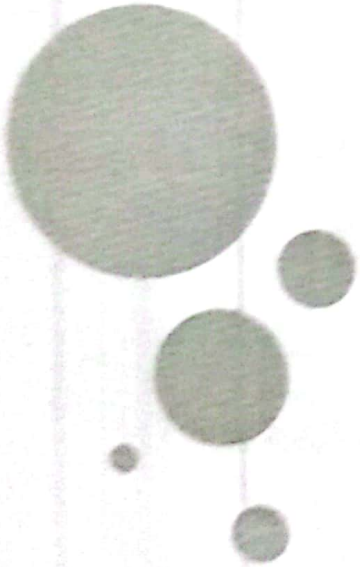
COURSE TITLE:

SYSTEM PROGRAMMING



By: Prof. Ebot Ebot Enaw

COURSE PLAN

- **The Course is titled System Programming and Structured into Seven Chapters namely:**
  - CHAPITRE I PROCESSES
  - CHAPITRE II SIGNALS
  - CHAPITRE III ORDINARY AND NAMED PIPES
  - CHAPITRE IV MESSAGE QUEUE
  - CHAPITRE V SHARED MEMORY
  - CHAPITRE VI SEMAPHORE
  - CHAPITRE VII SOCKET AND CLIENT SERVER PROGRAMMING
- The question that usually arises in Computer Systems is what to call all CPU activities. The answer lies in the following:
  - What is a process ;
  - What motivated the coining of the term process
  - What is the origin of a process

EARLY COMPUTER SYSTEMS

- Allow only one program to be executed at a time
- Program had complete control of the system and had access to all the system resources.

CURRENT-DAY COMPUTER SYSTEMS

- Allow multiple programs to be loaded into memory and executed concurrently
- This evolution required firmer control and more compartmentalization of the various programs

These needs resulted in the notion of :

Process: Program in execution

: Unit of work in a modern time sharing system,

OPERATING SYSTEM

Main concern:	• The execution of user programs • Also needs to take care of various system tasks that are better left outside the kernel itself.
Consists of :	A collection of processes: operating system, processes executing system code and user processes executing user code.
CPU multiplexed:	Potentially all these processes can execute concurrently with the CPU multiplexed amongst them.
Productivity improved:	By switching the CPU between processes, the Operating system can make the computer more productive

PROCESS CONCEPT

- Two main computer systems namely :
  1. **Batch System:** executes jobs
  2. **Time sharing system:** executes user programs or tasks

Single user time sharing system such as windows

- A user can run several programs at one time: a word processor, a web browser and an email package ;
- Even if user can execute only one program at a time. The OS may need to support its own internally programmed activities such as memory management.

All these activities are similar and are all called processes.
Job and process used interchangeably

THE PROCESS

- A program in execution
- More than the program code (text section)

It also includes:

• **The Current activity:**	As represented by the value of the program counter and the content of the processor's registers
• **Process stack contains:**	Temporary data (such as functions, parameters, return addresses and local variables)
• **Data sections contains:**	Global variables
• **Heap:**	Dynamically allocated memory during process run time

PROGRAM	PROCESS
• Passive entity such as a file containing list instructions stored on a disk (often called executable file)	• Active entity with a program counter specifying the next instruction to execute and a set of associated resources.

A program becomes a process when an executable file is loaded into memory by double clicking an icon representing the executable file and entering the name of the executable file in the command line.

