System device

1. File system

```
FileLock lock(long begin, long end, boolean shared);

release();// to release lock()
```

> 独锁和共享锁

```
public class LockingExample{
 public static final boolean EXCLUSIVE = false;
 public static final boolean SHARED = true;

 public static void main(String args[]) throws IOException{
    FileLock sharedLock = null;
    FileLock exclusiveLock = null;

    try{
        RandomAccessFile raf = new RandomAccessFile("file.txt","rw");

        // get the channel for the file
        FileChannel ch = raf.getChannel();

        // this locks 1/2 of the file - exclusive 独锁
        exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);

        /** Now modify the data ... */

        //release the lock
        exclusiveLock.release();

        //this locks the second half of the file - shared
        sharedLock = ch.lock(raf.length()/2 + 1,raf.length(),SHARED);

        //** Now read the data ... */

        //release the lock
        sharedLock.release();
    } catch {java.io.IOException ioe}{
        System.err.println(ioe);
    } finally {
        if (exclusiveLock != null)
            exclusiveLock.release();
        if(sharedLock != null)
            sharedLock.release();

    }
 }
}
```

2. process

running program

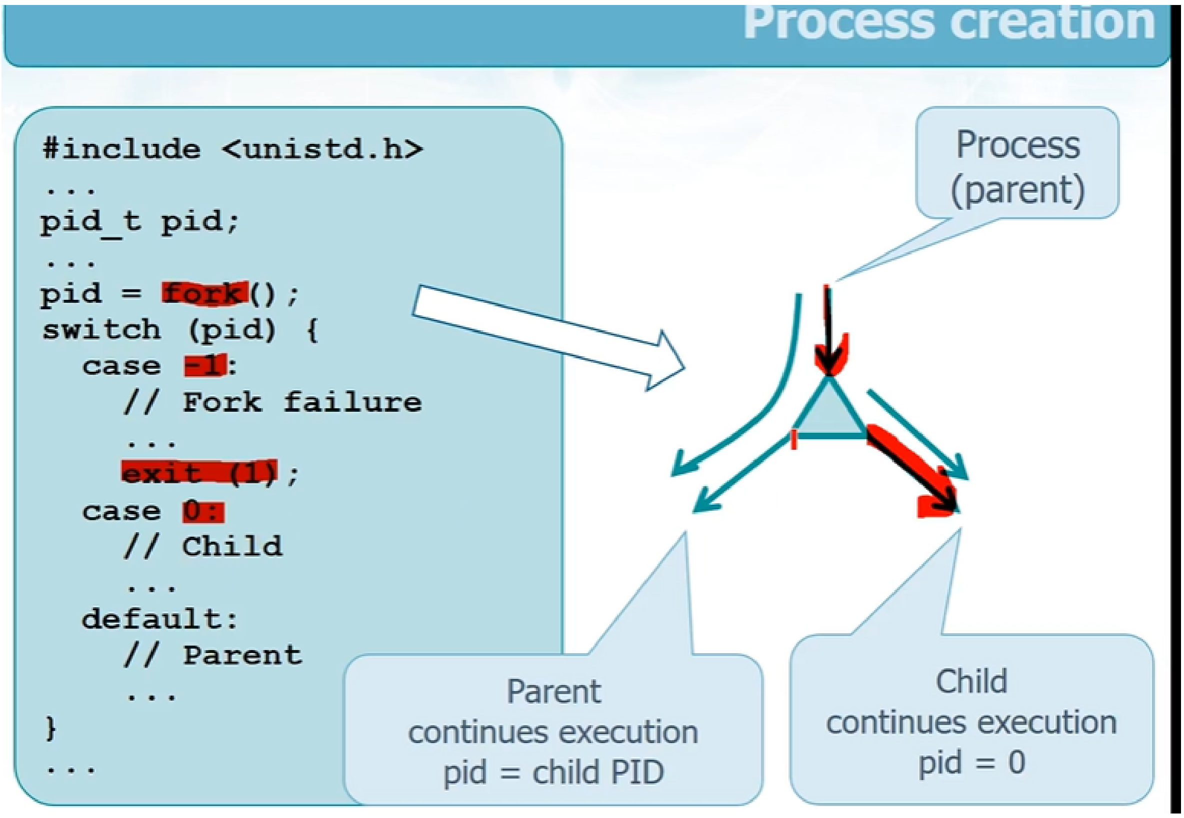
unique identifier -- PID(non negative integer)

```
getpid();
getppid();//get parent pid
```

