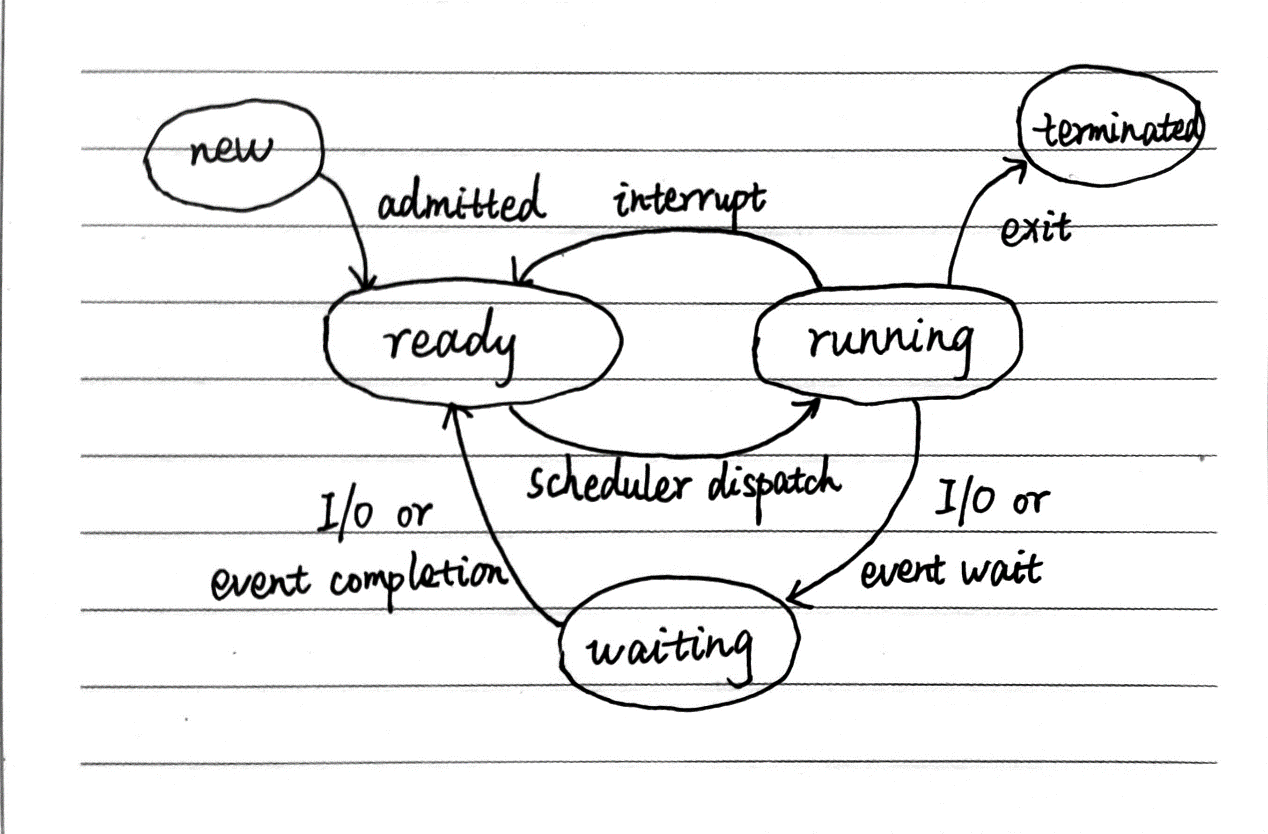
## 操作系统

## 第二次作业

Process

1. 画出进程的5状态转换图，并说明转换原因。



进程的5状态分别是创建状态（new），就绪状态（ready），运行状态（running），等待状态（waiting），终止状态（terminated）。

操作系统为创建一进程，构造了进程标识符并创建了管理进程所需的表格，但因资源有限还未允许执行该进程，此时进程处于创建状态。当该进程被允许执行，则转到就绪状态。就绪状态的进程根据调度/分派程序分派，一旦得到CPU就可以立即运行，即转到运行状态。在系统中，总只有一个进程处于运行状态，它拥有CPU的控制权。运行状态的进程遇中断时，转到就绪状态。运行状态的进程遇事件（如I/O操作），则进入等待状态。等待状态的进程即使给它CPU时间也无法执行，只能等待事件结束后转到就绪状态。运行中的进程终止时，进入终止状态，它不再具有执行资格。

