2.1 PROCESSES

2.1 Process Concept

- A process is an instance of a program in execution.
- Batch systems work in terms of "jobs". Many modern process concepts are still expressed in terms of jobs, (e.g. job scheduling), and the two terms are often used interchangeably.

2.1.1 The Process

Process memory is divided into four sections as shown in Figure below:

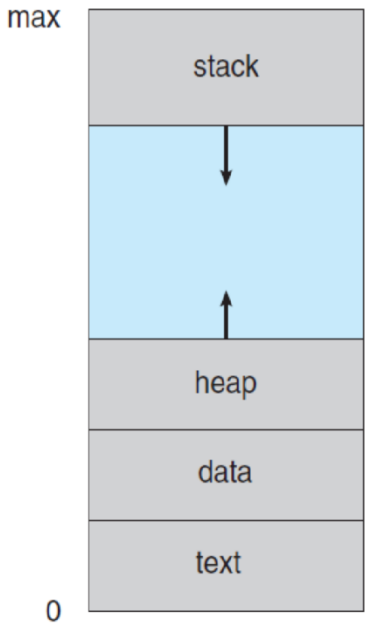


Figure 3.1 Process in memory.

- The text section comprises the compiled program code, read in from non-volatile storage when the program is launched.
- The data section stores global and static variables, allocated and initialized prior to executing main.
- The heap is used for dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables. Space on the stack is reserved for local variables when they are declared (at function entrance or elsewhere, depending on the language), and the space is freed up when the variables go out of scope. Note that the

stack is also used for function return values, and the exact mechanisms of stack management may be language specific.

- Note that the stack and the heap start at opposite ends of the process's free space and grow towards each other. If they should ever meet, then either a stack overflow error will occur, or else a call to new or malloc will fail due to insufficient memory available.
- When processes are swapped out of memory and later restored, additional information must also be stored and restored. Key among them are the program counter and the value of all program registers.

2.1.2 Process State

Processes may be in one of 5 states, as shown in Figure below.

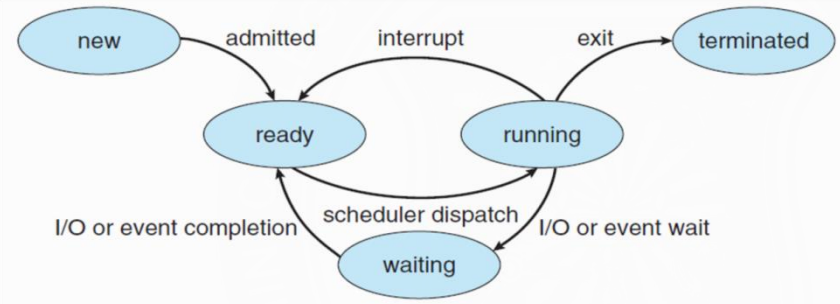


Figure 3.2 Diagram of process state.

New - The process is in the stage of being created.

Ready - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.

Running - The CPU is working on this process's instructions.

Waiting - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.

Terminated - The process has completed.

The load average reported by the "w" command indicate the average number of processes in the "Ready" state over the last 1, 5, and 15 minutes, i.e. processes who have everything they need to run but cannot because the CPU is busy doing something else.

Some systems may have other states besides the ones listed here.

2.1.3 Process Control Block

For each process there is a Process Control Block, PCB, which stores the following (types of) process-specific information, as illustrated in Figure (Specific details may vary from system to system.)

Process State - Running, waiting, etc., as discussed above.

Process ID, and parent process ID.

CPU registers and Program Counter - These need to be saved and restored when swapping processes in and out of the CPU.

CPU-Scheduling information - Such as priority information and pointers to scheduling queues.

Memory-Management information - E.g. page tables or segment tables.

Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.

I/O Status information - Devices allocated, open file tables, etc.

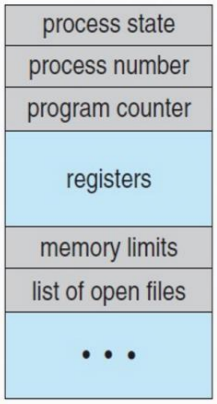


Figure 3.3 Process control block (PCB).

2.1.4 Threads

Modern systems allow a single process to have multiple threads of execution, which execute concurrently.

2.2 Process Scheduling

The two main objectives of the process scheduling system are to keep the CPU busy at all times and to deliver "acceptable" response times for all programs, particularly for interactive ones.

The process scheduler must meet these objectives by implementing suitable policies for swapping processes in and out of the CPU.

