Processes

Reading: Section 2.1 of Textbook (Tanenbaum)

What is a Process?

The program in execution.
A program is an inanimate entity; only when a processor “breathes life” into it does it become the “active” entity, we call a process.

Programs and Processes

A process is different than a program.

Consider the following analogy

Scenario-1: A computer scientist is baking a birthday cake for his daughter

- Computer scientist - CPU
- Birthday cake recipe - program
- Ingredients - input data
- Activities: - processes
  - reading the recipe
  - fetching the ingradients
  - backing the cake

Programs and Processes

Scenario-2: Scientist's son comes running in crying, saying he has been stung by a bee.

- Scientist records where he was in the recipe
- Reach first aid book and materials
- Follow the first aid action (high priority job)
- On completion of aid, cake baking starts again from where it was left

- the state of running process saved
- Another process fetched
- Processor switched for new process
- Completion of high priority job & return back to the last one

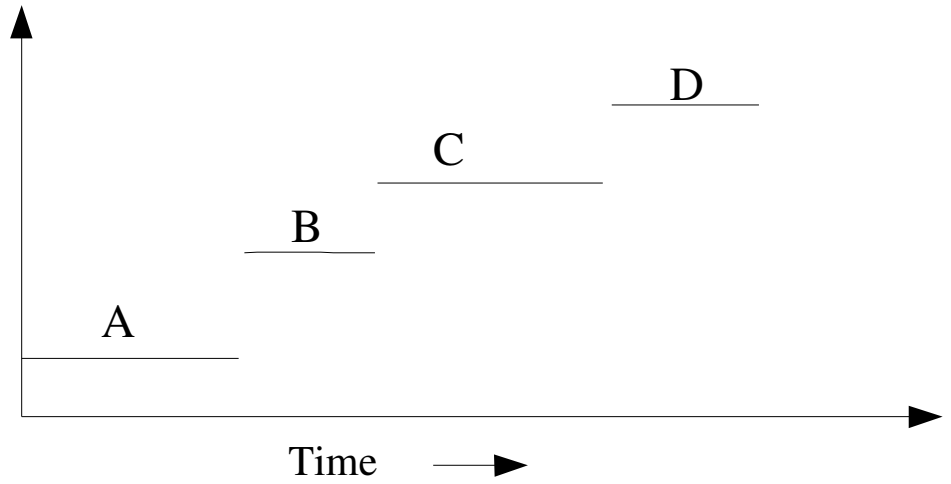
A process is an activity of some kind, it has program, input, output and state.

Process Models

- Uniprogramming
- Multiprogramming
- Multiprocessing

Uniprogramming

Only one process at a time.



Examples: Older systems
Advantages: Easier for OS designer
Disadvantages: Not convenient for user and poor performance

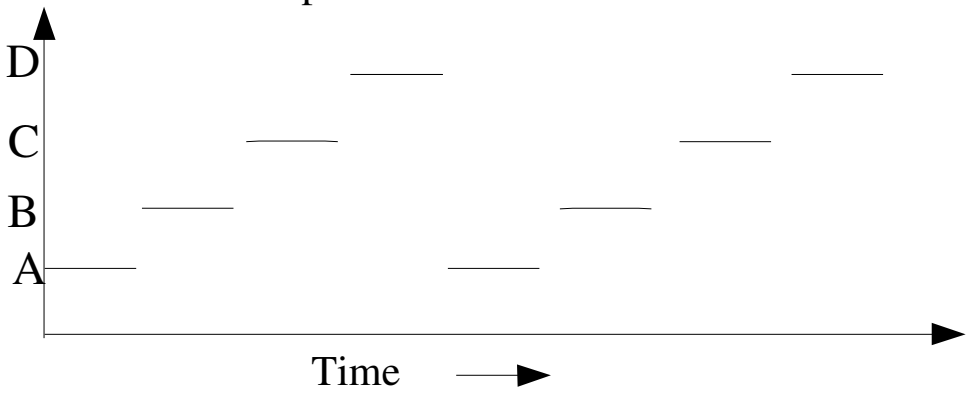
Multiprogramming

Multiple processes at a time.

OS requirements for multiprogramming:

Policy: to determine which process is to schedule.