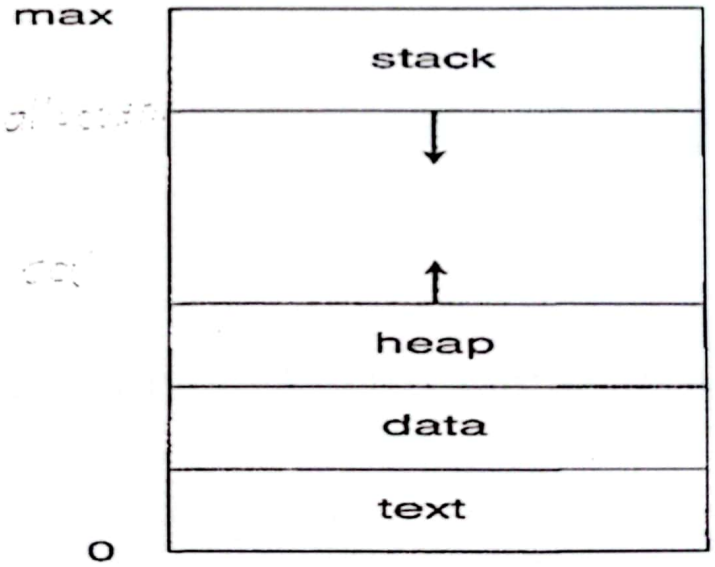
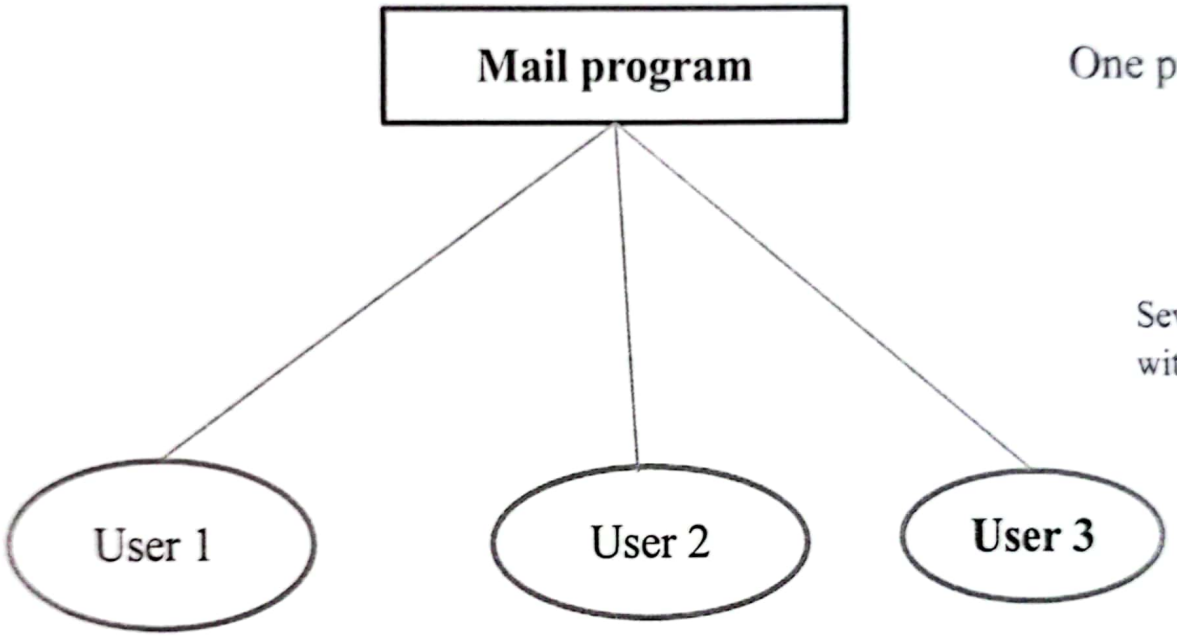


Figure 1: Process in memory

SEVERAL PROCESSES MAY BE ASSOCIATED WITH THE SAME PROGRAM

Two scenarios consider several execution sequences:

1. **Several users may be running different copies of mail program**



2. **Same user invokes many copies of a web browser**

PROCESS STATE

- Defined by the current activity of the process
- As a process executes, it changes states and can be in one of the following states.

