Concurrency

Week 2

Processes

Contents

- Processes
  - Definition
  - Creation
  - Termination
- Scheduling and Context Switching
  - Context Switching
  - Long-Term and Short-Term Scheduling
- Inter-process communication

Learning Outcomes of this Lesson

- At the end of this lesson, you will be able to:
  - Describe the concept of a process, its creation, its changing state, and its termination.
  - Understand multitasking and context-switching
  - Understand process types in terms of inter-process communications

Part One: Processes

- In this part, you will be learning the concept of a process and its life cycle. Try to find answers to the following questions:

1. What is a process?
2. What does a process need to execute (resources needed)
3. How is a process created?
4. What is a process state, and how does a process state change?
5. How does a process terminate?

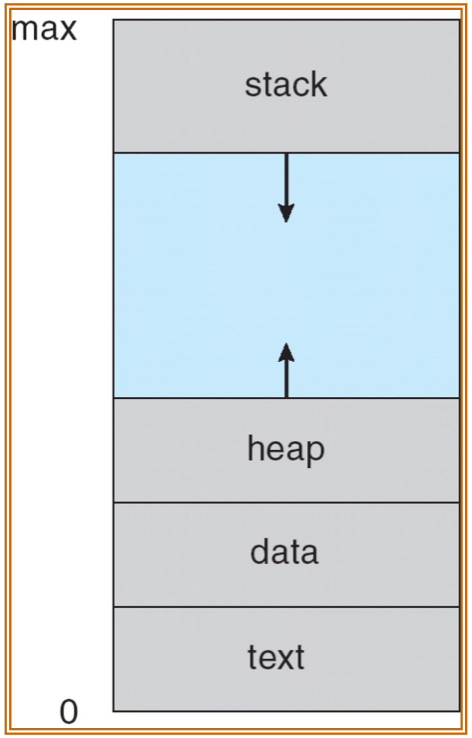
Process Concept: What is a Process?

- An operating system executes a variety of programs:
  - Text editors
  - Data processing
  - Web applications
  - Interactive games

- Process – a program in execution; process execution must progress in a sequential fashion

How a Program is Put in Execution?

- To put a program into execution we should:
  - Copy it (at least partially!) into the main memory
  - Set the program counter (pc) to point to the first instruction
  - Initialize stack
  - Reset flags
  - The process is ready, start running …



What is the difference between a program and a process?

- Process – a program in execution; process execution must progress sequentially (as from the slides before…). It is an active entity born from a program.
- A program can have multiple processes (initiates/owns)
- Multiple processes can be part of a program
- A process performs a task

What Resources are Used by a Process?

- A process has its own:
  - Code, data, and stack
    - Usually (but not always, when not?) has its own address space
  - Program State
    - CPU registers
    - The program counter (current location in the code)
    - Stack pointer
- Only one process can be running in the CPU at any given time!

Possible extra resource: https://www.youtube.com/watch?v=b4fsyrWegGo

When is a Process Created?

- Processes can be created in two ways
  - System initialization: some processes are created when the OS starts up
  - By executing a process creation system call: another process explicitly asks for a new process
- System calls can come from:
  - ***A user "request"*** to create a new process (system call executed from a user shell, for instance when a user clicks on an icon on the desktop)
  - ***already running processes***
    - User programs
    - System daemons*

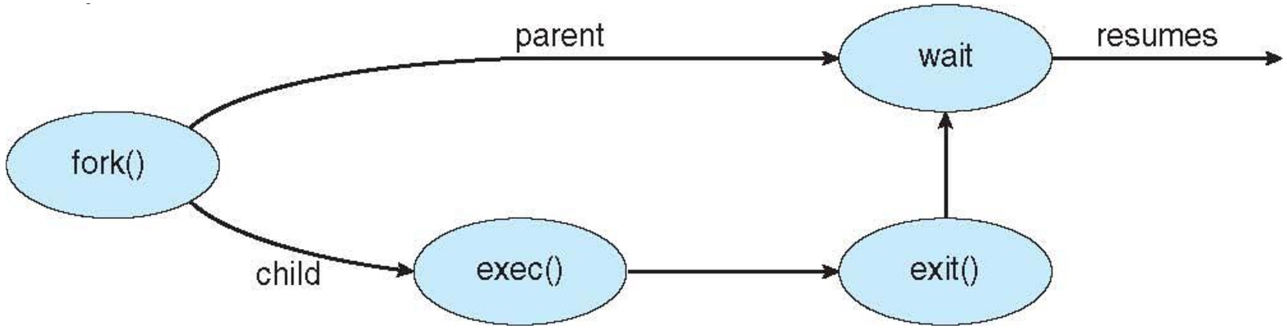
*System daemons: process that runs in background