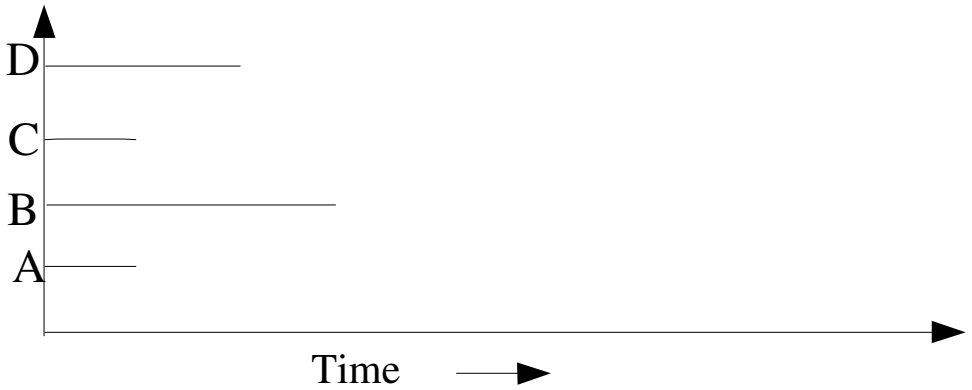
Mechanism: to switch between the processes.



Examples: Unix, WindowsNT
Advantages: Better system performance and user convenience.
Disadvantages: Complexity in OS

Multiprocessing

System with multiple processors.



Process States

A process goes through a series of discrete process states.

Running state:

- Process executing on CPU.
- Only one process at a time.

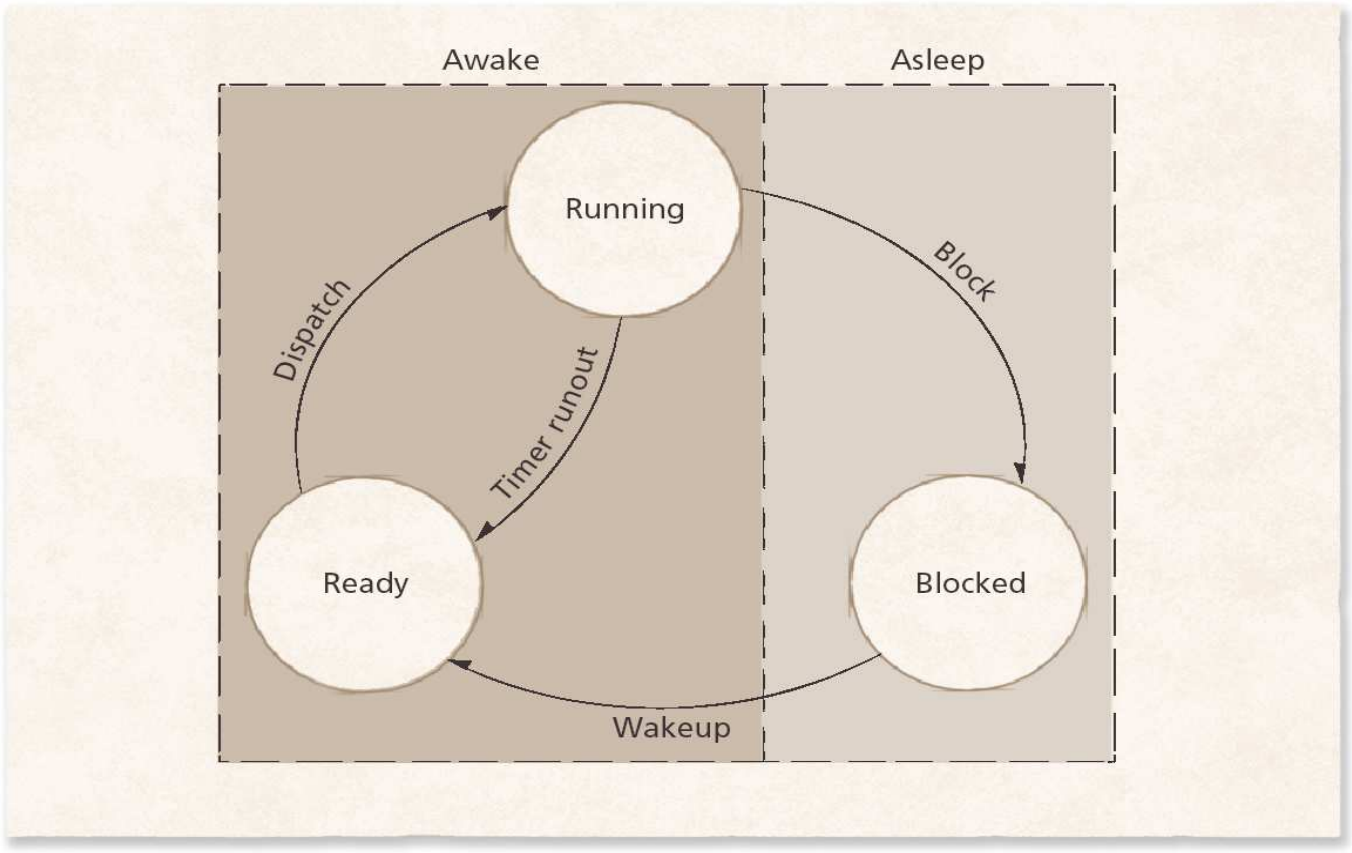
Ready state:

