PROCESS MODEL

At the end of this lecture cum class discussion, the class should have in-depth knowledge of the following

- Characteristic definition of a process
- Process/Task Control Block: 7 basic contents et
- Process States
- Threads
- Process scheduling
- Scheduling Queue
- Context Switch
- Operations on a process (Creation and termination)
- and Inter Process Communication

Assumptions

- We are assuming a **multiprogramming** OS that can switch from one process to another.
- Sometimes this is called *pseudo parallelism* since one has the illusion of a parallel processor.
- The other possibility is *real parallelism* in which two or more processes are actually running at once because the computer system is a parallel processor, i.e., has more than one processor.

The Process Model

Even though in actuality there are many processes running at once, the OS gives each process the *illusion* that it is running alone.

- **Virtual time**: The time used by just this processes. Virtual time progresses at a rate independent of other processes. Actually, this is false, the virtual time is typically incremented a little during systems calls used for process

switching; so if there are more other processors' 'overhead'' virtual time occurs.

- **Virtual memory**: The memory as viewed by the process. Each process typically believes it has a contiguous chunk of memory starting at location zero. Of course this can't be true of all processes (or they would be using the same memory) and in modern systems it is actually true of no processes (the memory assigned is not contiguous and does not include location zero).

  Think of the individual modules that are input to the linker. Each numbers its addresses from zero; the linker eventually translates these relative addresses into absolute addresses. That is the linker provides to the assembler a virtual memory in which addresses start at zero.

Virtual time and virtual memory are examples of **abstractions** provided by the operating system to the user processes so that the latter ``sees'' a more pleasant virtual machine than actually exists.

Process

"A process is a program in execution". This has become the common response to the question "What is a Process? However, given our background, it's important we define a process from the technical point of view. So I ask again, what really is a process?

A process is an active instance of a program currently under execution or capable of execution if given desired resources and processor is available. It has independent private data, program counter and is capable of communication with other processes.

Each process is represented in the OS by a Process Control Block, PCB. A PCB Contains the following information:

1. Process states: New/ Initialized, Running, Waiting, Ready, Terminated
2. Program Counter: hold the address of the nest instruction to be executed
3. C.P.U Registers: Accumulators, Index registers, GPR, Stack pointers, Conditional code information. Along with the program counter, state information is saved when an interrupt occurs.
4. CPU Scheduling Information: Process Priority, Pointer to scheduling queues.
5. Memory Management: Value of base and limit registers and page/segment tables.
6. Accounting Information: CPU time usage, Process numbers, time limit s et al.
7. I/O status information: List of i/o devices, open files among others.

Threads

If you went to a Secondary school in west Africa, you will remember the Daltons Atomic Theory; and the first goes thus “… atoms are the smallest indestructible unit of an element”. However, that theory was modified due to nuclear explosions that led to the creation of neutron, proton and electron. In the same way, you may have believed that process is the smallest unit of a program in execution. I shall now ***burst your brain.*** Threads are the smallest unit of a program in execution. And a process can spawn as many threads as possible. Take for instance, Your Microsoft word is a process in execution but one of its threads is the spell checker or what you comfortably call auto correct. The auto correct function is there to make sure that when you type your application letters, you don't use grammatical and spelling errors to humiliated the effort of your teachers. We shall look further into threads in subsequent classes.

Process Scheduling.

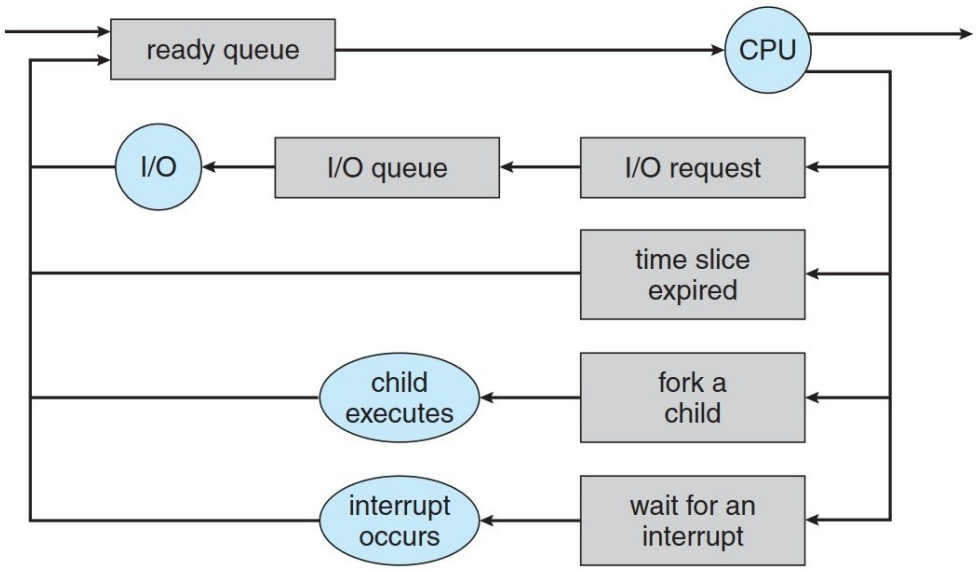
Have I ever told you the objective of multi Programming? Well, the objective of Multiprogramming is to have some (at least one) process running at all times in order to maximize the CPU usage (Remember the saying “nature abhors vacuum”). The objective of time sharing is to switch the CPU among many processes so frequently that users can interact with each program seamlessly. Given these objectives, how do we go about meeting this? Process Scheduler is the key!