2.1 process create

fork : create a new child process,child is a copy of the parent excluding the PID

```
pid_t fork (void);
//fork() 后的进程数 = 2^n 个 第n层 child process 个数
//fork()后返回两次，子进程返回值为0，父进程返回子进程ID
```



父进程与子进程的关系：数据空间相互独立，程序计数器值相同，共享代码空间。但不是全部copy。

	shared	exclusive
parent	1.source code(c) 2.open file descriptors 3.UID, GID 4.the root and the working directory 5.system resource and their utilization limits 使用限制 6.signal table	1.return ChildID 2.Data,heap,and stack space (inital value of var is inherited, but the space are separated)
child		1.return value 0

2.2 process termination

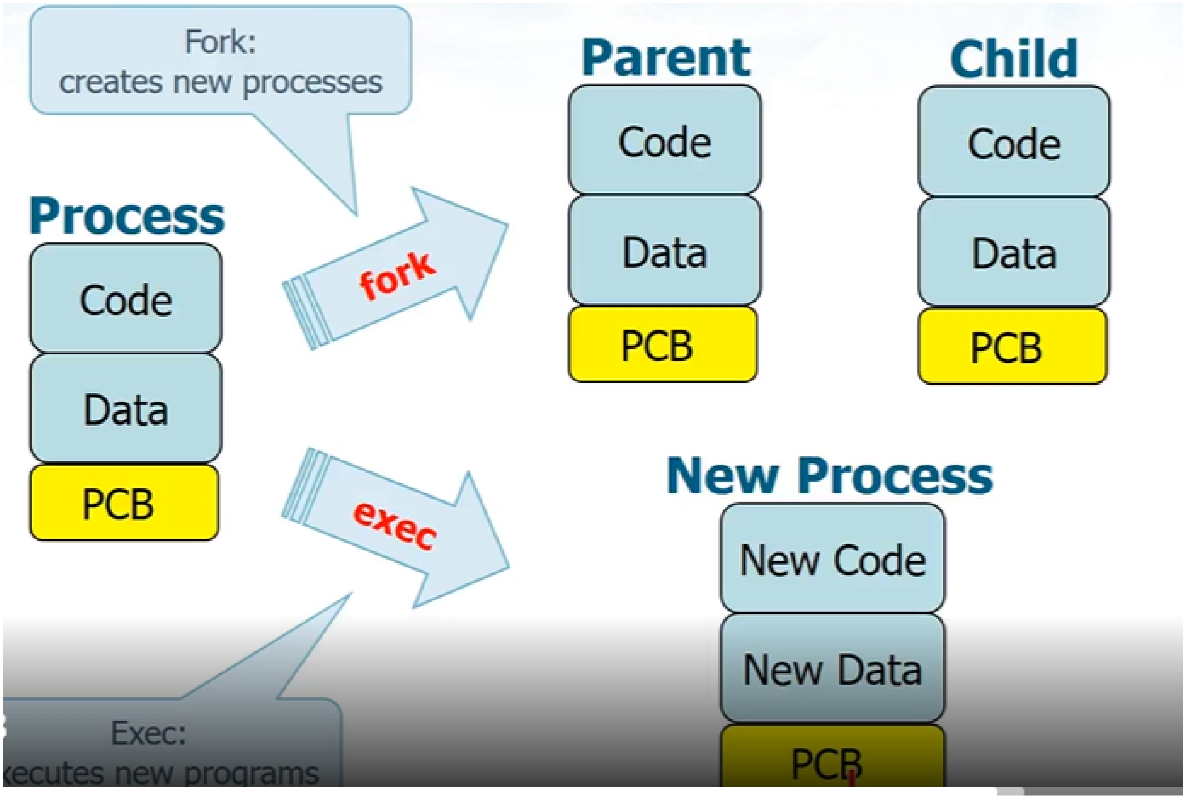
- return
- a exit system call

3 not normal methods

- call of the function abort
- termination signal, or a signal not caught is received
- last thread of process is cancelled