- Process that is not allowed to CPU but is ready to run.
- a list of processes ordered based on priority.

Blocked state:

- Process that is waiting for some event to happen (e. g. I/O completion events).
- a list of processes (no clear priority).

Process States



Process state transitions diagram

Process State Transitions

When a job is admitted to the system, a corresponding process is created and normally inserted at the back of the ready list.

When the CPU becomes available, the process is said to make a state transition from ready to running.

dispatch(processname): ready -> running.

To prevent any one process from monopolizing the CPU, OS specify a time period (quantum) for the process. When the quantum expire makes state transition running to ready.

timerrunout(processname): running -> ready.

When the process requires an I/O operation before quantum expire, the process voluntarily relinquishes the CPU and changed to the blocked state.

block(processname): running -> block.

When an I/O operation completes. The process make the transition from block state to ready state.

wakeup(processname): blocked -> ready.

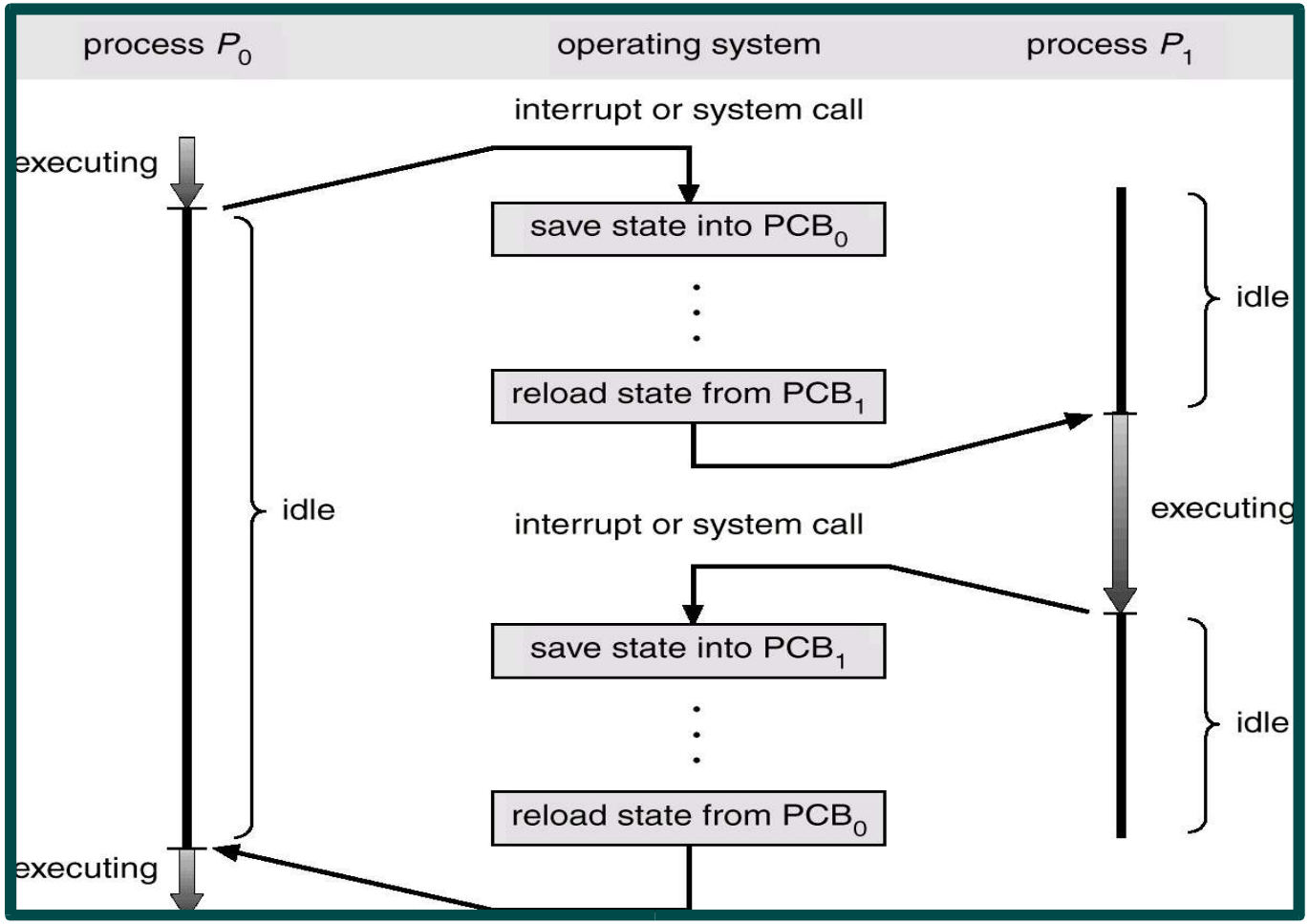
Process Control Block

Process must be saved when the process is switched from one state to another so that it can be restarted later as it had never been stopped.

The PCB is the data structure containing certain important information about the process -also called process table or processor descripter.

- *Process state*: running, ready, blocked.
- *Program counter*: Address of next instruction for the process.
- *Registers*: Stack pointer, accumulator, PSW etc.
- *Scheduling information*: Process priority, pointer to scheduling queue etc.
- *Memory-allocation*: value of base and limit register, page table, segment table etc.
- *Accounting information*: time limit, process numbers etc.
- *Status information:* list of I/O devices, list of open files etc.

Process Control Block



Operations on Processes

The processes in the system can execute concurrently, and they must be created and deleted dynamically. OS provide the mechanism for process *creation* and *termination*.