Process scheduler is primarily responsible for ensuring the required and right process has the CPU’s attention at the right time. In other classes you have learnt about different scheduling techniques, let’s see how this works.

When a process enters into the system, it is placed on the job queue. The job queues are a linked list that consist of all processes in the system.

Now, processes that are residing in the main memory and are ready and waiting to execute are kept in the ready queue. The ready queue contains pointers to the first and final PCB in the list. Each PCB includes a pointer that points to the next PCB in the ready queue.

Device Queue: When a process runs, and it requires a resource/device it is put in the device queue. The list of processes waiting for a particular device is called the device queue. Each device has its own queue.



Queueing-diagram representation of process scheduling.

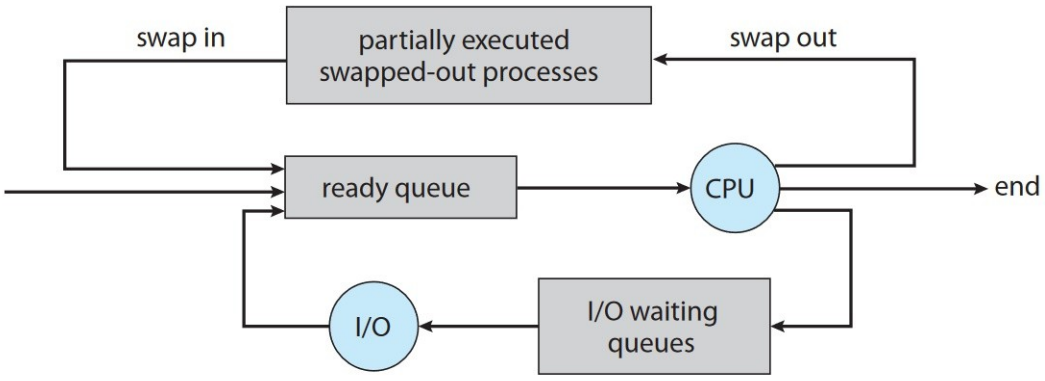
A process is said to be dispatched if it is selected for execution. After execution, the PCB and resources are deallocated.

Scheduler

The selection Process from one queue to another is done by the scheduler. Processes may be I/O bound or CPU bound. A process is said to be I/O bound if it requires more of I/O devices than CPU during the course of its execution. I believe with this few points of mine, you can tell what A CPU bound process is.

Just a two –aside

Have you heard of Swapping? DO you know what it is related to process scheduling? Well, if you don't just look at the diagram below. Swapping is the removal of a process from memory (and active contention of the CPU). this reduces the degree of multi Programming. Now What's the degree of Multi Programing?



Addition of medium-term scheduling to the queueing diagram.

Context Switch

Context Switching is a kernel function that removes the CPU from a process and transfers execution to another process. It entails the saving of the state of the current process and a restore of the state of a different process. Context switching is a waste of CPU time as no meaningful job is done during switching. Switching Speed id determined by hardware, Memory speed, no of registers to be copied, existence of special instructions.

