Operating System

Lecture 3 – 2017.3.7

Chapter 3

Process Concept

Process Scheduling

Operations on Processes

Interprocess Communication

Moodle

Contact Moodle Admin

Mr. Amjed Altaweel
amjed.altaweel.88@gmail.com

If you got an old account send your related e-mail (user) to be activated.
If you wont have an account sent your active email address.

In both conditions send your complete Three Name / Dept. / Student Info
Ask for him at the secretary of Com Eng. Dept.

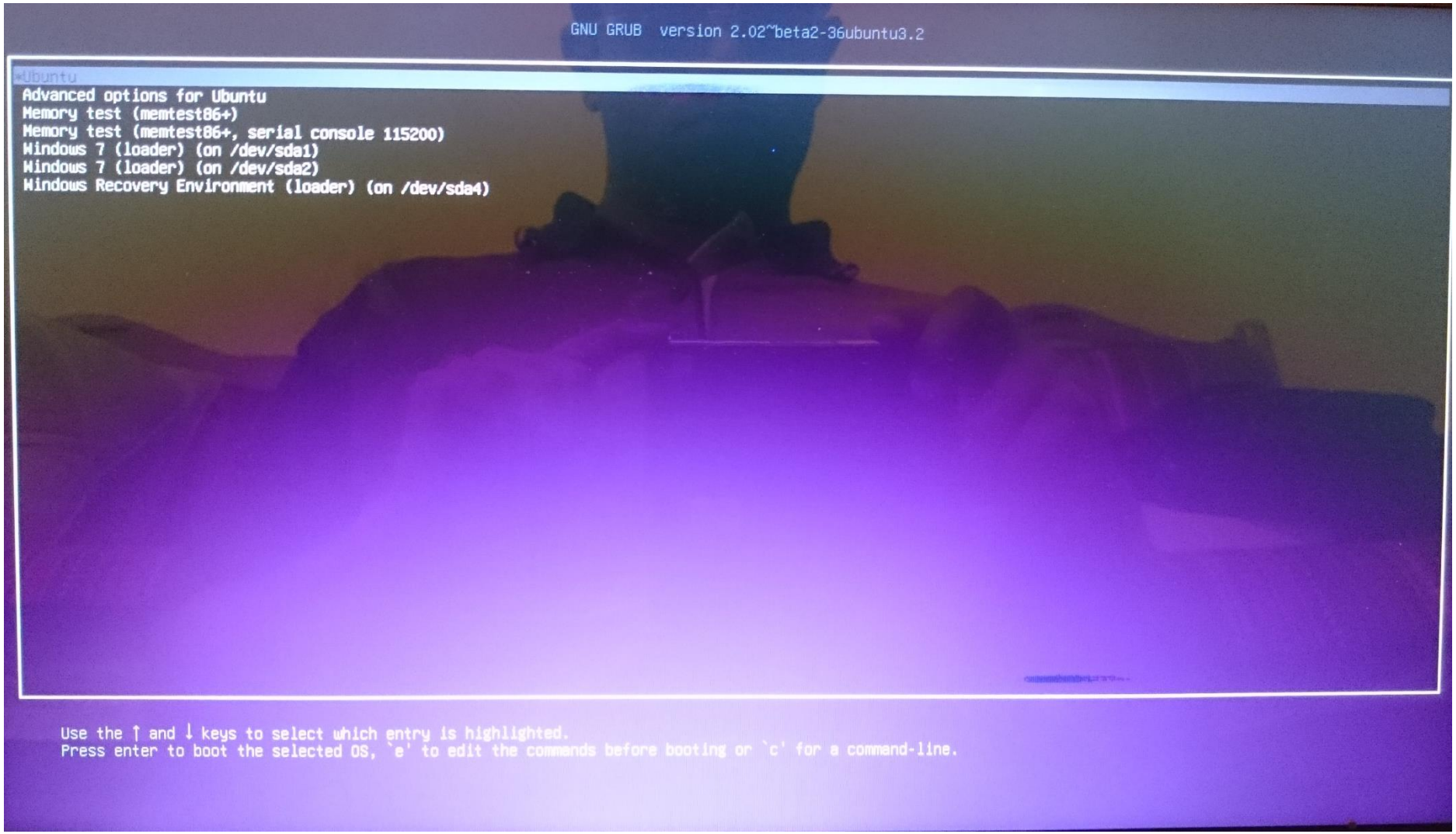
No excuses accepted for inactive accounts