Process Creation Options

- Resource-sharing options
  - Parent and children share all resources
  - Children share a subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation Options (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **`fork()`** system call creates new process
  - **`exec()`** system call used after a **`fork()`** to replace the process' memory space



When do Processes End?

- Conditions that terminate processes can be
  - Voluntary
  - Involuntary

  - Voluntary
    - Normal exit: when the process has no more instruction to execute.
    - Error exit
  - Involuntary
    - Fatal error
    - Killed by another process

Process Termination

- The process executes the last statement and then asks the operating system to delete it using the exit() system call.
  - Returns status data from the child to the parent (via wait())
  - Process resources are deallocated by the operating system
- The parent may terminate the execution of the children's processes using the abort() system call.
  - Some reasons for doing so:
    - A child has exceeded the allocated resources
    - Task assigned to a child is no longer required
    - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates

Process Termination

- Some operating systems do not allow a child to exist if its parent has terminated. If a process *terminates*, then all its children must also be terminated.
  - cascading termination**.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for the termination of a child process by using the **`wait()`** system call **.**
  - The call returns status information and the PID of the terminated process

```
PID = wait(&status);
```

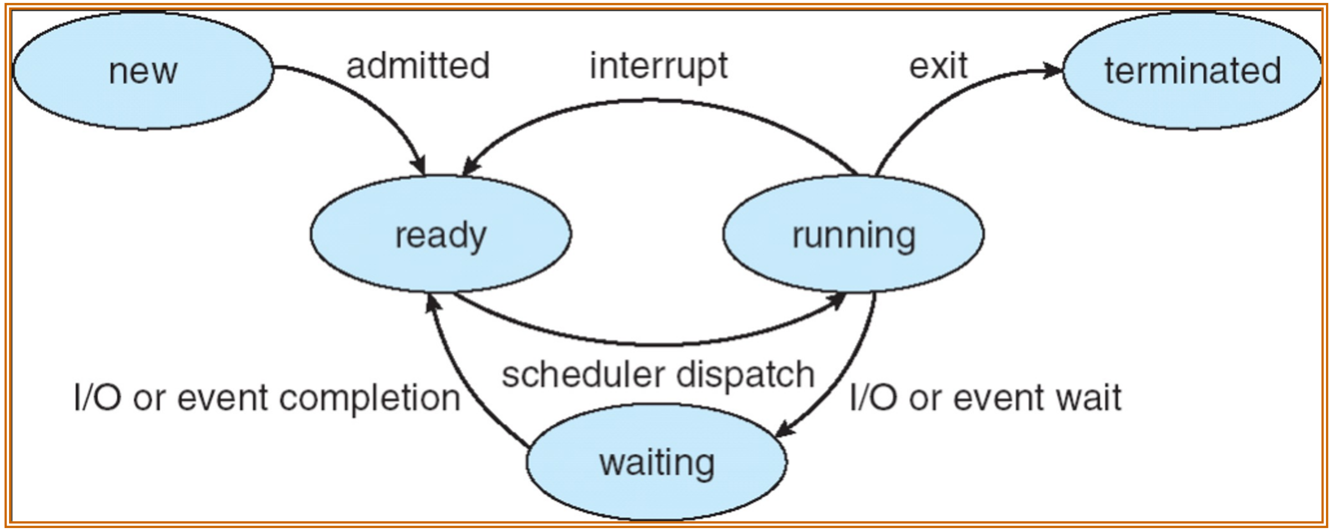
- If no parent is waiting (did not invoke **`wait()`**, *yet*), process is a **zombie (*NIX)**
- If **parent** is **terminated** without invoking **`wait()`**, process is an **orphan (*NIX)**