2. Describe the differences among short-term, medium-term, and long-term scheduling.

Short-term scheduling selects which process should be executed next and allocates CPU. It’s invoked very frequently and must be fast.

Medium-term scheduling moves the blocked processes to external memory to reduce the degree of multi-programming. When these processes are ready to run, they are transferred from external storage into memory.

Long-term scheduling (job scheduler) selects which processes should be brought into the ready queue and controls the degree of multiprogramming. It’s invoked very infrequently and may be slow.

短程调度（CPU调度）选择可被下一个执行并分配CPU的进程，切换频率高、速度快；

中程调度将内存中处于阻塞状态的进程换至外存上（挂起），降低多道程序的度。当这些进程重新具备运行条件时，再从外存上调入内存。

长程调度（作业调度）选择可以进入就绪队列的进程，并控制多道程序的“道”，速度较慢、频率不高；

3. Describe the actions taken by a kernel to context-switch between processes.

To context-switch between processes, the kernel must save the state of the old process and load the saved state for the new process. System uses PCB to save or load the state of the process.

当系统在进程间进行上下文切换时，内核必须保存旧进程状态并为新进程加载之前保留的状态。系统用PCB进程控制块来记录进程的特征和状态。

4. 采用下述程序，确定A、B、C、D四行中pid和pid1的值。（假设父进程和子进程的pid分别为2600和2603）

#include

#include

#include

int main()

{

pid\_t pid,pid1;

pid=fork();

if (pid<0)

{

fprintf(stderr,"fork fail");

return 1;

}

else if (pid==0)

{

pid1=getpid();

printf("child:pid=%d",pid); //A

printf("child:pid1=%d",pid1); //B

}

else

{

pid1=getpid();

printf("parent:pid=%d",pid); //C

printf("parent:pid1=%d",pid1); //D

wait(NULL);

}

return 0;

}

若程序运行到A、B语句处，说明pid=fork()的返回值为0，程序在子进程中，则pid1=getpid()的返回值为子进程号2603；

若程序运行到A、B语句处，说明pid=fork()的返回值大于0，程序在父进程中，且pid=fork()的返回值为子进程号2603，pid1=getpid()的返回值为父进程号2600；

即A:child:pid=0 B:child:pid1=2603 C:parent:pid=2603 D:parent:pid1=2600

5. 课本P104页的3.10题：使用以下程序，请解释一下行X和Y的输出是什么。

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

#define SIZE 5

int nums[SIZE]={0,1,2,3,4};

int main(){

int i;

pid\_t pid;

pid=fork();

if(pid==0){

for(int i=0;i<SIZE;i++){

nums[i]\*=-i;

printf(“CHILD : %d ”,nums[i]); /\* LINE X \*/

}

}

else if(pid>0){

wait(NULL);

for(int i=0;i<SIZE;i++){

printf(“PARENT : %d ”,nums[i]); /\* LINE Y \*/

}

}

return 0;

}

若程序运行到行X处，说明pid=fork()的返回值为0，程序在子进程中。程序在for循环中执行5次，输出CHILD:0 CHILD:-1 CHILD:-4 CHILD:-9 CHILD:-16；

若程序运行到行Y处，说明pid=fork()的返回值大于0，程序在父进程中。父进程将首先等待子进程运行结束，但子进程修改全局变量并不影响父进程中变量的值。因此，程序在for循环中执行5次，分别输出PARENT:0 PARENT:1 PARENT:2 PARENT:3 PARENT:4；