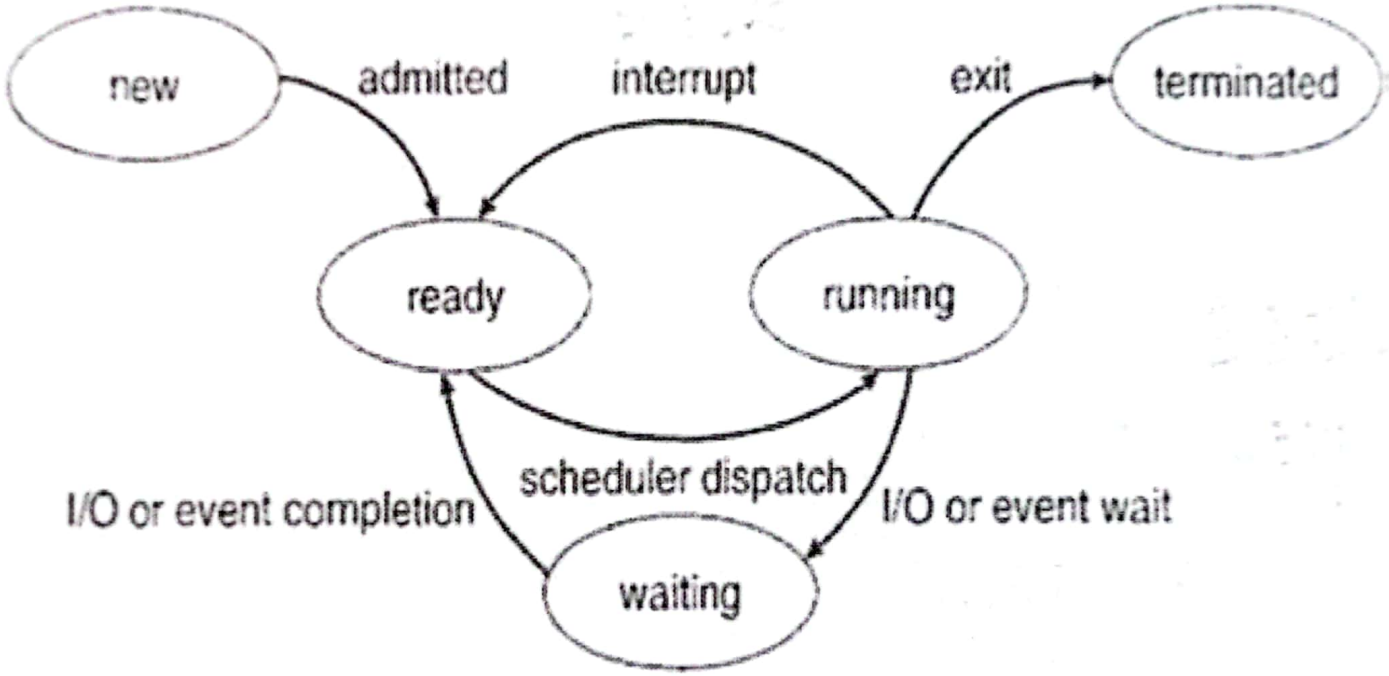


Figure 2: Diagram of process states

Note: Only one process can be running on any processor at any instant. Many processes may be ready and waiting, however.

PROCESS CONTROL BLOCK

- Each process is represented in the operating system by a data structure called a process control block (PCB). It is also called a task control block.
- It contains many pieces of information associated with a scientific process including:

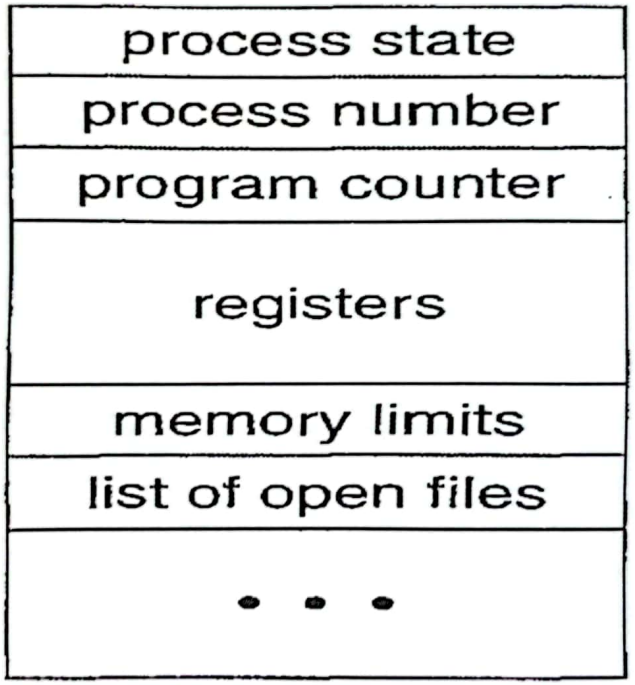


Figure 3: Process Control Block

THREADS

Process model so far	Modern OS have extended the process concept
A process is a program that performs a single thread of execution	Allow a process to have multiples threads of execution
When a process is running a Word processor program, a single thread of instructions is being executed	Performs more than one task at a time

Performs one task at a time. User cannot simultaneously type in characters and run the spell checker

PROCESS SCHEDULING

Multiprogramming objective:	Having some process running at all times to maximise CPU utilisation
Time sharing objective:	Switch the CPU among processes so frequently that users can interact with each other, while it runs

To meet these objectives

Process scheduler

Selects an available process (possibly from a set of several available process) for program execution on the CPU

Single processor system	multiple processor system
• Only one process runs at a time. Other processes will have to wait until the CPU is free and can be rescheduled	• Many processes run simultaneously

TIME SHARING VS MULTITASKING

- The main difference between time sharing and multitasking is that time sharing allows multiple users to share a computer resource simultaneously using multiprogramming and multitasking while multitasking allows a system to execute multiple tasks or processes simultaneously.
- While time sharing allows multiple users to use a computer system at a time, multitasking allows multiple tasks or processes to use a computer system at a time. Hence there is a functional difference between time sharing and multitasking.

SCHEDULING QUEUES

- Processes entering the system are put into a job queue, which consists of all processes in the system;
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called already queue.

PCB IN LINUX

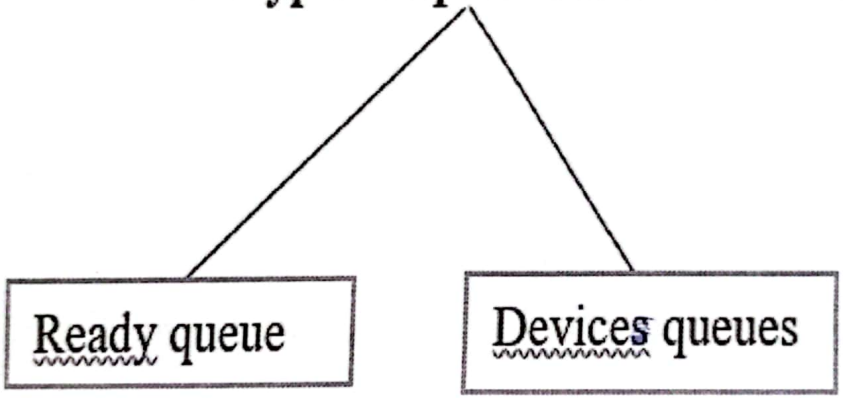
- Represented by the C structure, task-struct contains all the necessary information for representing a process including:
  - The state of the process
  - Scheduling and memory management information
  - List of open files
  - Pointers to the processes' parents and any of its children