Process hierarchies

- Parent creates a child process
  - Child processes can create their own children
- Forms a hierarchy
  - UNIX calls this a “process group”
  - If a process exits, its children are “inherited” by the exiting process’s parent
- Windows has no concept of process hierarchy (in its base definition)
  - All processes are created equal
  - we can specify ***now*** how the inheritance kind with code (Win10)

Process State

- As a process executes, it changes its ***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 CPU
  - **terminated**: The process has finished execution



Process State Transition

- Possible transitions between states are

  1 - Process enters **ready** queue

  2 - Scheduler picks this process (change to **running** state)

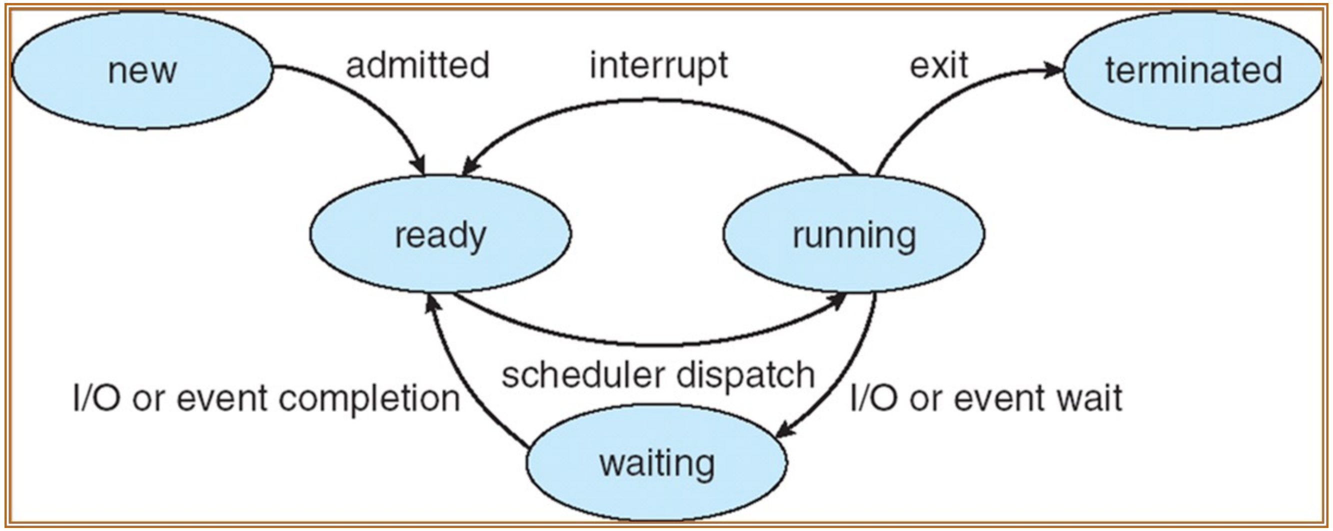
  3 - Scheduler picks a different process (go back to **ready** state)

  4 - Process **wait**s for event (such as I/O)

  5 - Event occurs (go to **waiting** state)

  6 - Process exits (**terminated**)

  7 - Process ended by another process (**terminated**)



Part One: Processes

- Now you can give answers to the following questions:

1. What is a process?
2. What does a process need to execute (resources needed)
3. How is a process created?
4. What is a process state, and how does a process state change?
5. How does a process terminate?

Part Two!

Part Two: Scheduling and Context Switching

- In this part, you will learn how processes share CPU time. Pay attention to find the answers to the following questions:

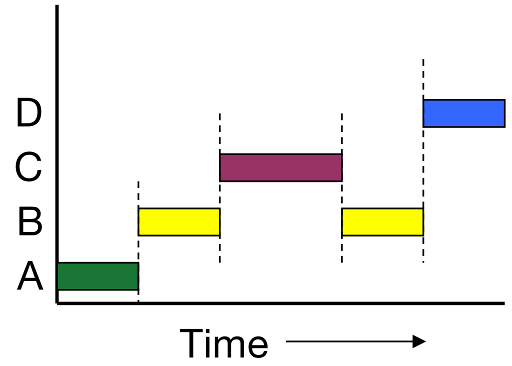
1. What is meant by CPU time sharing?
2. Why do we need time-sharing?
3. What is context-switching?
4. What is the context of a process and how is it stored?
5. What is process scheduling? What is a simple process scheduling model?