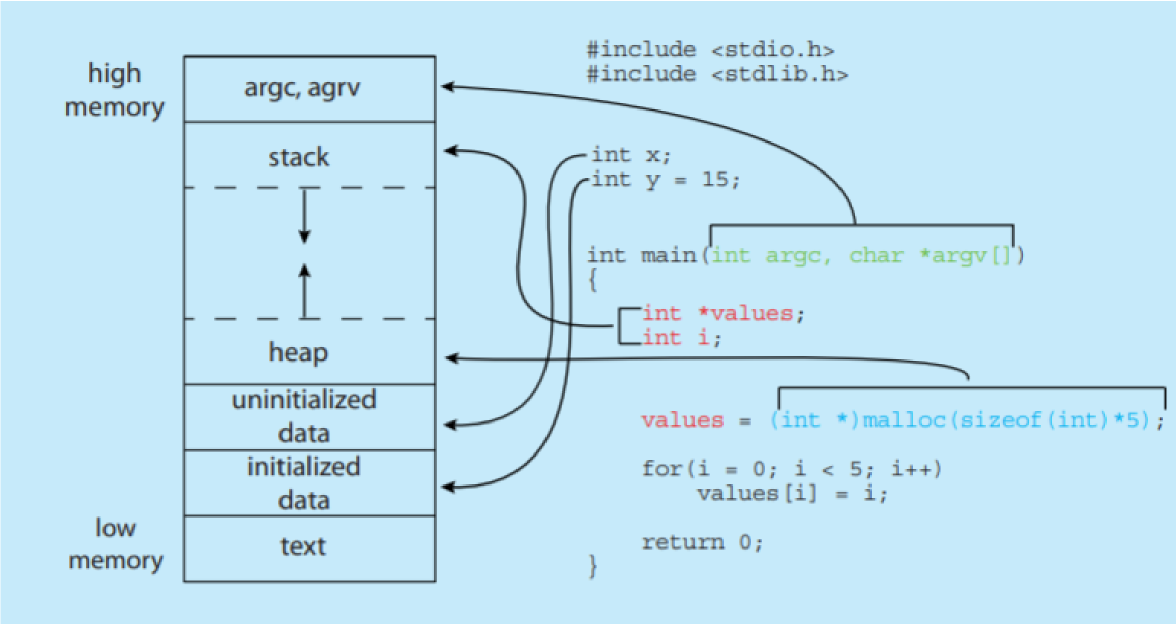
2.4 exec

exec create a new process



重点：画the process generation tree .

2.5 memory



2.6 process state

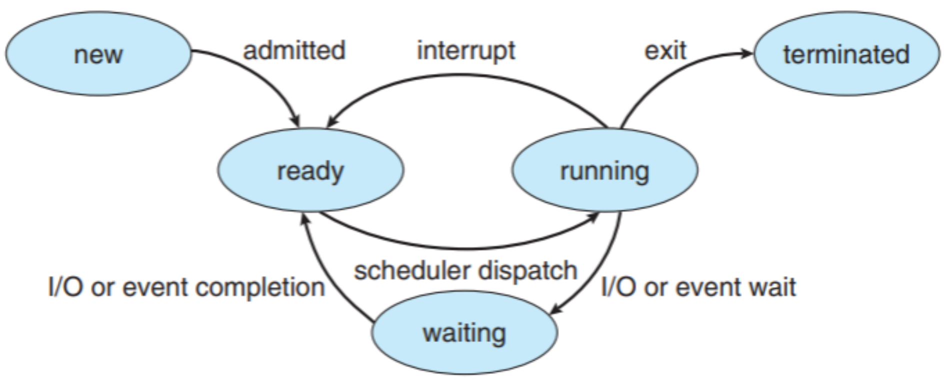


Figure 3.2 Diagram of process state.

