姓名：刘远

学号：200111223

进程的状态：

一、三态模型：

三态模型：在多道程序系统中，进程在处理器上交替运行，状态也不断地发生变化。进程一般有3种基本状态：**运行、就绪和阻塞。**

（1）就绪：当一个进程获得了除处理机以外的一切所需资源，一旦得到处理机即可运行，则称此进程处于就绪状态。就绪进程可以按多个优先级来划分队列。例如，当一个进程由于时间片用完而进入就绪状态时，排入低优先级队列；当进程由I / O操作完成而进入就绪状态时，排入高优先级队列。

（2）运行：当一个进程在处理机上运行时，则称该进程处于运行状态。处于此状态的进程的数目小于等于处理器的数目，对于单处理机系统，处于运行状态的进程只有一个。在没有其他进程可以执行时（如所有进程都在阻塞状态），通常会自动执行系统的空闲进程。

（3）阻塞：也称为等待或睡眠状态，一个进程正在等待某一事件发生（例如请求I/O而等待I/O完成等）而暂时停止运行，这时即使把处理机分配给进程也无法运行，故称该进程处于阻塞状态。

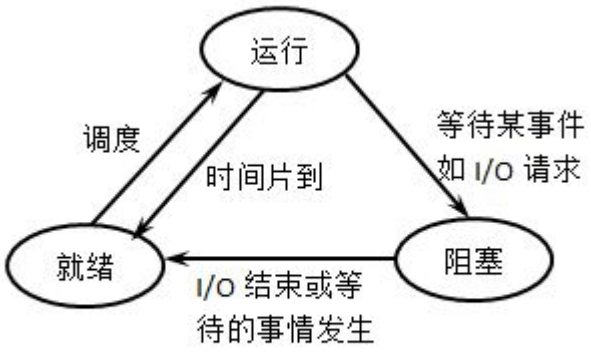

图 1. 进程的三态模型

二、五态模型

五态模型：对于一个实际的系统，进程的状态及其转换更为复杂。引入**新建态和终止态**构成了进程的五态模型。

新建态： 对应于进程刚刚被创建时没有被提交的状态，并等待系统完成创建进程的所有必要信息。 进程正在创建过程中，还不能运行。操作系统在创建状态要进行的工作包括分配和建立进程控制块表项、建立资源表格（如打开文件表）并分配资源、加载程序并建立地址空间表等。创建进程时分为两个阶段，第一个阶段为一个新进程创建必要的管理信息，第二个阶段让该进程进入就绪状态。由于有了新建态，操作系统往往可以根据系统的性能和主存容量的限制推迟新建态进程的提交。

终止态：进程已结束运行，回收除进程控制块之外的其他资源，并让其他进程从进程控制块中收集有关信息（如记帐和将退出代码传递给父进程）。类似的，进程的终止也可分为两个阶段，第一个阶段等待操作系统进行善后处理，第二个阶段释放主存。

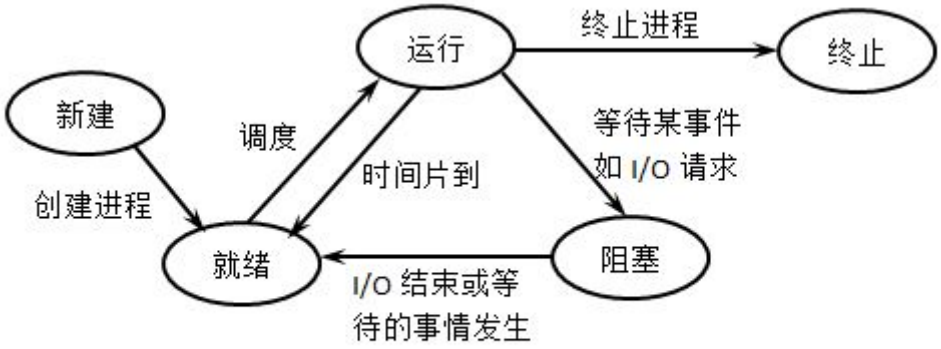

图 2. 进程的五态模型

由于进程的不断创建，系统资源特别是主存资源已不能满足所有进程运行的要求。这时，**就必须将某些进程挂起，放到磁盘对换区**，暂时不参加调度，以平衡系统负载；进程挂起的原因可能是系统故障，或者是用户调试程序，也可能是需要检查问题。

活跃就绪：是指进程在主存并且可被调度的状态。

静止就绪（挂起就绪）：是指进程被对换到辅存时的就绪状态，是不能被直接调度的状态，只有当主存中没有活跃就绪态进程，或者是挂起就绪态进程具有更高的优先级，系统将把挂起就绪态进程调回主存并转换为活跃就绪。