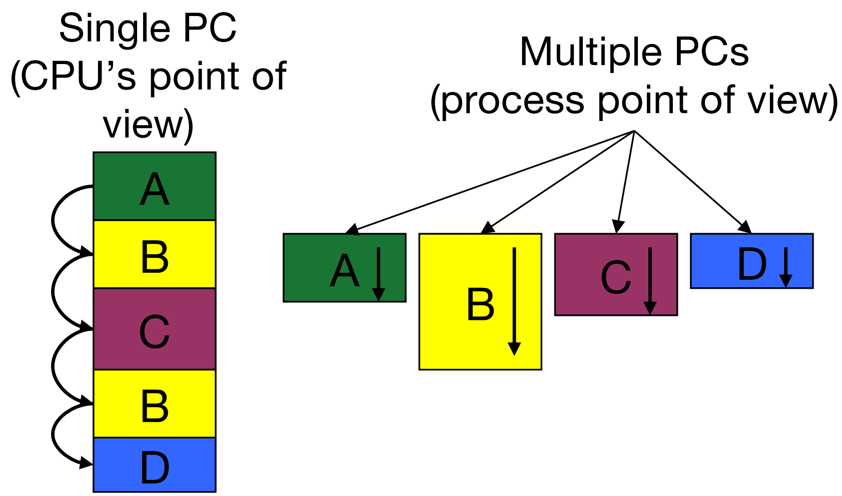
Switching from Process to Process

- Running multiple processes at the same time requires sharing CPU time (Assuming a single CPU/core)
- Stopping a process and allocating CPU to another process is called context switching
- Context switching is used for:
  - Maximizing CPU utilization by running a ready process when the current process is blocked (waiting for I/O for instance)
  - Creating the illusion of a multi-processor system on single-processor systems by running each process for a short period named a quantum.
- Context-switch time is *overhead*; the system does no useful work while switching context