Program counter :-	Points to next instruction to be executed for a given process
Current pointer :-	Points to the process currently executing on the system

PROCESS EXECUTION-SCHEDULING

- Process is allocated the CPU
- It executes for a while and eventually :
  - quits
  - is interrupted
  - waits for the occurrence of a particular event such as completion of an I/O request eg: disk
- The list of processes waiting for particular I/O device is called a device queue. Each device has its own device queue

Two types of queues are available



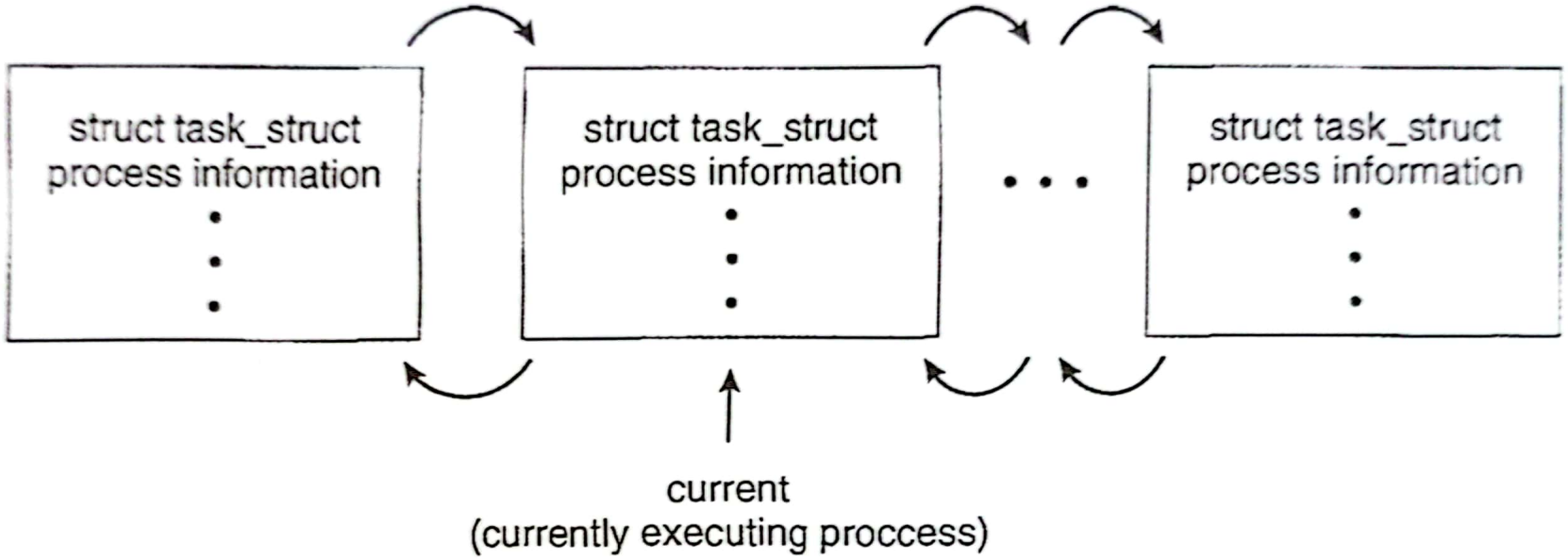


Figure 5: Active processes in Linux

PROCESS REPRESENTATION ON LINUX (DOUBLE LINKED LIST)

Within the Linux kernel, all active processes are represented using a doubly linked list of task-struct and the kernel maintains a pointer – current – to the process currently executing on the System

Linked list:

Two types

Double linked list (active Process representation in Linux)

Each node consists of :

- A data value
- A pointer to the next node
- A pointer to the previous node

Single linked list

Each node consists of :

- A data value
- A pointer to the next node

CHANGING STATES

- If current is a pointer to the process currently executing, its state is changed with the following:

  Current State = new_state

QUEUE REPRESENTATION (LINKED LIST)

A queue is generally stored as a linked list.

- Ready queue header contains: - pointers to the first and final PCUs in the list
- Each PCB includes : - A pointer field that points to the next PCB in ready queue