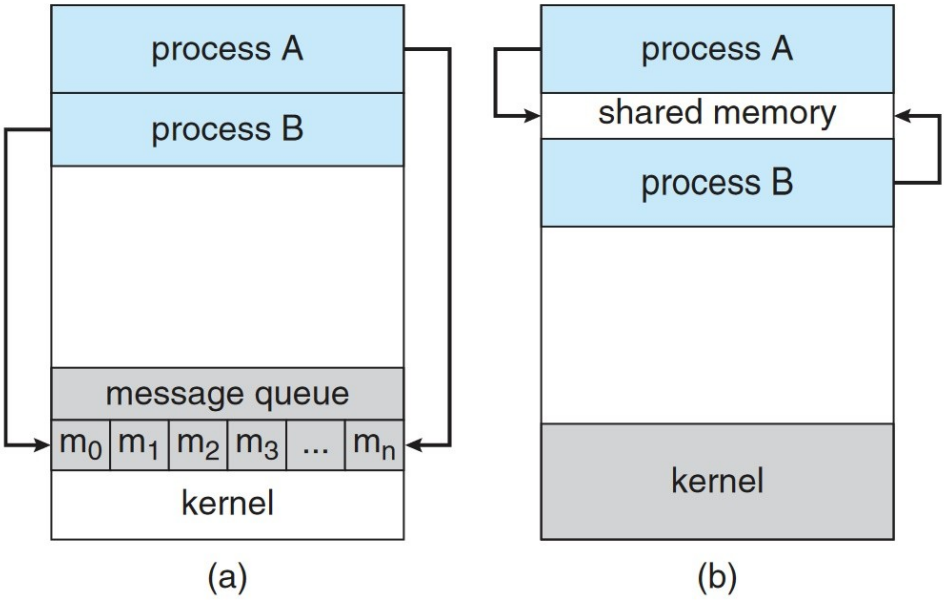
Inter-process Communication

A process may be independent or a cooperating process. A process may choose to communicated for the following reasons.

- ✓ Information sharing
- ✓ Computing Speed
- ✓ Modularity
- ✓ Convenience

There are two fundamental models of inter-process Communication

- Shared Memory: Faster, Bound and Unbound Buffer
- Message Passing: Small data, distributed systems
-



Communications models. (a) Message passing. (b) Shared memory.

Process Creation

From the users or external viewpoint there are several mechanisms for creating a process.

1. System initialization, including daemon processes.
2. Execution of a process creation system call by a running process.
3. A user request to create a new process.
4. Initiation of a batch job.

But looked at internally, from the system's viewpoint, the second method dominates. Indeed, in unix only one process is created at system initialization (the process is called *init*); all the others are children of this first process.

Why have init? That is why not have all processes created via method 2?
Ans: Because without init there would be no running process to create any others.

Process Termination

Again from the outside there appear to be several termination mechanisms.

1. Normal exit (voluntary).
2. Error exit (voluntary).
3. Fatal error (involuntary).
4. Killed by another process (involuntary).

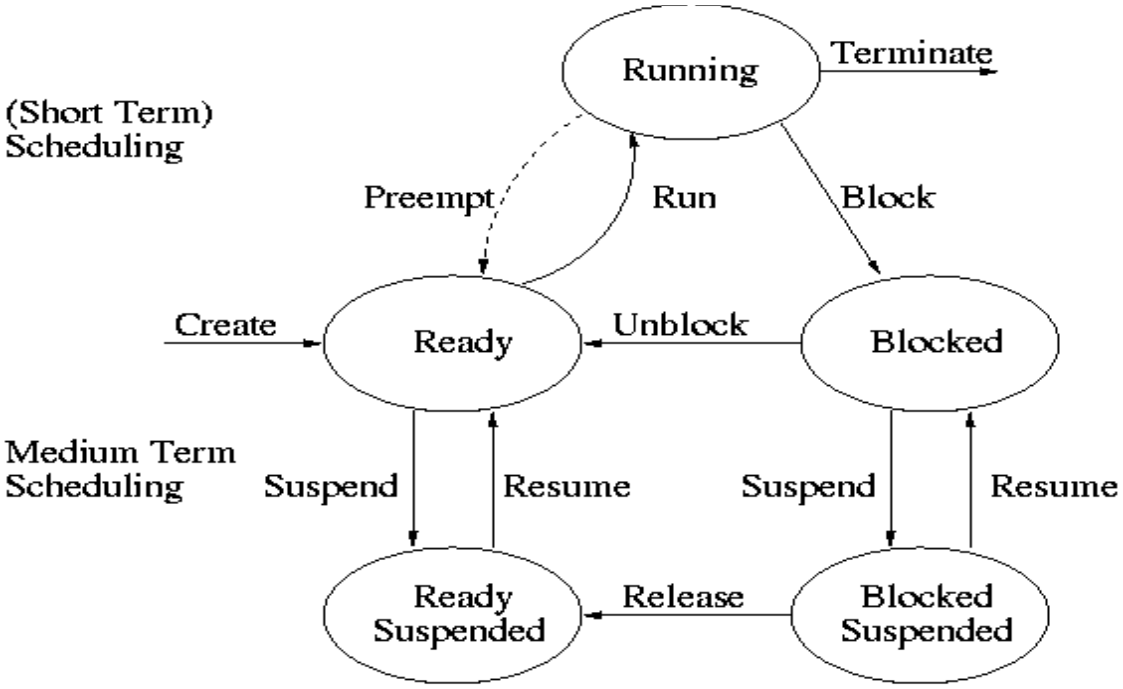
And again, internally the situation is simpler. In Unix terminology, there are two system calls *kill* and *exit* that are used. Kill (poorly named in my view) sends a signal to another process. If this signal is not caught (via the **signal** system call) the process is terminated. There is also an ``uncatchable'' signal. Exit is used for self-termination and can indicate success or failure.

