switch
  - Processors are getting faster while context switch operations are remaining basically the same

- The Process Control Block (PCB) supports the context switch by storing all its own process information
  - This means the OS does not need to store this information; it only needs to read and access the PCB information for a new process

Context Switch, continued
P_0 running, then P_1, then P_0



Process Scheduling

- The objective of multiprogramming,
  is to have some process running at all times.
  in order to maximise CPU utilisation

- The objective of timesharing is to switch the CPU
  among processes so frequently that users
  can interact with each program while running,
  without seeing any program stoppage or delay

- To meet these 2 objectives,
  the process scheduler selects
  an available process for program execution on the CPU.

Process Queues

- As we know, at any one time,
  many different processes exist.

- For example, as we have seen, processes may be the following states
  - Running using the CPU
  - Waiting for I/O (input/output)
  - Ready to be run

- Depending on its state, a process is put into a particular type of **process queue**.

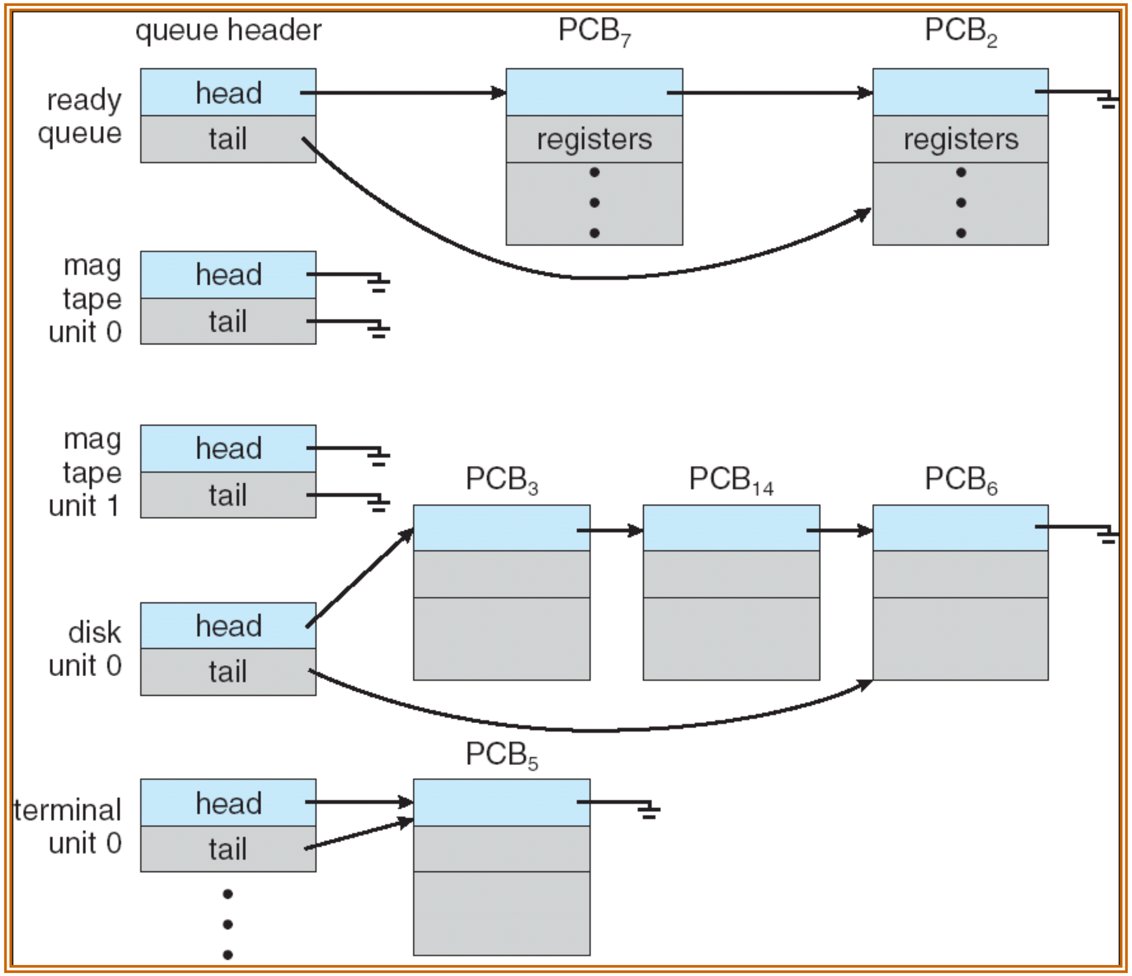
Process Queues

- **Job queue:** Set of all processes in the system
  - e.g. Look at Task Manager,
    all processes running on the computer are shown
- **Ready queue:** Set of all processes residing in main memory
  and ready to be executed
- **Device queue:** Set of processes waiting for an I/O device
  - There are many device queues
    as each device will have its own queue
- As a process executes,
  it migrates among the various queues

Process Queues

- As a process enters the system they are put into a **job** queue.
- The queues are generally stored as **linked lists.**
- Example of process migration:
  - A new process is put in the **READY** QUEUE.
  - Once the process is allocated the CPU and is executing, it could issue an I/O request and then be put into the **DEVICE(I/O)** QUEUE.

Process Queues, continued



Example:

Linked list design

Process Queues, continued

- The number of processes in each queue is important to the Operating System.
- It needs to know how much resources to apply to each
  - There is no point in giving time to operating the Device Queues if nothing is waiting for that device
- The choice of what to do with a Queue is handled by a **Scheduler**
  - We will be looking at schedulers later

Summary

- The concept of a Process has been introduced
- The data a Process must hold and store in PCB has been shown
- Process States and the operations at each have been presented
- Notion of how processes move between various queues also detailed