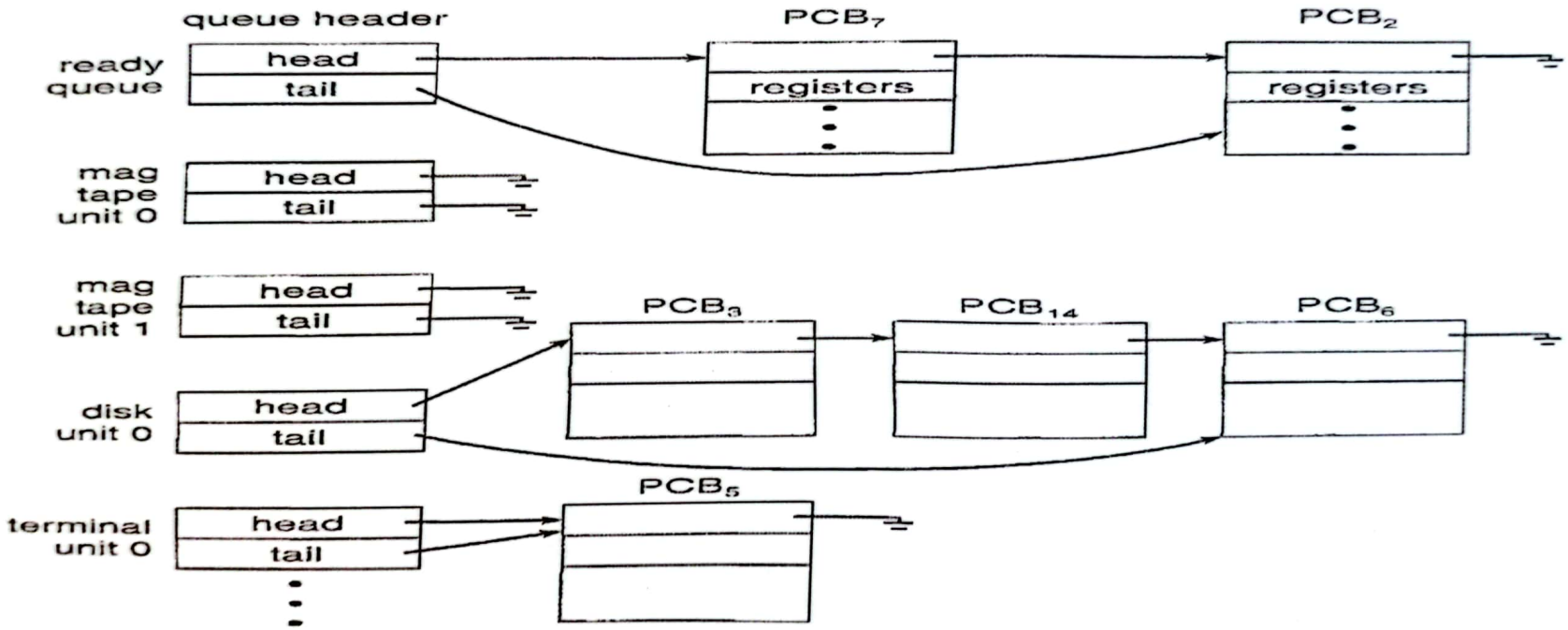


Figure 6: The ready queue and various I/O device queues.

PROCESS SCHEDULING REPRESENTATION

- Representation of process scheduling → queuing diagram
  - Rectangular box represents the queue ;
  - Circle represent the resources that serve the queues ;
  - Arrows indicate the flow of process in the System.
- A new process is initially put in the ready queue.
  - It waits there until it is selected for execution or is dispatched ;
  - Once the process is allocated the CPU and is executing one of several events could occur:
  - The process could issue an I/O request and then be placed in I/O queue ;
  - The process could create a subprocess and waits for the sub process termination ;
  - The process could be removed forcibly from the CPU as a result of an interrupt and be put back in the ready queue.