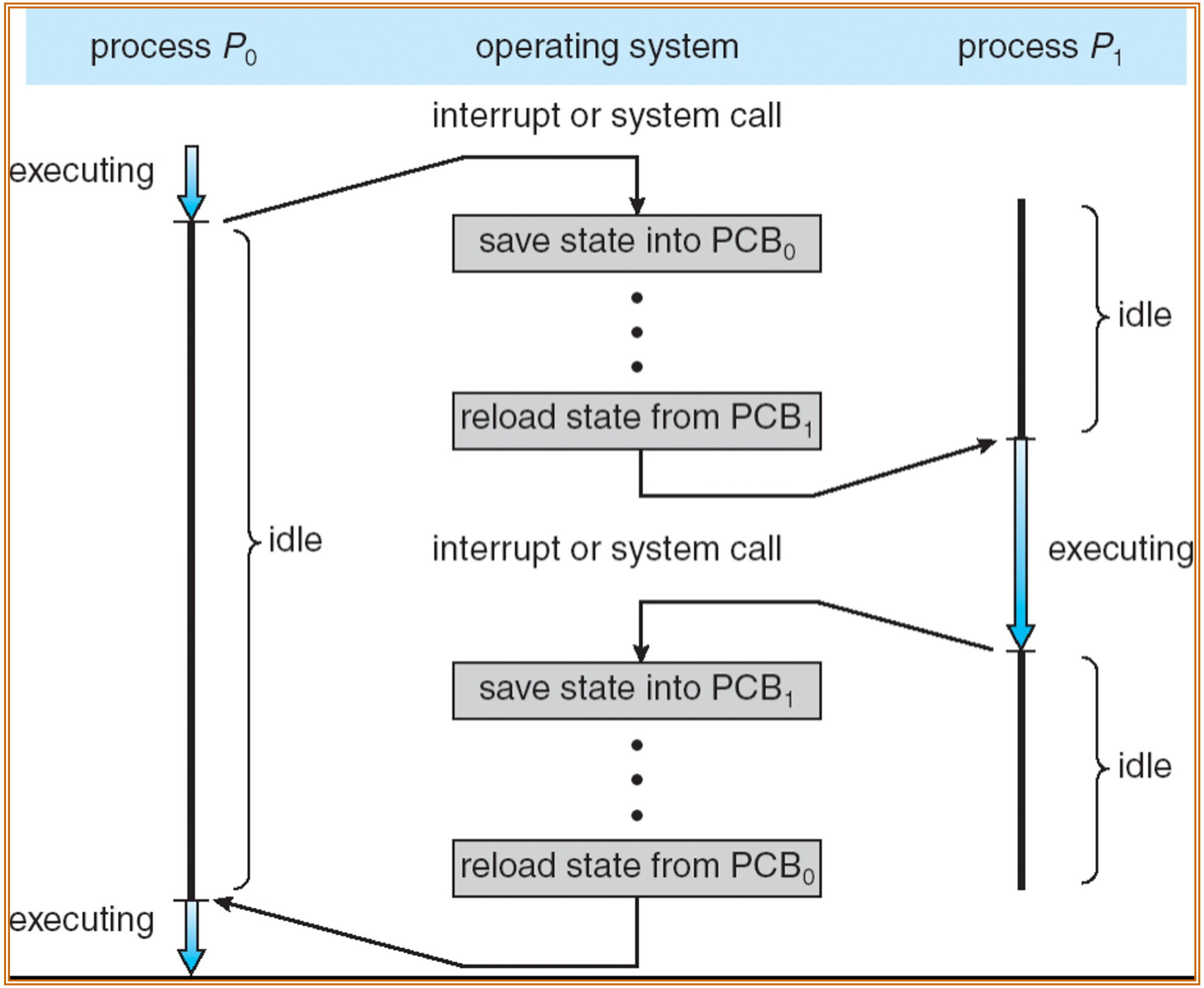
The Process Model: Multiprogramming

- Multiprogramming of 4 programs
- Conceptual model
  - 4 independent processes
  - Processes run sequentially
- Only one program is active at any instant!
  - That instant can be very short…





Context Switching



PCB: process control block (see op4 year 1)

What is a Process Control Block?

All information about a process is stored in a data structure named Process Control Block (PCB)

May be stored on stack

Process management
Registers
Program counter
CPU status word
Stack pointer
Process state
Priority / scheduling parameters
Process ID
Parent process ID
Signals
Process start time
Total CPU usage

File management
Root directory
Working (current) directory
File descriptors
User ID
Group ID