活跃阻塞：是指进程已在主存，一旦等待的事件产生便进入活跃就绪状态。

静止阻塞：是指进程对换到辅存时的阻塞状态，一旦等待的事件产生便进入静止就绪状态。

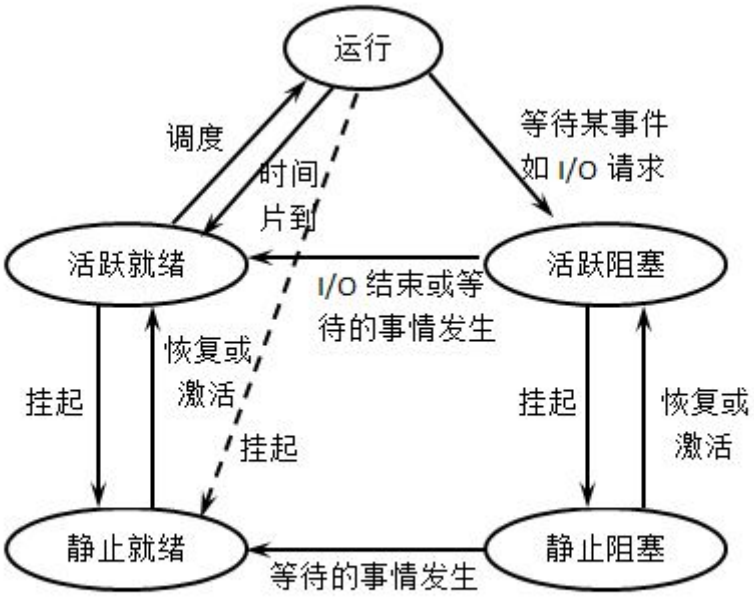

图 3. 细分进程状态及其转换

单进程程序是否可以把cpu利用率跑满?

对于单核cpu，单线程就可以把cpu跑满

对于多核cpu，单进程程序可以有多个线程，多个线程之间的并行可以把多核cpu的利用率跑满

进程控制函数

1、创建进程

fork函数：用户从已存在的进程(父进程)中创建一个新进程(子进程)。

fork创建的子进程是父进程的一个复制品，复制内容包括父进程的执行状态，即子进程将会从父进程当前执行的命令的下一条命令开始执行。

fork函数原型：

```
        pid_t fork(void)1.
```

返回值：父进程中返回子进程的进程号，一个大于0的整数，而子进程则返回0，出错返回-1.

因此，可以通过返回值判断该进程是父进程还是子进程。

fork出错可能有两种原因：1、当前的进程数已达上限 2、系统内存不足

2、exec函数族

exec函数族提供了一个在进程中启动另一个进程执行的方法，并且调用exec并不创建新进程。

exec函数原型：

```
        int execl(const char * path, const char * arg, ...)
        int execv(const char * path, char * const argv[])
        int execle(const char * path, const char * arg, ..., char * const
envp[])
        int execve(const char * path, char * const argv[], char * const envp[])
        int execlp(const char * file, const char * arg, ...)
        int execvp(const char * file, char * const argv[])1.2.3.4.5.6.
```

这些函数若调用成功，则加载新的进程从启动代码开始执行，不再返回，否则返回-1.(出错才有返回值)

exec + l(list,将新进程的每个命令行参数都作为一个参数传入，最后一个参数为NULL，起标记作用

+ v(vector,构造一个指向各参数的指针数组，将首地址传入，数组最后一个指针为NULL)

+ e(environment,传入新的环境变量表，其他exec函数仍使用当前的环境变量表执行新程序)

+ p(path，参数若含“/”则视为路径，否则视为不带路径的程序名)

例如：

```
            execlp("ps","ps","-ef",NULL);//第一个ps是程序名，第二个ps是第一个命令行参数
            perror("exec ps");//execlp执行出错时才会执行
            exit(1);1.2.3.
```

3、退出进程

一个C语言的程序总是从main()函数开始执行

main函数原型：

```
            int main(int argc, char * argv[])1.
```

argc: 命令行参数的数目
argv：指向参数的各个指针所构成的数组

进程正常退出的三种方式：由main()函数返回，调用exit()函数，调用*exit()*或Exit()函数
exit函数原型：

```
        void exit(int status)
        void _exit(int status)1.2.
```

_exit()：直接使进程停止运行，清除其使用的内存空间，并清除其在内核的各种数据结构
exit()：在执行退出之前加了若干道工序，检查文件的打开情况，把文件缓冲区中的内容写回文件（清理I/O缓冲）。
/若进程涉及对文件的操作，为了保证数据的完整性就一定要使用exit()函数/
例如：

```
        printf("output begin\n");//p1
        printf("content in buffer");//p2
        exit(0);//p1,p2都显示
        //_exit(0);//只显示p11.2.3.4.
```

//在一个进程调用exit之后，该进程并不会马上完全消失，留下一个称为僵尸（Zombie）的数据结构。

Zombie进程:
如果一个进程已经终止，但是它的父进程尚未调用wait或waitpid对它进行清理，这时的进程状态称为僵尸（Zombie)进程。任何进程在刚终止时都是僵尸进程。

4、wait和waitpid
如果一个父进程终止，而它的子进程还存在（仍在运行或成为僵尸进程），此时这些子进程的父进程将
改为init进程。只要有子进程终止，init就会调用wait函数清理它。

wait函数原型：

```
        pid_t wait(int * status)
        pid_t waitpid(pid_t pid, int * status, int options)1.2.
```

status: 为空——代表任意状态结束的子进程；不为空——指定状态结束的子进程。
pid: 设置等待进程，
pid>0: 等待进程ID等于pid的子进程，不管其它子进程的状态，直至指定的子进程结束才停止等待
pid=-1: 等待任意一个子进程退出
pid=0: 等待其组ID等于调用进程的组ID的任一子进程函数传入值
pid<-1: 等待其组ID等于pid的绝对值的任一子进程
options：
WNOHANG：若由pid指定的子进程不立即可用，则waitpid不阻塞，此时返回值为0
WUNTRACED：若实现某支持作业控制，则由pid指定的任一子进程状态已暂停，且其状态自暂停以

来

还未报告过，则返回其状态函数传入值。

返回值：调用成功——返回清理掉的子进程ID；没有子进程退出——返回0；出错——返回-1