OS Linux Review

- Linux History
- Distributions Debian / Ubuntu
- Installation (Single/Dual mode –Virtual Box – Portable)
- Command line (Terminal)
- $ chmod ug=rw,o-rw,a-x *.txt

U=user (owner) g=group o=other / man / r (recursively)

Objectives


GNU GRUB version 2.02~beta2-36ubuntu3.2
*Ubuntu
Advanced options for Ubuntu
Memory test (memtest86+)
Memory test (memtest86+, serial console 115200)
Windows 7 (loader) (on /dev/sda1)
Windows 7 (loader) (on /dev/sda2)
Windows Recovery Environment (loader) (on /dev/sda4)
Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, `e' to edit the commands before booting or `c' for a command-line.

Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To explore interprocess communication using shared memory and message passing

Process Concept

Process

Is a program in execution (is the unit of work in a modern time-sharing). A process will 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. The more complex the operating system is, the more it is expected to do on behalf of its users.

operating-system processes execute system code,

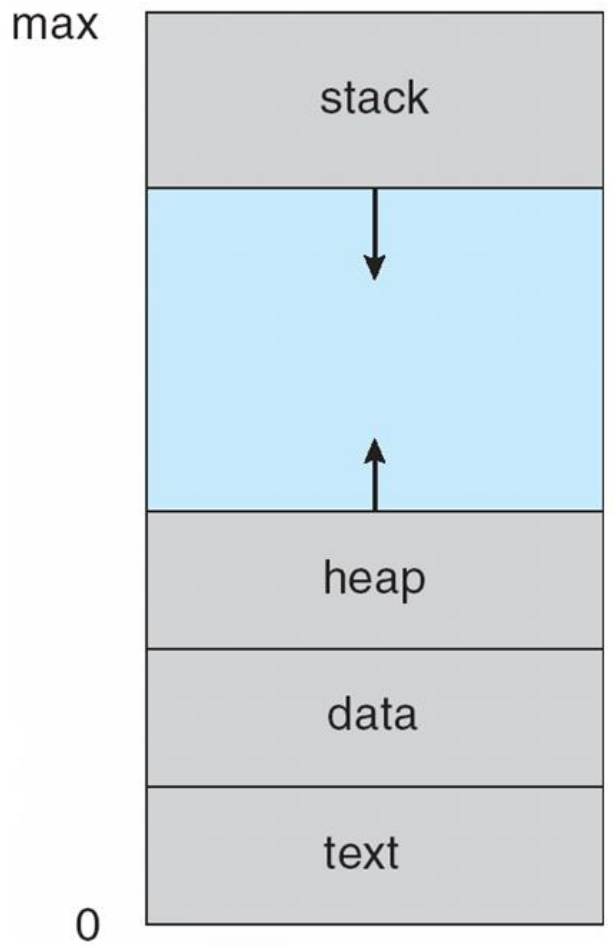
and

user processes execute user code.

All these processes may execute concurrently.

Process Concept

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



Structure of process in memory

Process Concept

- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  Program becomes process when executable file loaded into memory

- Execution of program started via GUI mouse clicks, command line entry of its name, etc

- One program can be several processes
  Consider multiple users executing the same program