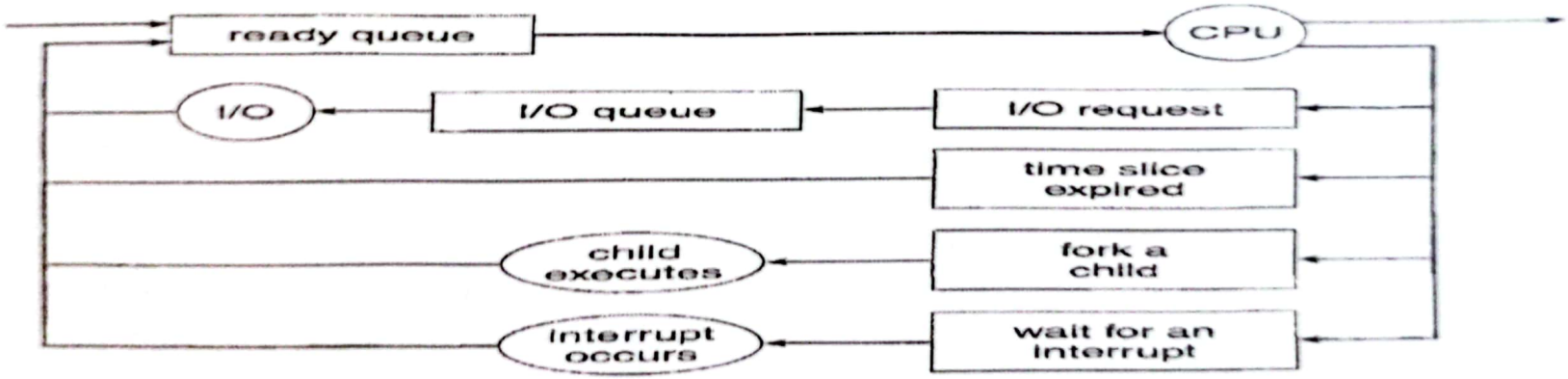


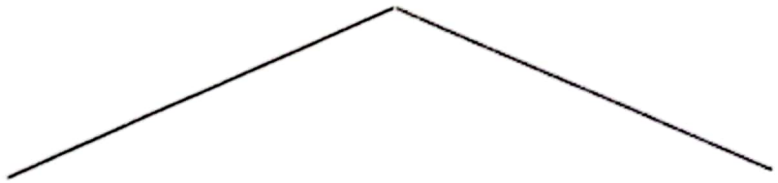
Figure 7: Queuing diagram representation of process scheduling.

A process continues this cycle until it terminates:
- It is removed from all queues
- Its PCB and resources deallocated.

SCHEDULER

- A process migrates throughout the scheduling queues throughout its lifetime
- The OS must select, for scheduling purposes processes from these queues in some fashion
- The selection process in carried out by the appropriate scheduler.

Types of scheduler



Batch System

Long-term scheduler (job scheduler)

- More processes submitted than can be executed immediately
- These processes are spooled to a mass-storage device (typically a disk) where they are kept for later execution
- The long-term scheduler selects processes from this pool and loads them into memory for execution

Time-sharing System

Short-term scheduler (CPU scheduler)

- Selects from among the processes that are ready to execute
- Allocates the CPU to one of them

LONG-TEM SCHEDULER

- Executes less frequently
- Minutes may separate the creation of one new process and next
- Controls the degree of multiprogramming (number of processes in memory)
- If the degree of multiprogramming in stable, then the average rate of process creation must be equal to the average departion rate of processes leaving the system
- May be invoked only when a process leaves the System
- Can there take a longer time to which process should be selected for execution
- Careful selection needs to be made since processes either I/O bound or CPU bound good mix required for System stability
- If all processes are I/O bound ready queue will always be empty
- If all processes are CPU bound I/O bound waiting queue will almost always be empty
- Combination of both

SHORT-TERM SCHEDULER

- Must select a new process for the CPU (execution) frequently
- Often executes at least once every 100 milliseconds
- Short time b/w executions this must be fast

⟹ • CPU will have little work.

⟹ • Devices will go unused and the system will be unbalanced

⟹ • System with best performance

TIME SHARING SYSTEMS

- In time-sharing systems such as Unix and Microsoft Windows Systems, the longterm scheduler is absent and every new process is simply put in memory for the short term scheduler.

SWAPPING

- Introduce an additional, intermediate level scheduling- medium term scheduler.
  - Remove processes from memory (and from active contention for the CPU, thus reduce the degree of multiprogramming.
  - Later reintroduce the process into memory, continue execution where it left off.

