Process

The Process

A process is a program in execution.

- A program is a passive entity, such as a file containing a list of instructions stored on disk
- A process is an active entity, with a program counterspecifying the next instruction to execute and a set of associated resources
- A program becomes a process when an executable file is loaded into memory.
- Although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences

A process includes:

- program code(text section)
- program counter
- contents of processor's register
- proess stack (temporary data : return value ,parameters)
- data section (global variables)
- heap (the memory that is dynamically allocated during process run time0)

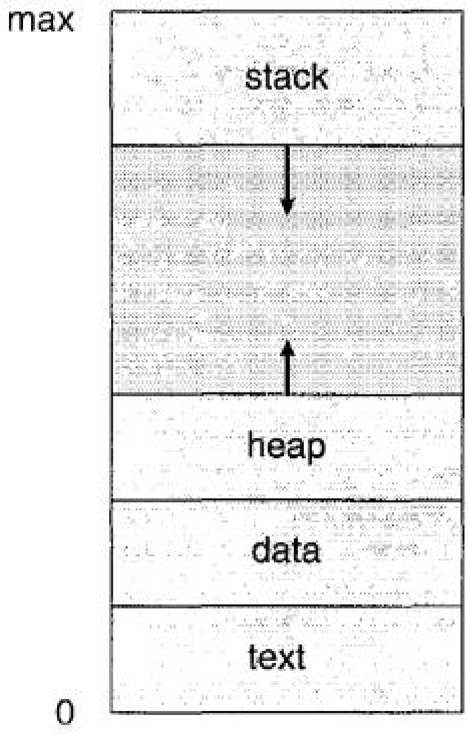

Figure 3.1 Process in memory.

Process State

As a process executes, it changes state.(五态模型)

- **New**. The process is being created.
- **Running**. Instructions are being excuted.
- **Waiting**. The process is waiting for some event to occur(such as an I/O completion or reception of a signal) (等待 I/O或者信号).
- **Ready**. The process is waiting to be assigned to a processor(by scheduler).
- **Terminated**. The process has finished execution.

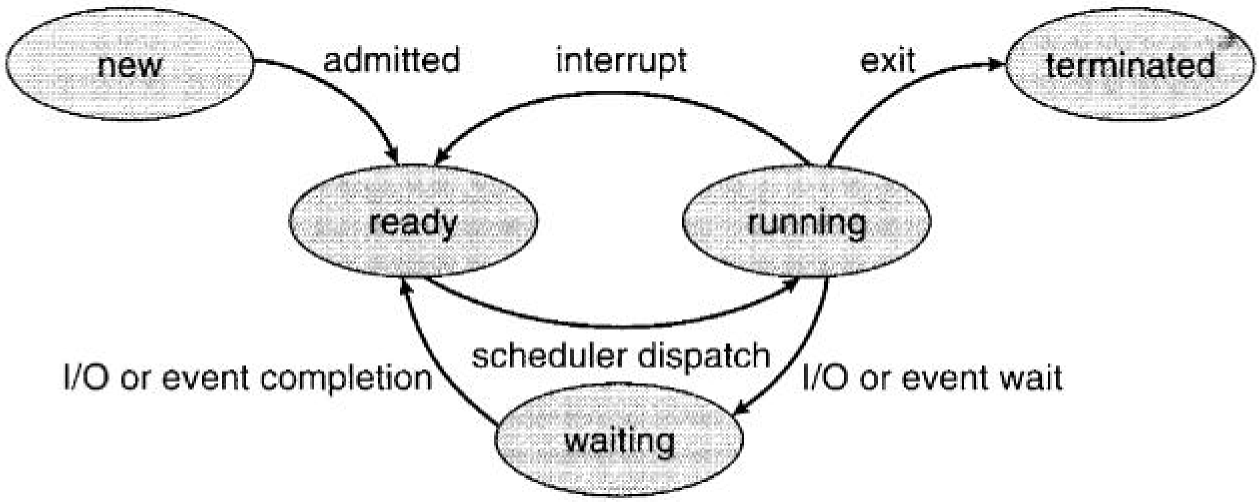

Figure 3.2 Diagram of process state.