2.2.1 Scheduling Queues

- All processes are stored in the **job queue.**
- Processes in the Ready state are placed in the **ready queue.**
- Processes waiting for a device to become available are placed in **device queues**. There is generally a separate device queue for each device.
- Other queues may also be created and used as needed.

.

- Job queue − This queue keeps all the processes in the system
- Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.

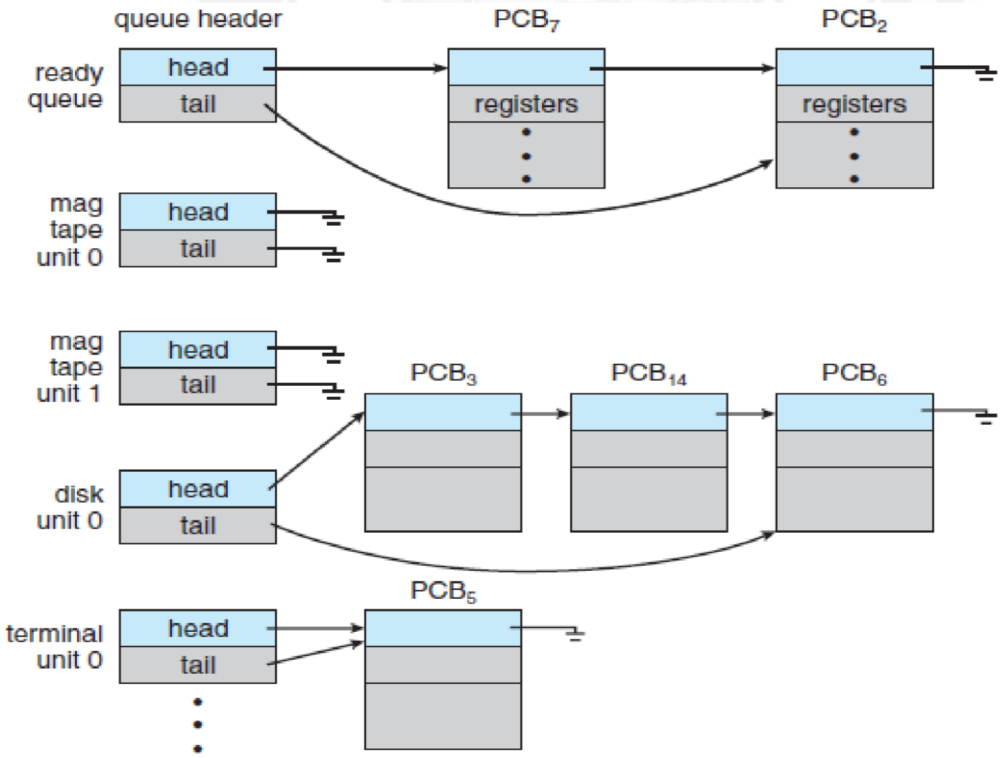


Figure 3.5 The ready queue and various I/O device queues.

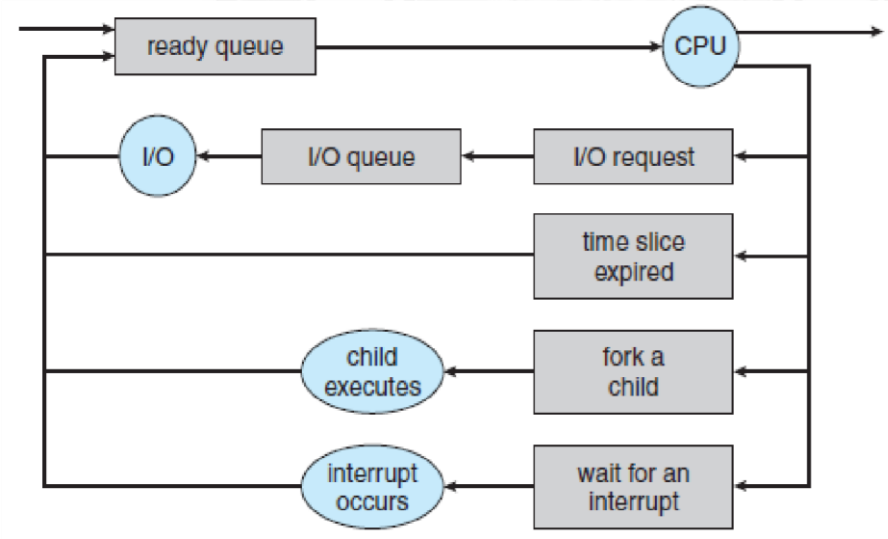
2.2.2 Schedulers

• A **long-term scheduler** is typical of a batch system or a very heavily loaded system. It runs infrequently, It is also called a job scheduler. It selects processes from the queue and loads them into memory for execution.