Memory management
Pointers to text, data, stack
or
Pointer to page table

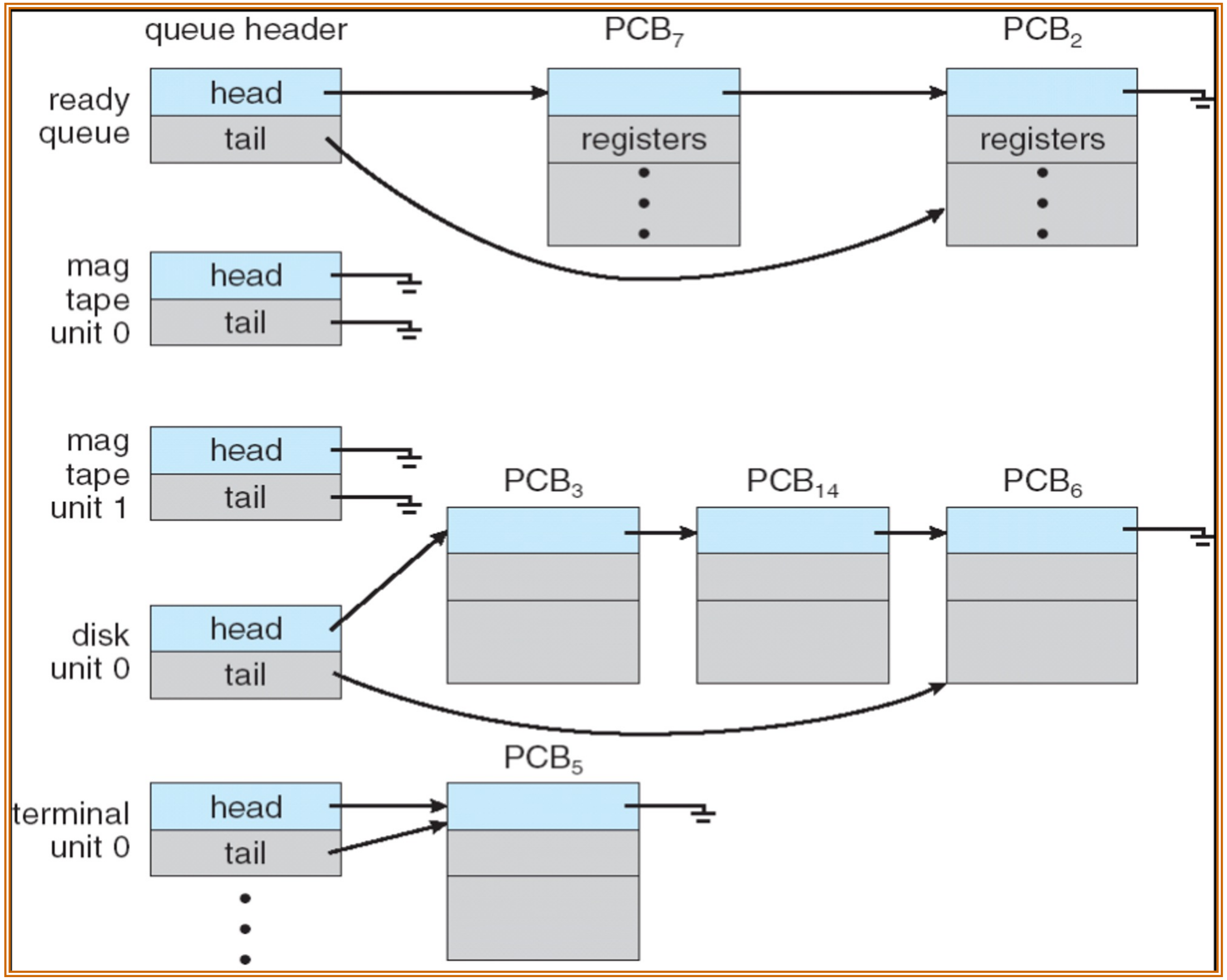
Process Scheduling

- Context switching requires choosing a process to allocate CPU when the CPU is withheld from the current process.
- Process scheduling refers to the methods used for selecting the next process to run (see Operating System course for the details)
- To apply process scheduling algorithms, multiple queues are created

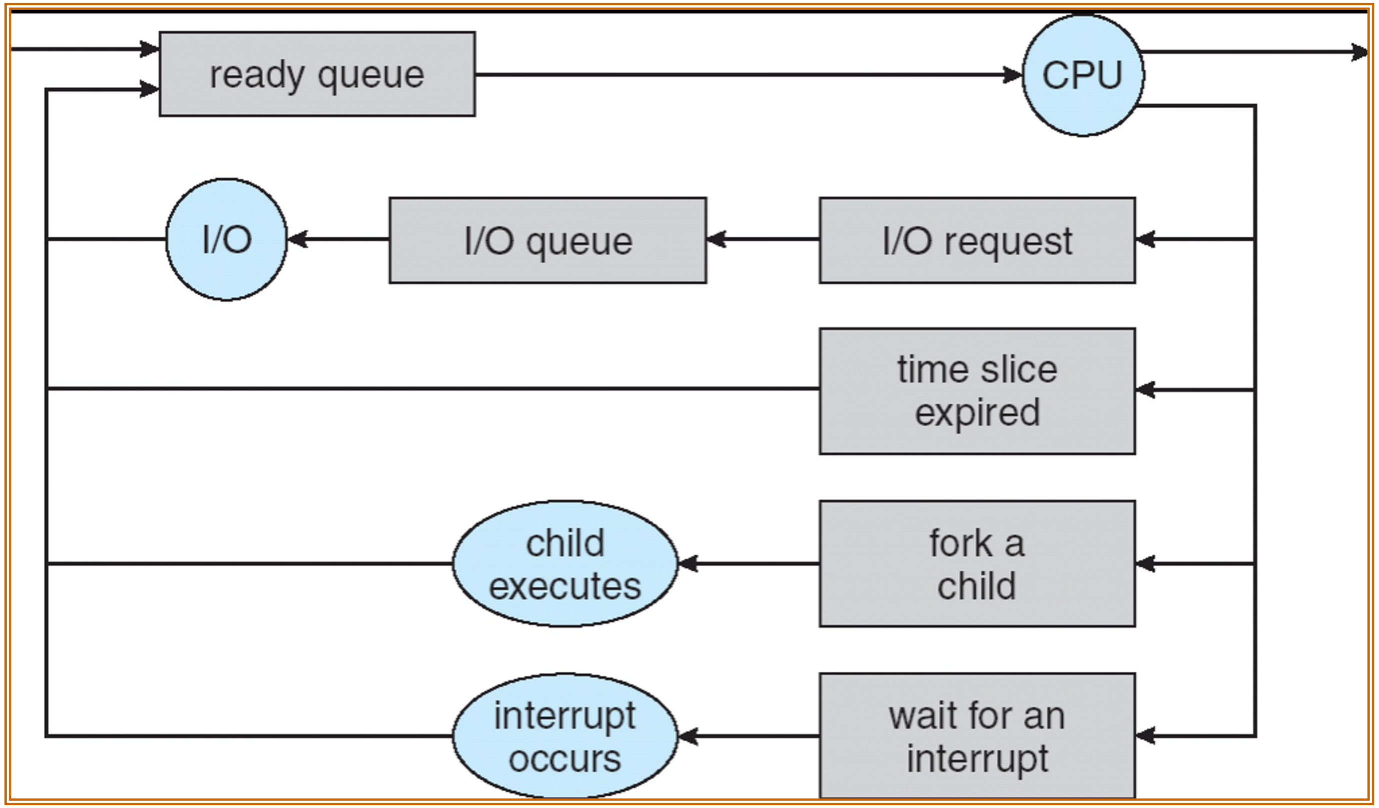
Process Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to be executed
- **Device queues** – a set of processes waiting for an I/O device
- Processes migrate among the various queues