It is important to realize that **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 process control block (PCB)-also called a task control block.

- **Process state**. The state may be new, ready, running, waiting, halted, and so on.
- **Program counter**. The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers** . Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
- **Cpu-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information.**
- **Accounting information**. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers,and so on.
- **I/O status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

(ps: Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.)

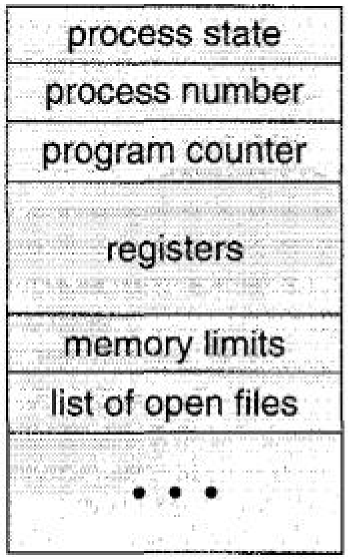

Figure 3.3 Process control block (PCB).

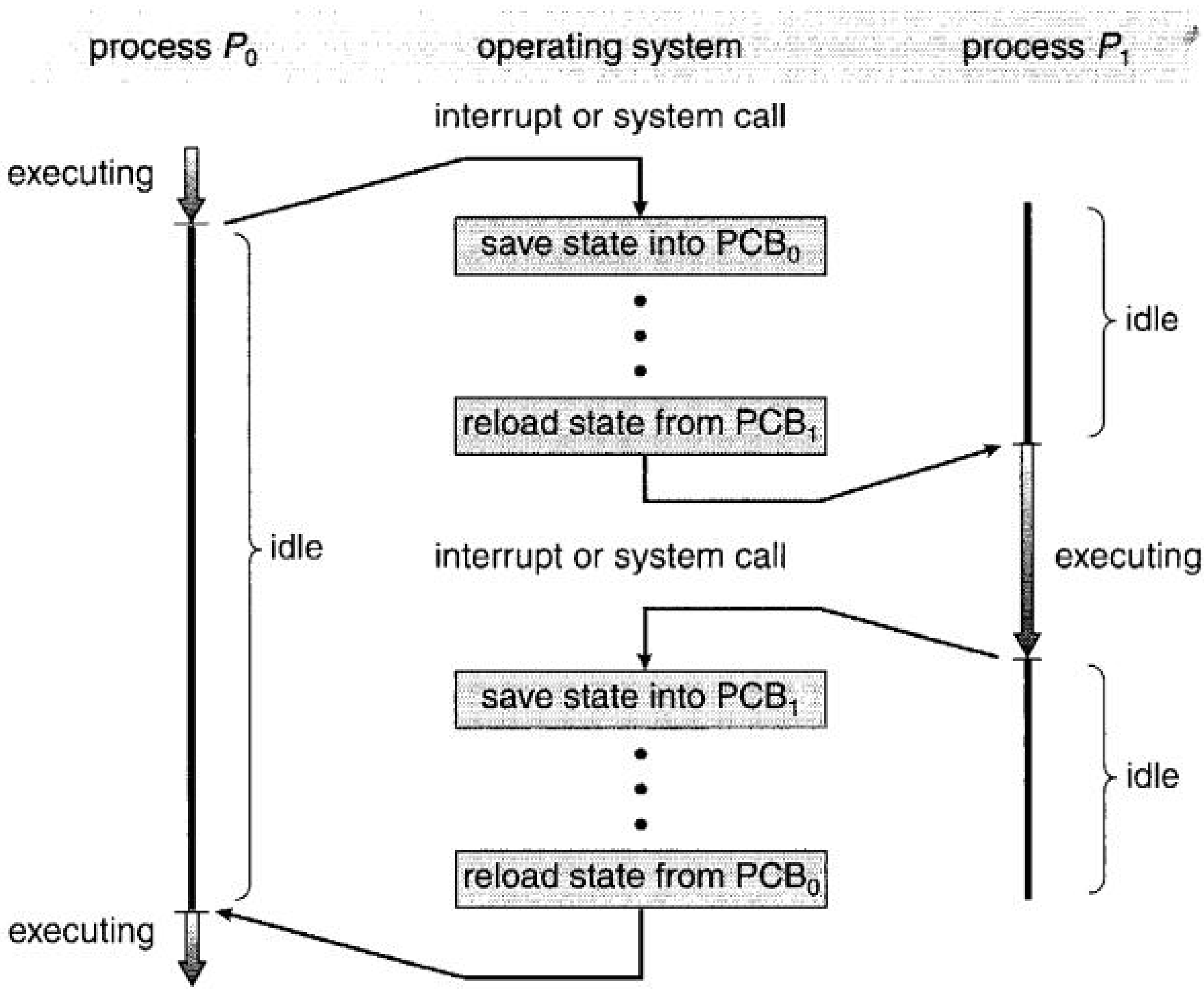

Figure 3.4 Diagram showing CPU switch from process to process.

Process Scheduling

- The objective of **multiprogramming** is to have some process running at all times, to **maximize CPU utilization**.
- The objective of **time sharing** is to **switch** the CPU among processes so **frequently** that users can **interact** with each program while it is running.
- the **process scheduler** selects an **available** process (possibly from a set of several available processes) for program execution on the CPU.

Scheduling Queues

- **job queue** which consists of **all processes** in the system.(As a job enter the system they are put int a job queue).
- **ready queue** which consists of processes that are **residing in main memory** and are **ready** and **waiting** to execute. (implement as **linked list**, a ready-queue header contains pointers to the first and final **PCBs** in the list)
- **device queue** The list of processes waiting for a particular I/O device (there are many processes in the system, the disk may be busy with the I/O request of some other process.). Each device has its **own device queue**.