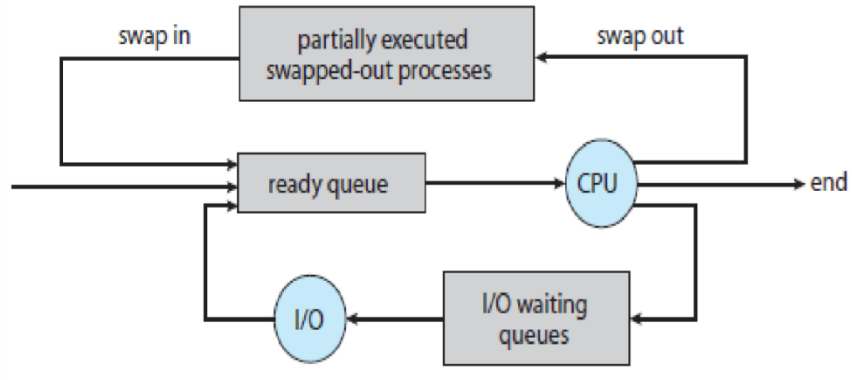
• The **short-term scheduler**, or CPU Scheduler, runs very frequently, on the order of 100 milliseconds, and must very quickly swap one process out of the CPU and swap in another one. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

• Some systems also employ a **medium-term scheduler**. When system loads get high, this scheduler will swap one or more processes out of the ready queue system for a few seconds, in order to allow smaller faster jobs to finish up quickly and clear the system. See the differences in Figures below.

• An efficient scheduling system will select a good **process mix** of **CPU-bound** processes and **I/O bound**processes.



Queueing-diagram representation of process scheduling

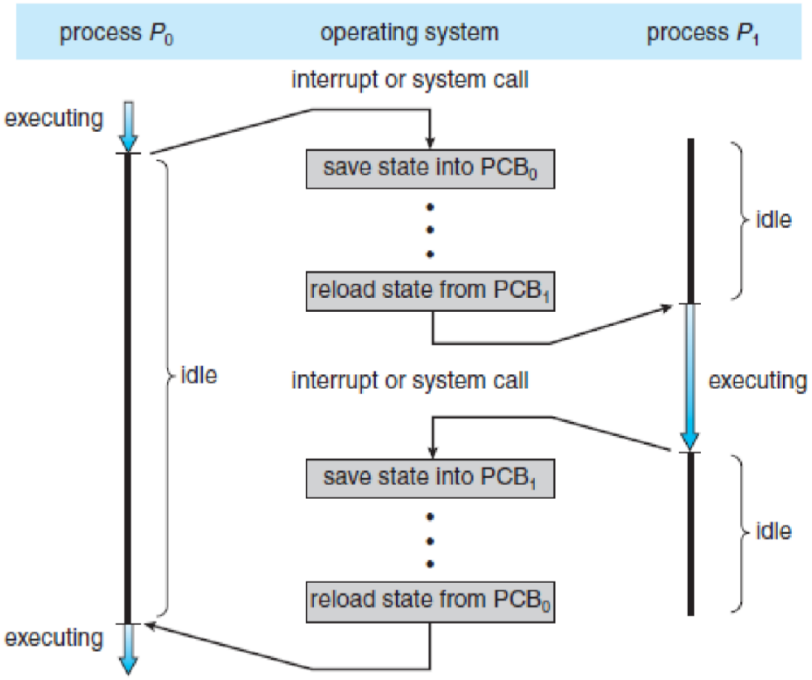


Addition of a medium-term scheduling to the queuing diagram

2.2.3 Context Switch

Definition: Switching the CPU between processes is called context switch. A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block

- Whenever an interrupt arrives, the CPU must do a **state-save** of the currently running process, then switch into kernel mode to handle the interrupt, and then do a **state-restore** of the interrupted process.
- Similarly, a **context switch** occurs when the time slice for one process has expired and a new process is to be loaded from the ready queue. This will be initiated by a timer interrupt, which will then cause the current process's state to be saved and the new process's state to be restored.
- Saving and restoring states involves saving and restoring all of the registers and program counter(s), as well as the process control blocks described above.
- Context switching happens VERY VERY frequently, and the overhead of doing the switching is just lost CPU time, so context switches (state saves & restores) need to be as fast as possible.


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
reload state from PCB_1
idle
idle
interrupt or system call
executing
save state into PCB_1
reload state from PCB_0
idle
executing