- **Necessity**
  - Improve the process mix
  - Requires memory to be freed up

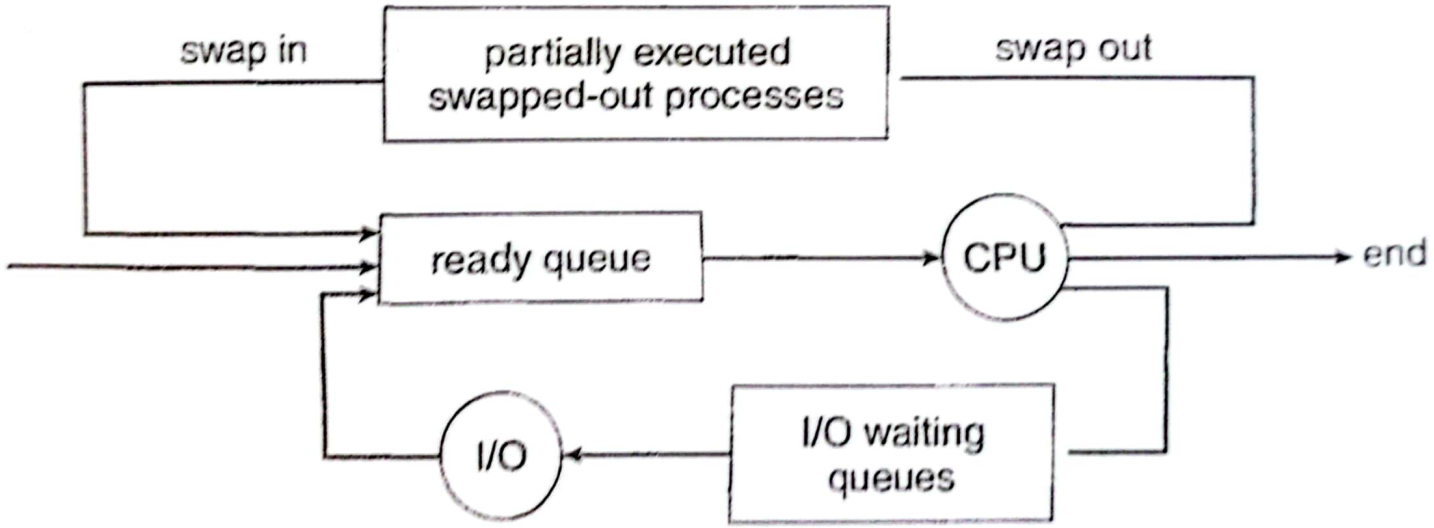


Figure 8: addition of medium-term scheduling to the queuing diagram

CONTEXT SWITCH

- Interrupts cause the OS to change a CPU from its current tasks and run a kernel routine
- When an interrupt occurs, the system needs to save the current content of the process running on the CPU so it can restore that context when its processing is done.
- The context is represented in the PCB of the process includes:
    - CPU registers
    - The process state
    - Memory management information
- Generally we perform a state save of the current state of the CPU (current process) be it kernel or user mode and then a state restore to resume operations. This task is know as context switch.
- 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.

OPERATIONS ON PROCESS

- Process- are identified according to a unique identifier (or Pid) typically an integer number:
    - Can execute concurrently in most systems
    - May be created or deleted dynamically
    - We explore mechanisms involved in creating processes

PROCESS CREATION

- A process may create several new processes via a create-process system call during the course of execution,
- The creating process is called a parent process.
- The new processes are called children of that process, which may in turn create other processes forming a tree of processes.

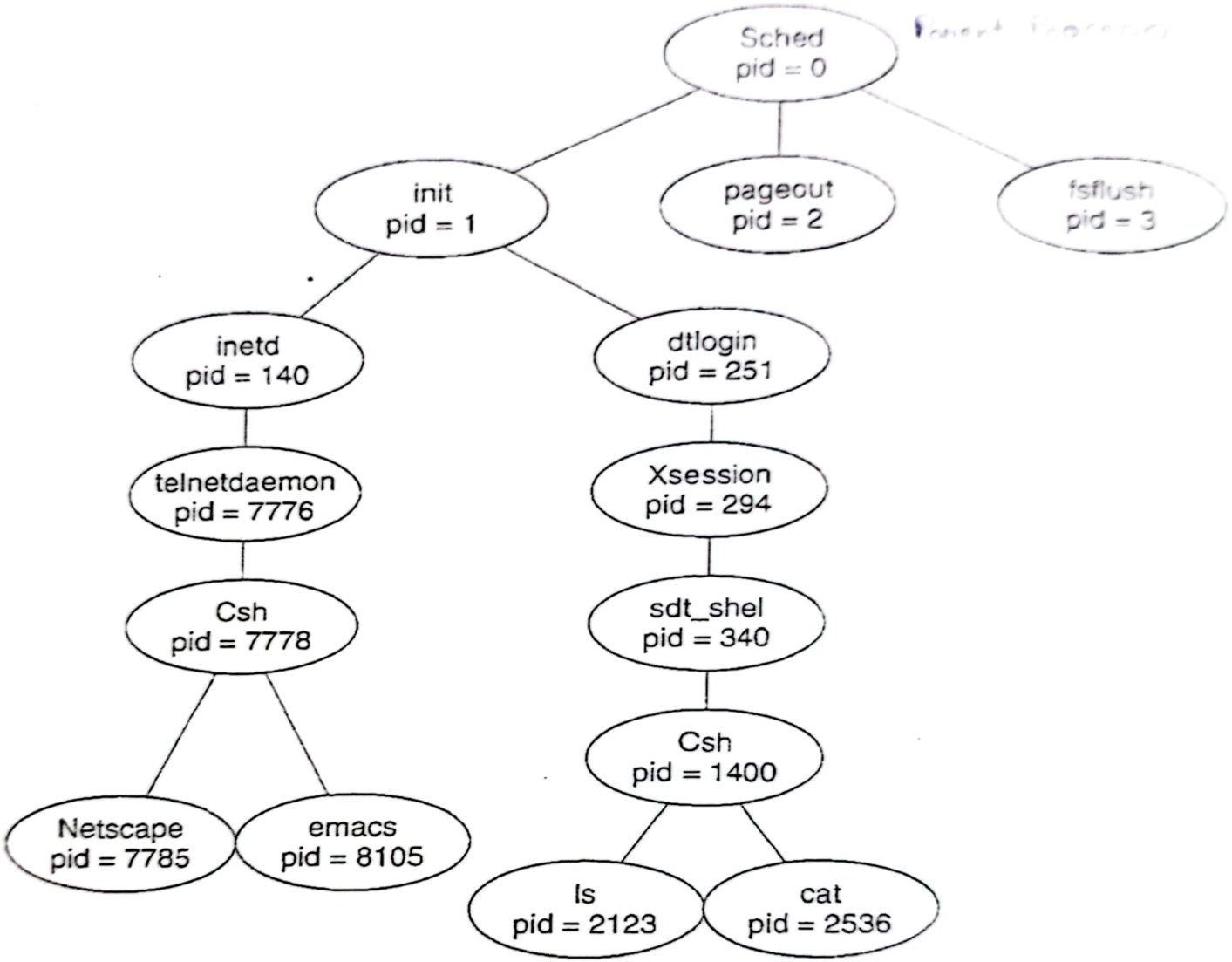
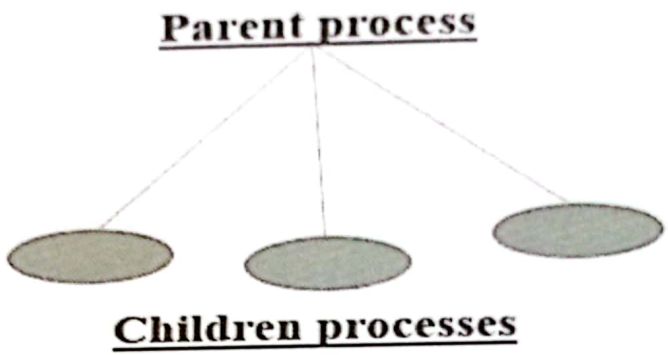