Process State

As a process executes, it changes **state**

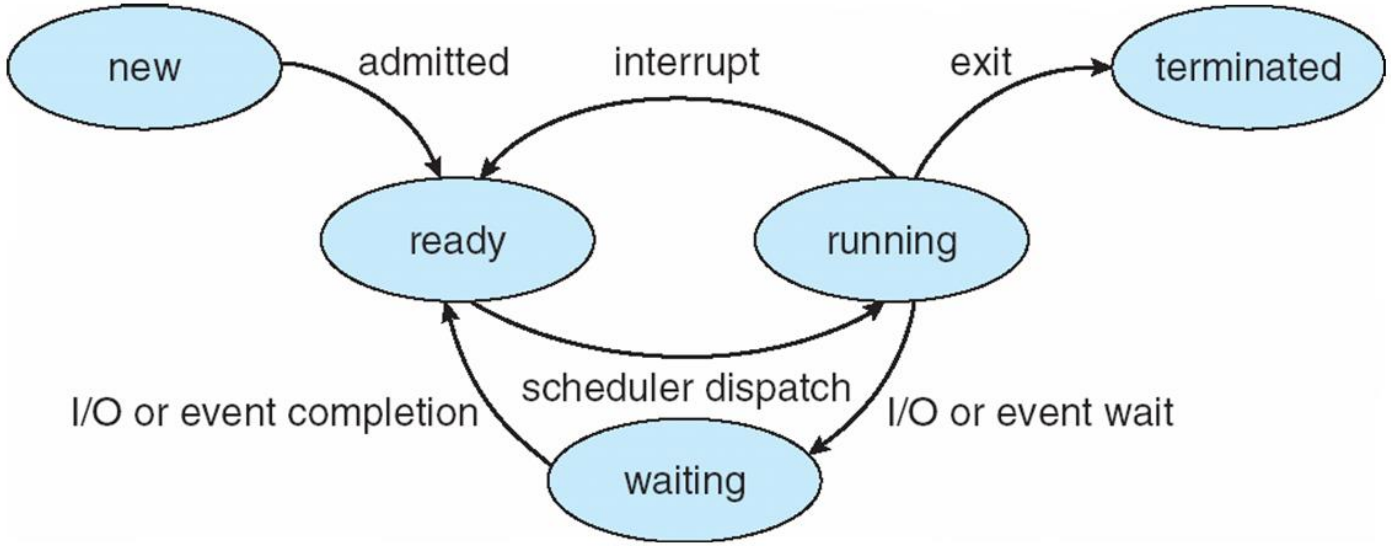
new: The process is being created

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

terminated: The process has finished execution



Process State Diagram

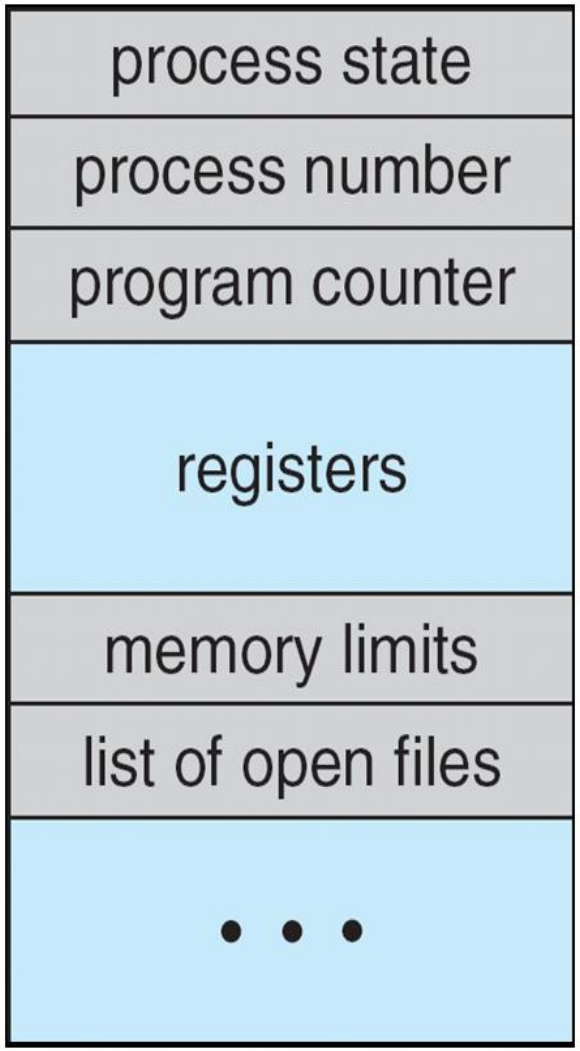
It is important to realize that only one process can be *running* on any processor at any instant. (old OS)