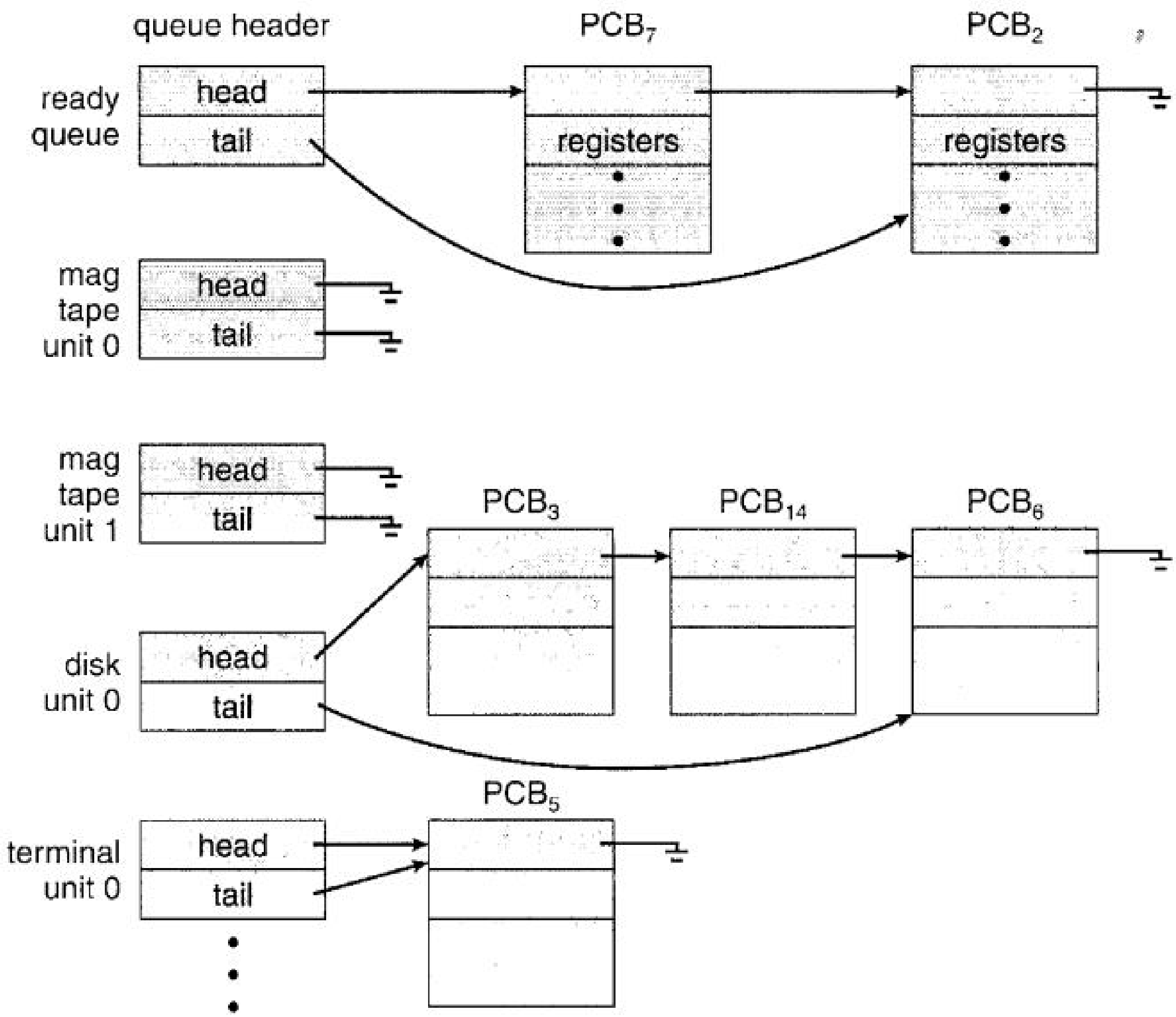

Figure 3.6 The ready queue and various I/O device queues.

During Process Running

1. The process could **issue an I/O request** and then be placed in an I/O queue. (I/O)
2. The process could create a new subprocess and **wait** for the subprocess's termination.(wait for some event to complete)
3. The process could be removed forcibly from the CPU, as a result of an **interrupt**, and be put back in the ready queue.

对于1，2来说进程从runing态转换到waiting态，等到事件（IO/其它事件）完成之后，从waiting 态转化为ready态
对于 3， 由 中断打断（interrupted)，从runing 态转化为 waiting态.