PCB(process control block)

- process state
- process counter
- CPU registers
- CPU-scheduling information: priority,pointer to scheduling queue
- memory-management information
- I/O status information
- accounting information

2.7 CPU scheduling

2.7.1 scheduling algorithm

- First come first served scheduling(短任务优先可以减小等待时间)

decrease the waiting time,but the long running CPU bound starve

The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3. Thus, the average waiting time is $(0 + 24 + 27)/3 = 17$ milliseconds. If the processes arrive in the order P_2, P_3, P_1, however, the results will be as shown in the following Gantt chart:

The average waiting time is now $(6 + 0 + 3)/3 = 3$ milliseconds. This reduction

- Round- Robin scheduling

  time slice(10-100milliseconds in length) 合适大小，不然会引起频繁的上下文切换

  circle queue (new process are added to the tail of the ready queue)

  大于1 time quantum 的process ， timer 关闭，并且产生中断剩余未完成部分放在the tail of ready queue.

- priority scheduling

  preemptive: if priority higher than running process then preempt

  nonpreemptive: add the head of ready queue

  问题：indefinit blocking or starvation

  增加aging，increase the priority of a waiting process by 1.

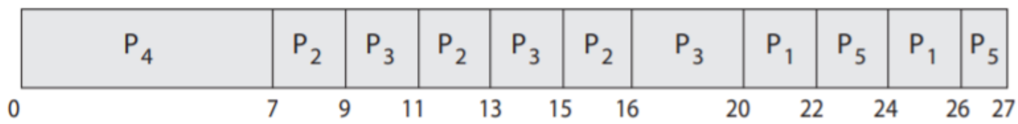
  解决办法：先执行优先级高的，对于优先级相同的结合RR scheduling的time slice。

priority 0 process.

Another option is to combine round-robin and priority scheduling in such a way that the system executes the highest-priority process and runs processes with the same priority using round-robin scheduling. Let's illustrate with an example using the following set of processes, with the burst time in milliseconds:

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

Using priority scheduling with round-robin for processes with equal priority, we would schedule these processes according to the following Gantt chart using a time quantum of 2 milliseconds:



- multilevel feedback queue scheduling

use past behavior to predict the future and assign job priority(SJF)

CPU bursts ,if a procee uses too much cpu time, it will be moved to lower-priority queue(aging to prevent starvation)

when a new process arrive ,it will preempt if it higher than current process

An entering process is put in queue 0. A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2. Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty. To prevent starvation, a process that waits too long in a lower-priority queue may gradually be moved to a higher-priority queue.

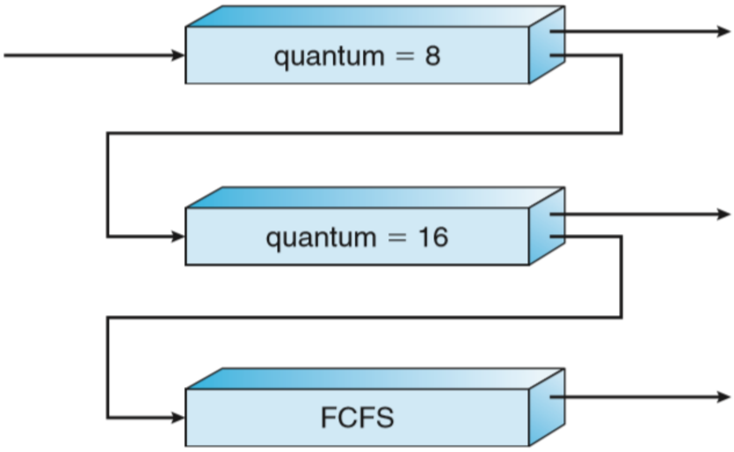


Figure 5.9 Multilevel feedback queues.

2.7.2 preemptive and nonpreemptive scheduling(抢占式和非抢占式调度)

nonpreemtive scheduling: keep the cpu until it releases it either by terminating or by switching to the waiting state

preemptive scheduling: modern operating systems use

-->导致问题：two process share data, 数据不一致问题

3. unix signals

interrupt

reaction:

signal(SIGname,SIG_DFL)

signal(SIGname,SIG_IGN)

signal(SIGname,signalHandleFunction)