Process Control Block (PCB)

Information associated with each process

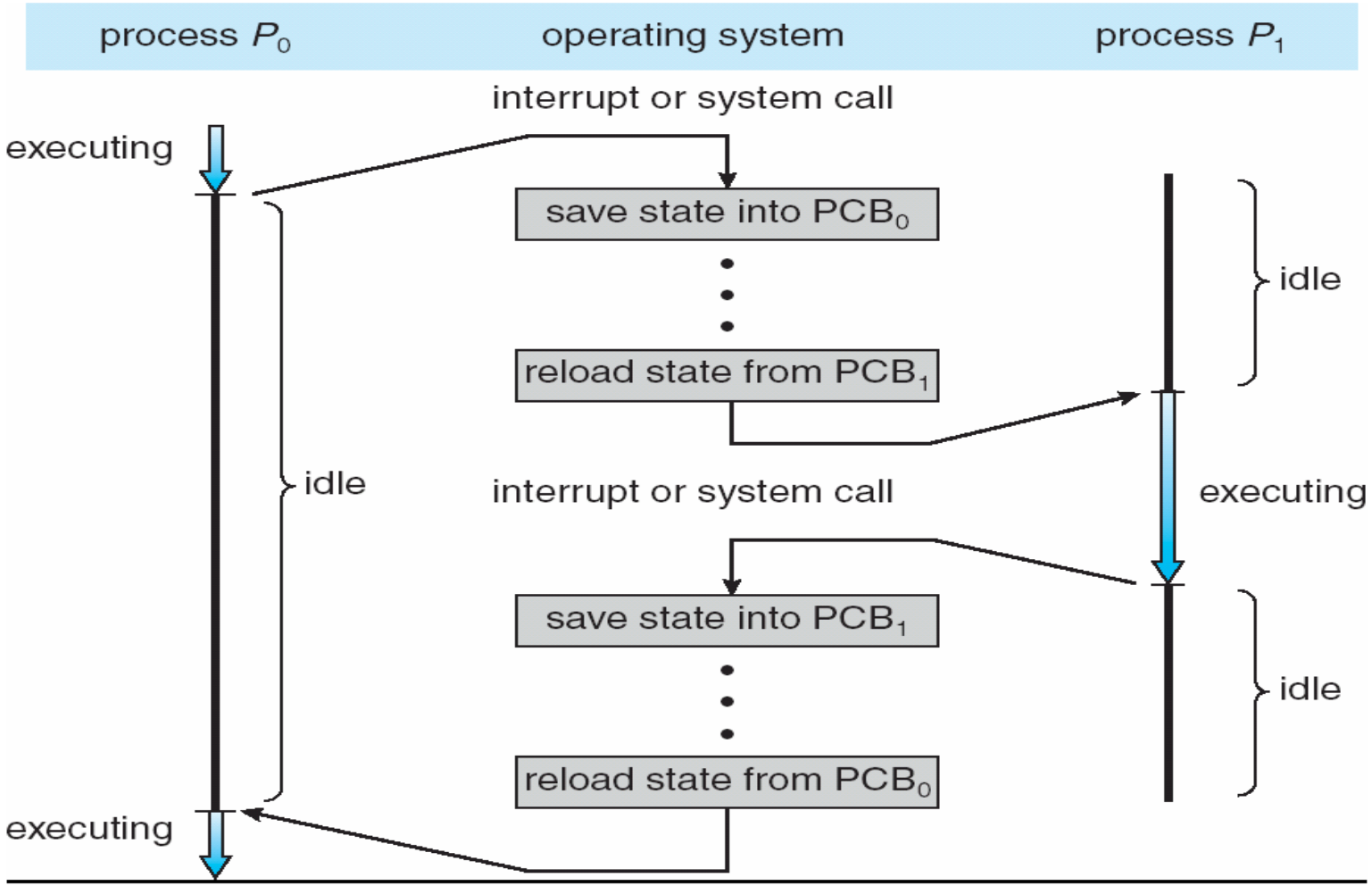
(also called **task control block**)

- **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



PCB Diagram

CPU Switch From Process to Process



Threads

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

A traditional (or ***heavyweight***) process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.