Process Representation in Linux

task struct:

```
pid_t pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

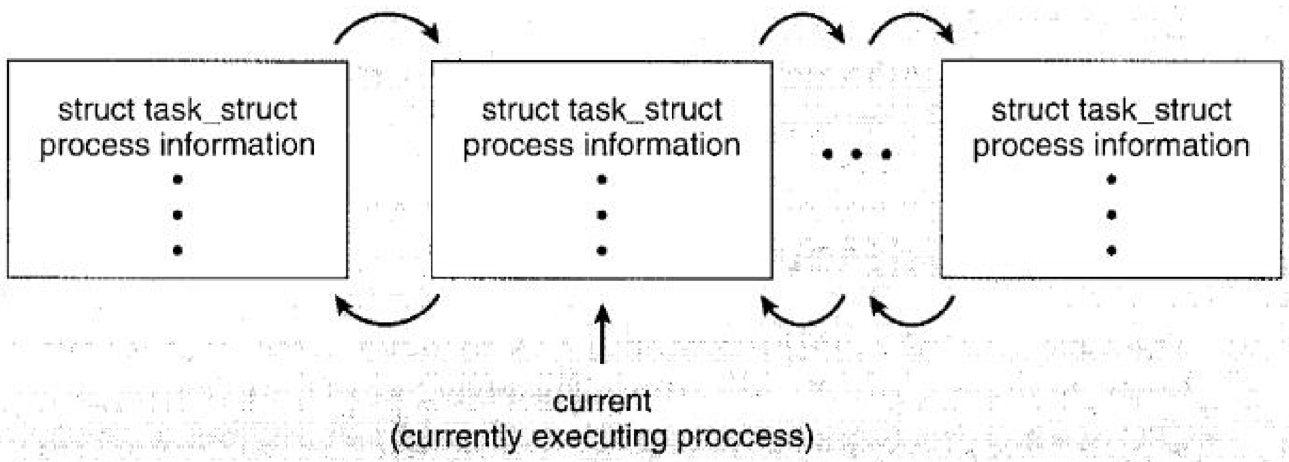

Figure 3.5 Active processes in Linux.

Schedulers

- **Long-term scheduler(job scheduler)** selects processes from job pool and loads them into memory for excution.(more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk) called job pool)
- **short-term scheduler(CPU scheduler)** selects from among the processes that are ready to execute and allocates the CPU to one of them.
- **medium-term schedule(swapping)** he key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove processes from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off.

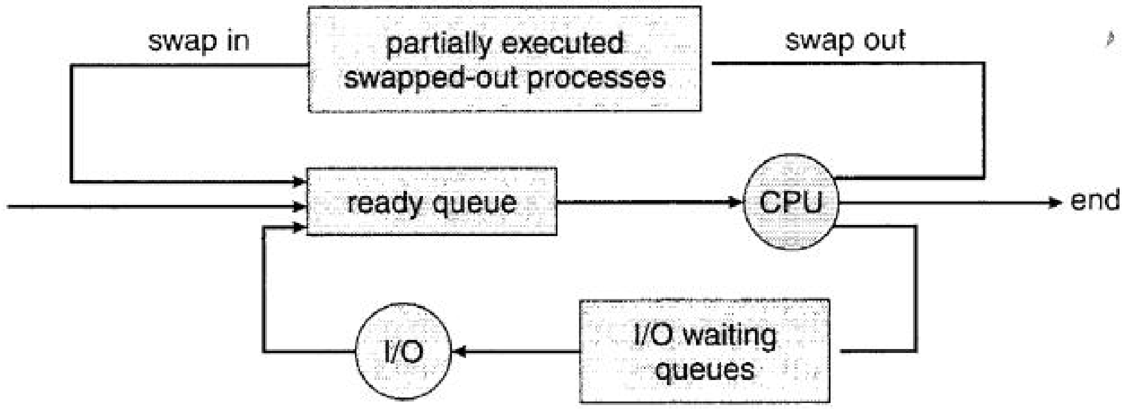

Figure 3.8 Addition of medium-term scheduling to the queueing diagram.