Figure 9: A tree of processes on a typical solaris system

RESOURCES NEEDED BY A PROCESS TO ACCOMPLISH ITS TASK

- CPU time
- Memory
- Files
- I/O devices



- Obtain its resources directly from the OS
- Constrained to subset of resources of the parent process.
  (helps prevent any process from overloading the system by creating too many sub processes, including initialization data).

Parent-child processes two possibilities exist in terms of:

Execution:	1. The parent continues to execute concurrently with children
	2. The parent waits until some or all of its children have terminated
Address space:	1) The child process is a duplicate of the parent (it has the same program and data as the parent)
	2) The child process has a new program loaded into it

```c
#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;
}
```

Figure 10: creating a separate process using UNIX fork () system call

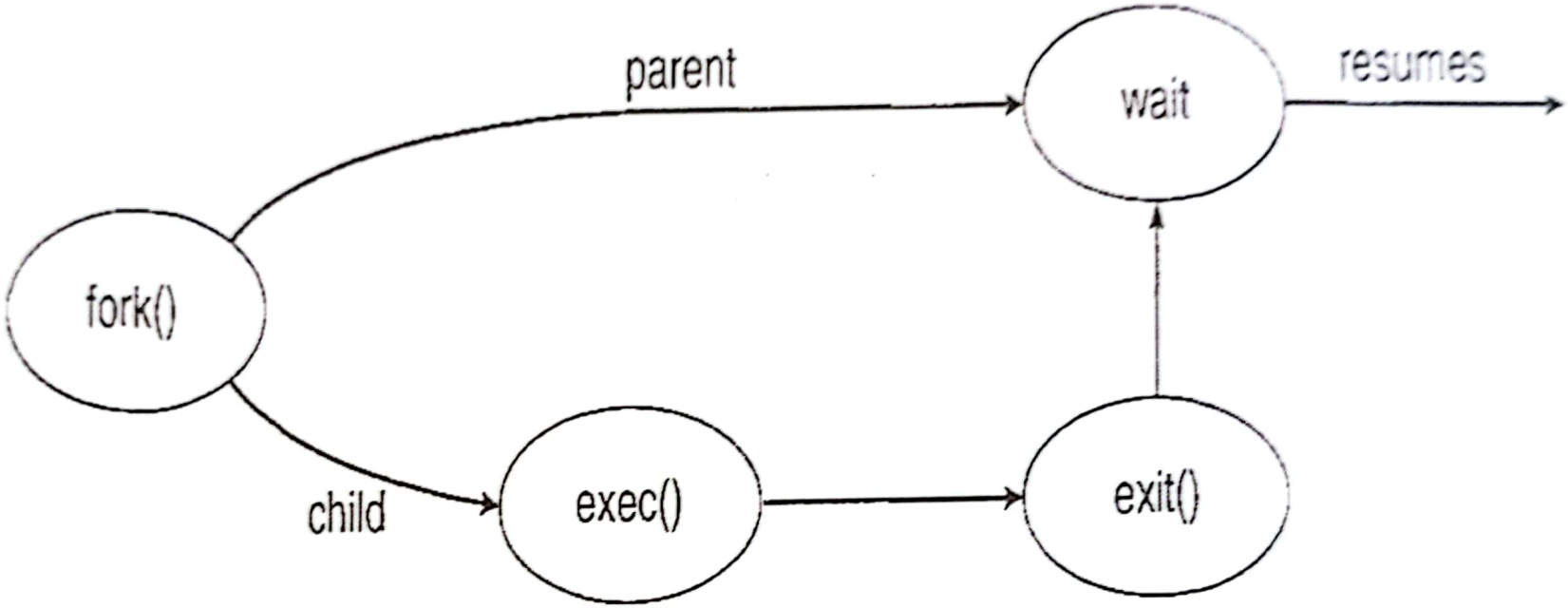


Figure 11: Process creation using fork () system call

QUESTIONS/CLARIFICATIONS