Threads

- So far, process has a single thread of execution

Most modern operating systems have extended the process concept to allow

- Consider having multiple program (multicore systems) 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
- Chapter 4

OS Kernel Assignment

A possible clue about assignment sources in source book

e.g:

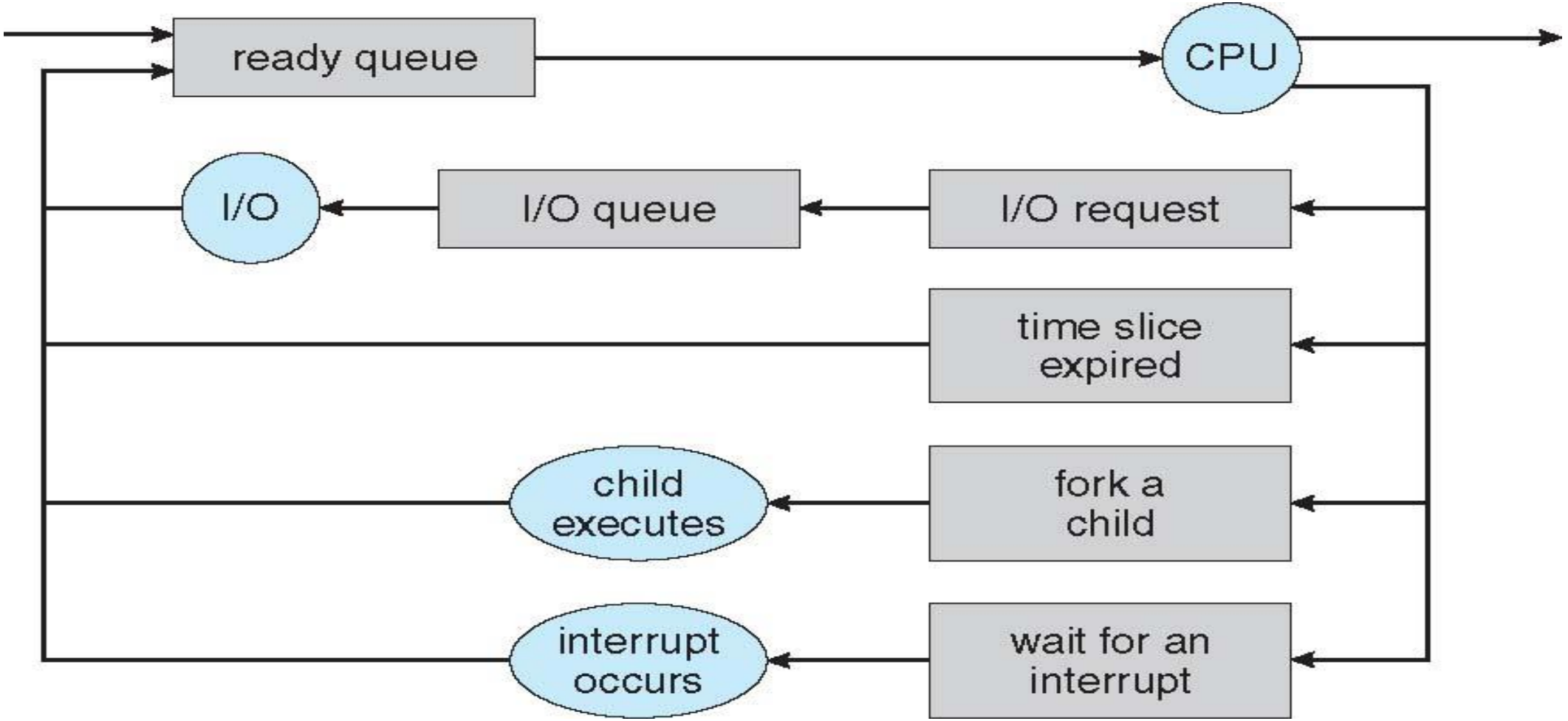
P.G 110 Abraham source book **PROCESS REPRESENTATION IN LINUX**

Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** 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
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Representation of Process Scheduling

Queueing diagram rectangular box represents queues (Ready – Device), resources (circles) , flows

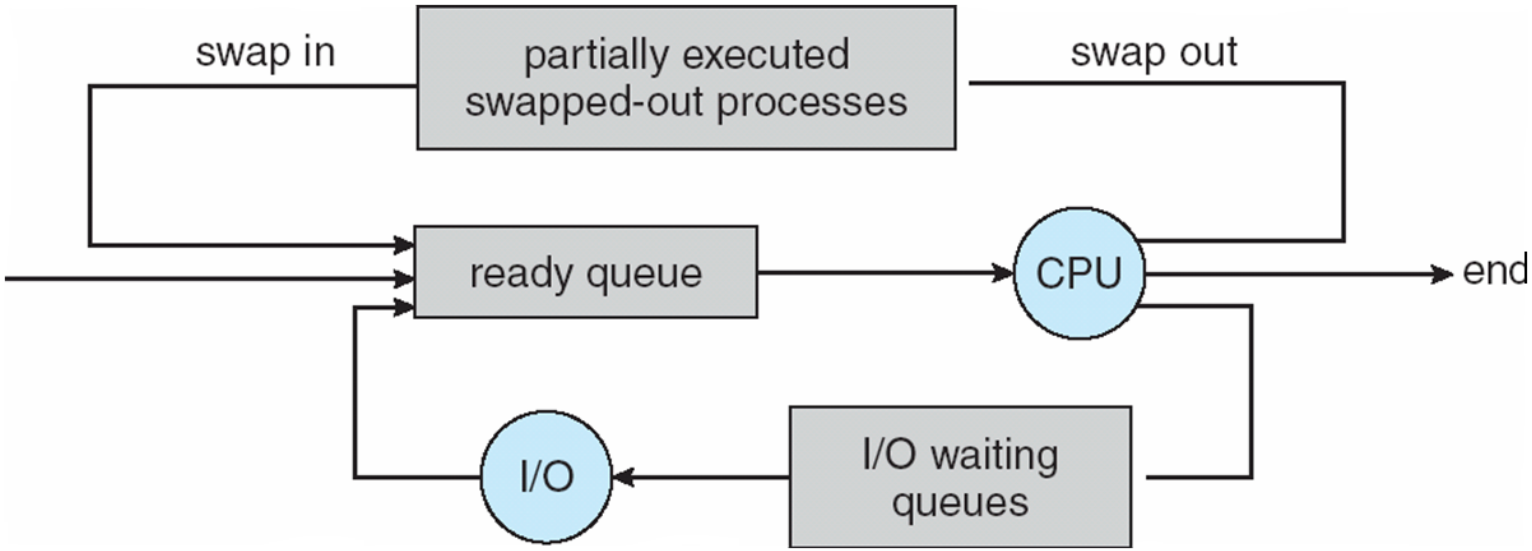


Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – 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**
- 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
- Long-term scheduler strives for good ***process mix***