ps: 关于三种调度的个人理解.

- **长程调度** 这个调度主要是为了解决需要选择哪些进程读入内存（选择哪些进程被同时执行），因为计算机同时可以执行的进程数是有限的。通过长程调度我们可以控制计算机同时执行的进程数（多道的程度,degree of multiprogramming)，同时计算机的进程类型主要分为 I/O-bound process 和 CPU-bound process，在同时执行的进程中，一个好的 I/O进程和CPU进程的比例可以提升I/O设备和 CPU的效率（如果某一类进程过多，会导致另一种设备处于闲置状态）。长程调度可以很好地控制这两类进程的比例。长程调度主要出现在批处理系统中，很多现代操作系统不存在长程调度（比如linux)，长程调度的频率比较低。
- **中程调度** 这个调度也成为swaping，主要是选择将哪些进程中途移出主存（移入硬盘），和将哪些进程从硬盘移入主存，继续执行。中程调度和长程调度类似，但是本质上是不同的。两者的区别在于，长程调度是选择**哪些进程共同执行**（可以理解为被选择的进程原本还在进程池中没有被执行）而中程调度则是选择将一个执行到一半（正在执行）的进程，调出主存（或者反过来将进程调入主存继续执行），中程调度的目的是为了，降低多道（multiprograming)的程度，减少进程对CPU的争夺，释放主存资源。（中程调度可能是因为主存资源不足，需要释放主存资源），其频率比长程调度高。
- **短程调度** CPU调度，就是从 ready queue中选择某些进程来执行（选择将CPU资源分配给ready queue中的某个进程），CPU调度的频率很高（可能每100毫秒就要执行一次）,因此CPU调度的选择时间会很大程度影响CPU性能。