Ready Queue and Various I/O Device Queues



Simplified Model of Process Scheduling



Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the *ready queue from the job queue*
  - The long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the *degree of multiprogramming*
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - The short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- 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

Part Two: Scheduling and Context Switching

- Now you can give answers for the following questions:

1. What is meant by CPU time sharing?
2. Why do we need time-sharing?
3. What is context-switching?
4. What is the context of a process and how is it stored?
5. What is process scheduling? What is a simple process scheduling model?

Part Three!

Part Three: Inter-process communications

- In this part, you will learn about communication between processes. Try to find answers to the following questions:

1. What is meant by communication between processes?
2. Does every process have to communicate with other processes?
3. What are the main methods for communication between processes?

Process Classes (kinds)

- **The independent** process cannot affect or be affected by the execution of another process
- **The cooperating** process can affect or be affected by the execution of another process

- Advantages/objectives of process cooperation
  - Information Sharing
  - Computational speed-up (given the right conditions)
  - Modularity (ease of maintenance and abstractions of tasks)
  - Convenience

Example: Cooperating Processes

- Producer-Consumer Problem
  - A process named producer generates data items. A second process named consumer utilizes the data
  - *producer* process shares the information with the *consumer* process using a shared buffer which can be either a(n):
    - *Unbounded buffer* places no practical limit on the size of the buffer
    - *bounded buffer* assumes that there is a fixed buffer size

Inter-Process Communication (IPC)

- The mechanism for processes to communicate and 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 a communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Facilitating Inter-Process Communication

- If the cooperating processes share information for computation speed-up or modularity, they can be part of the same process.
- Being part of the same process makes it possible to
  - Share the code
  - Share the data
  - Share the resources
- However, each one should have its own
  - Stack
  - Registers
  - Program counter
- Each sub-part of a process is named a thread (next week)

Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- The 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 **inter-process communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Part Three: Inter-process communications

- Now, you can give answers to the following questions:

1. What is meant by communication between processes?
2. Does every process have to communicate with other processes?
3. What are the main methods for communication between processes?

Summary

- A process is a program in execution
- A process is created at system initialization, or through a system call issued by another process
- Processes may terminate voluntarily or be killed by another process
- Processes may take different states during execution
- The operating system revokes the CPU from the running process and grants it to another process, a procedure called context switching
- Operating systems use different scheduling algorithms to decide which process should run next

Quiz Time

- Please, answer the quiz questions for week two.
- You have 10 minutes.
- We will discuss