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 (drive out – fill in)**



Context Switch

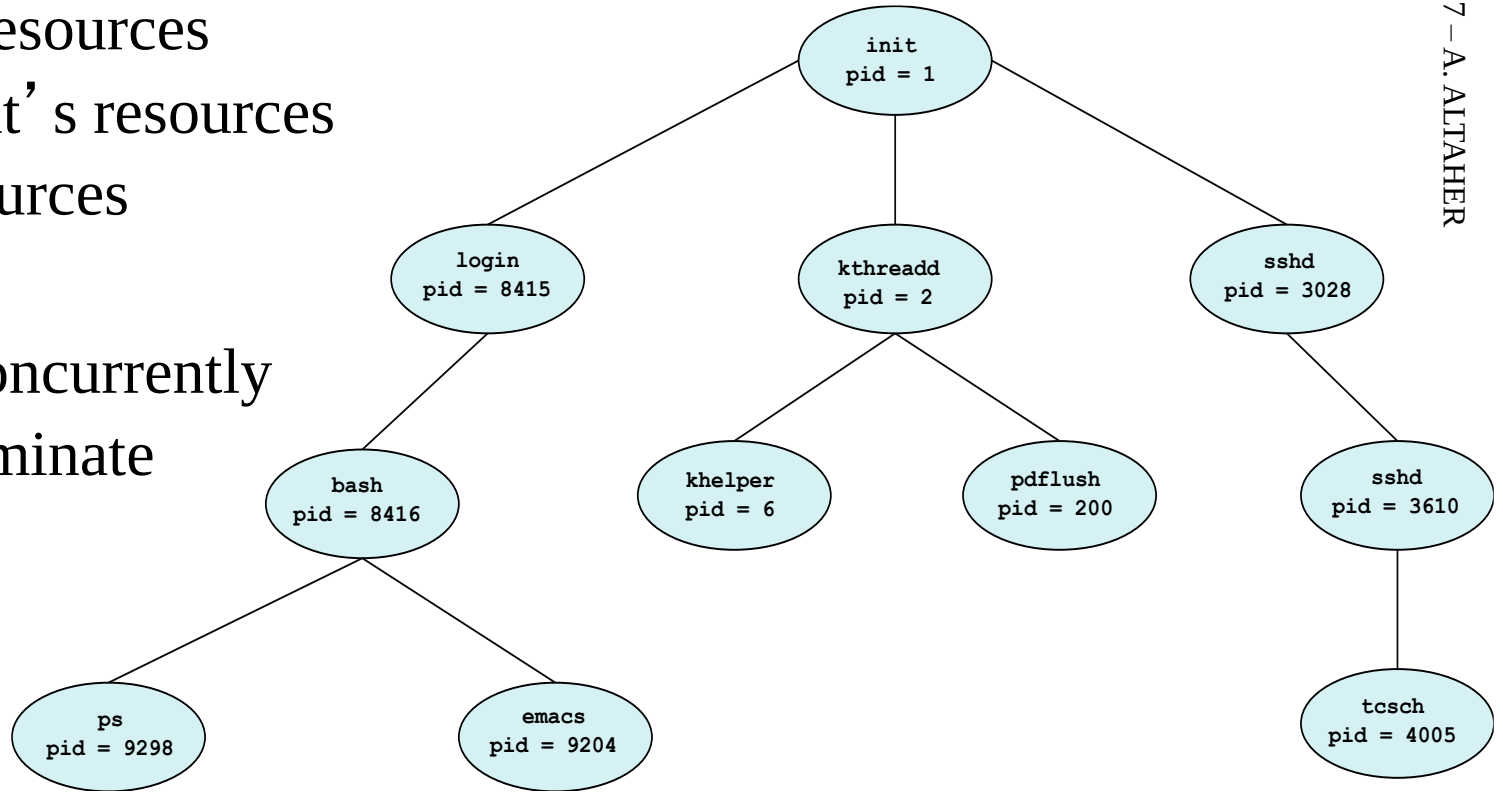
- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - ➢ The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - ➢ Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Operations on Processes

- System must provide mechanisms for:
  - process creation,
  - process 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



A Tree of Processes in Linux

Parent & child Process Facts

Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many child processes.

When a process creates a new process, two possibilities for execution exist:

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

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

There are also two address-space possibilities for the new process:

1. The child process is a duplicate of the parent process (it has the same program and data as the parent).

2. The child process has a new program loaded into it.

Process Termination

- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - Returns status data from child to parent (via **wait()**)
  - Process' resources are deallocated by operating system

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

Process Termination

- Some operating systems do not allow child to exists 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 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 waiting (did not invoke (استحضار) **wait()**) process is a **zombie**
- If parent terminated without invoking **wait** , process is an **orphan**

Multiprocess Architecture – Chrome Browser

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 different types of processes:

- **Browser** process manages user interface, disk and network I/O
- **Renderer** process renders web pages, deals with HTML, Javascript. A new renderer created for each website opened
  - Runs in **sandbox** restricting disk and network I/O, minimizing effect of security exploits
- **Plug-in** process for each type of plug-in

Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- ***Independent*** process cannot affect or be affected by the execution of another process
- ***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**

Communications Models

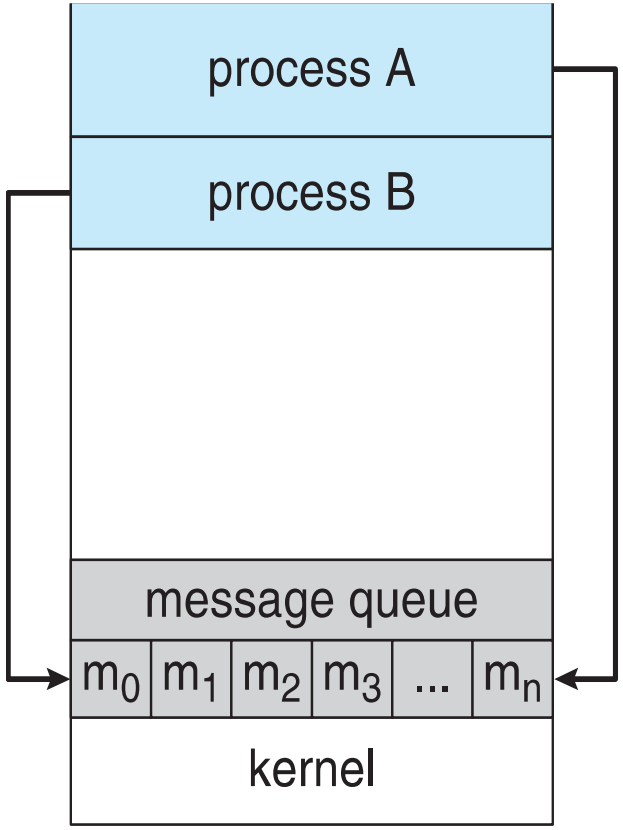
Message passing pros:

1- useful for exchanging smaller amounts of data, because no conflicts

2- easier to implement in a distributed system

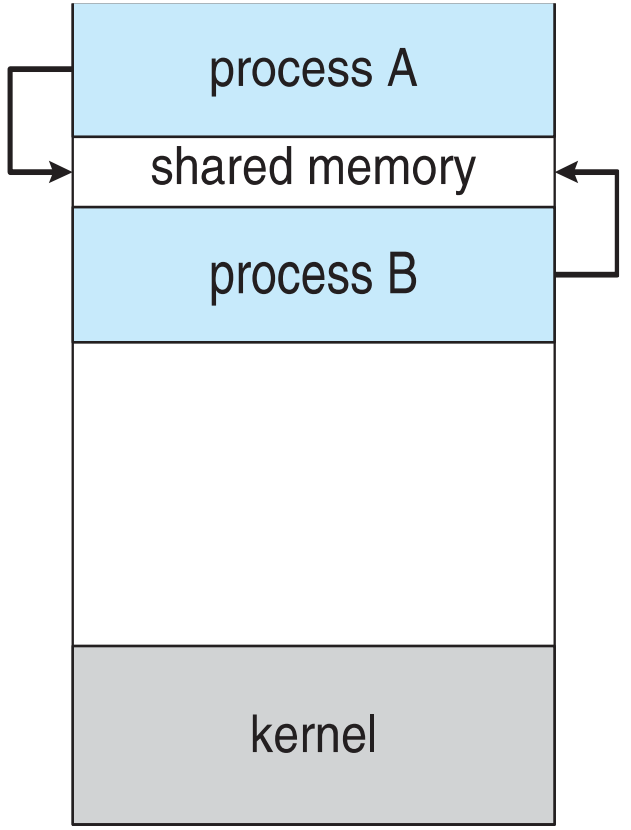
Alternatively –
there are systems that provide distributed shared memory

(a) Message passing.



(a)

(b) shared memory.



(b)

Shared memory
Pros:
1- Faster

2- no assistance from the kernel is required

Cons:

suffers from cache coherency issues because shared data migrate among the several caches

Chapter 4 Threads