- Process Creation.
- Process Termination.

Process Creation

There are four principal events that cause the the process to be created:

» Systut

Process Creation

Two ways to create a new process

- Build a new one from scratch
  - Load specified code and data into memory.
  - Create and initialize PCB.
  - Put processes on the ready list.
- Colon an existing one (e.g. Unix fork() syscall)
  - Stop the current process and save its state.
  - Make copy of code, data, stack, and PCB.
  - Add new process PCB to ready list.

Unix Process Creation Ex.

```
#include<stdio.h>
int main(int argc, char *argv[])
        {
        int pid;
                pid = fork();    /* create new process */
                if(pid < 0) {    /* error occurred */
                        fprintf(stderr, "fork failed");
                        exit(-1);
                                }
                else if(pid == 0){     /* child process */
                        execlp("/bin/ls", "ls", Null); }
                else {                          /* parent process */
                        wait(Null);
                        printf("Child Complete");
                        exit(0);
                        }
                return 0;
        }
```

Process Termination

Process are terminated on the following conditions

1. Normal exit.
2. Error exit.
3. Fatal error.
4. Killed by another process.

Example:

In Unix the normal exit is done by calling a *exit* system call. The process return data (output) to its parent process via the *wait* system call.

kill system call is used to kill other process.

Home Work

HW #2:

1. Q. No. 1, 2 & 3 from the Textbook (Tanenbaum).
2. What are disadvantages of too much multiprogramming?
3. List the definitions of process.
4. For each of the following transitions between the process states, indicate whether the transition is possible. If it is possible, give an example of one thing that would cause it.
   a) Running -> Ready
   b) Running -> Blocked
   c) Blocked -> Running

Reading: Section 2.2 of Textbook (Tanenbaum)