Process States and Transitions

The diagram on the right contains much information.

- Consider a running process P that issues an I/O request
    - The process blocks
    - At some later point, a disk interrupt occurs and the driver detects that P's request is satisfied.

- P is unblocked, i.e. is moved from blocked to ready
- At some later time, the operating system looks for a ready job to run and picks P.



- A preemptive scheduler has the dotted line preempt; A non-preemptive scheduler doesn't.
- The number of processes changes only for two arcs: create and terminate.
- Suspend and resume are medium term scheduling
  - Done on a longer time scale.
  - Involves memory management as well.
  - Sometimes called two level scheduling.

THE FIVE PROCESS STATES

These process models contain five states that are involved in the lifecycle of a process.

- New
- Ready
- Running
- Blocked/ waiting
- Exit.

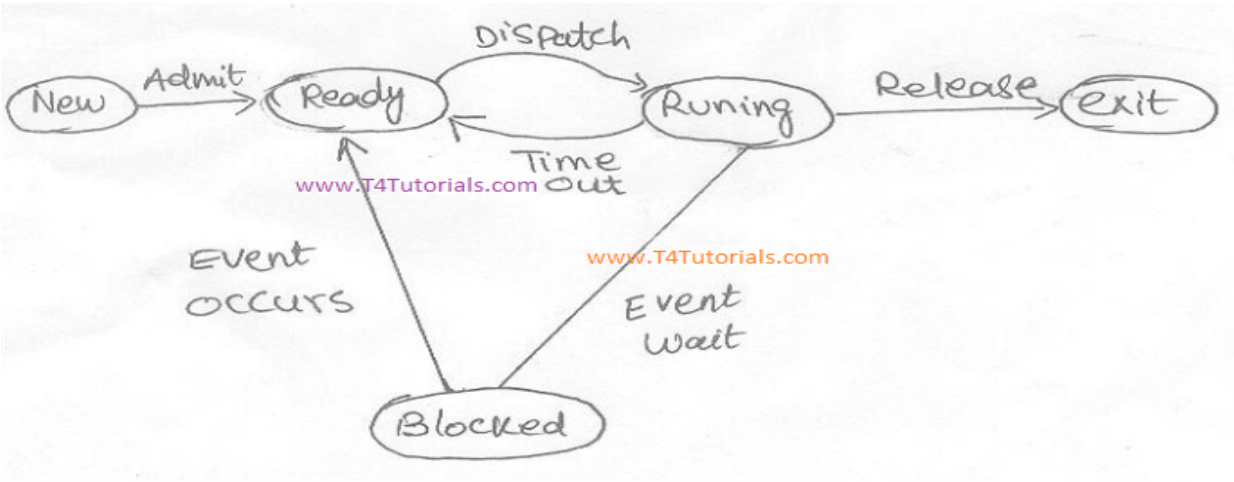
New: A process has been created but has not yet been admitted to the pool of executable processes.

Ready: Processes that are prepared to run if given an opportunity. That is, they are not waiting on anything except the CPU availability.

Running: The process that is currently being executed. (Assume single processor for simplicity.)

Blocked/waiting: A process that cannot execute until a specified event such as an IO completion occurs.

Exit: A process that has been released by OS either after normal termination or after abnormal termination (error).



Study Questions

1. Give the technical Definition of a process

2. Using an illustration, explain the process cycle
3. What is a Task Control Block and what are functions?
4. Define Context Switch, elucidate the factors affecting the speed of switch and state reason(s) why the speed of switching must be increased.
5. What is a Scheduler? Why do we need one in an Operating System?
6. Do you think Process Should Communicate? If Yes, State the fundamental types and reasons for communication. If No, State your reasons and design a sketch an architecture that surpasses the existing OS structure.
7. With respect to a scheduler, give 3 types of queues and define them.
8. Differentiate Process and threads
9. Similarities or differences between Process and Jobs and Tasks
10. Using your pen and paper, Create the Process “**drinking”** and terminate the created process using any OS Mechanism of Choice.