苏州大学实验报告

院、系	计算机学院	年级专业	20 计科	姓名	柯骅	学号	2027405033
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无		实验日期	2023.03.02	

实验名称 实验 1 进程控制与进程调度

一、 实验目的

1. 加深对进程概念的理解，进一步认识并发执行的实质。
2. 掌握 Linux 操作系统中进程的创建和终止操作。
3. 掌握在 Linux 操作系统中创建子进程并加载新映像的操作。
4. 加深对进程概念的理解，明确进程和程序的区别
5. 深入理解系统如何组织进程
6. 理解常用进程调度算法的具体实现

二、 实验内容

1. （实验 3.1 进程的创建）

(1) 编写一个 C 程序，并使用系统调用 fork()创建一个子进程。要求如下：

a. 在子进程中分别输出当前进程为子进程的提示、当前进程的 PID 和父进程的 PID、根据用户输入确定当前进程的返回值、退出提示等信息。

b. 在父进程中分别输出当前进程为父进程的提示、当前进程的 PID 和子进程的 PID、等待子进程退出后获得的返回值、退出提示等信息。

(2) 编写另一个 C 程序，使用系统调用 fork()以创建一个子进程，并使用这个子进程调用 exec 函数族以执行系统命令 ls。

2. （实验 3.2 进程调度算法的模拟）

编写 C 程序模拟实现单处理机系统中的进程调度算法，实现对多个进程的调度模拟，要求采用常见进程调度算法（如先来先服务、时间片轮转和优先级调度等算法）进行模拟调度。

三、 实验步骤

1. （实验 3.1 进程的创建）

(1) fork 创建进程实验

本实验主要目的是学会在 Linux 下使用 fork()创建进程，并验证 fork()的返回值。首先在主程序中通过 fork()创建子进程，并根据 fork()的返回值确定所处的进程是子进程还是父进程，然后分别在子进程和当前进程（父进程）中调用 getpid()、getppid()、wait()等函数以完成实验内容。

使用 fork()创建子进程

```
1. pid_t childpid=fork();
```

在子进程中，使用 getpid()获取当前进程的 pid，使用 getppid()获取父进程的 pid，获取用户输入的返回值后结束进程

```
1. if(childpid==0)//fork()的返回值为 0，表示当前在子进程中
2. {
3.     printf("CHILD : 我是子进程\n");
4.     printf("CHILD : 子进程的 pid 为: %d\n", getpid());    //输出当前进程 pid
```

教务处制

```
    printf("CHILD : 我的父进程的 pid 为: %d\n", getppid()); //输出当前进程的父进程的 pid
    printf("CHILD : fork()的返回值是: %d\n",childpid); //输出 fork()的返回值
    printf("CHILD : 睡眠一秒\n");
    sleep(1);   //让进程睡眠一秒
    printf("CHILD : 输入一个返回值(0~255): ");
    int retval;
    scanf("%d",&retval); //输入子进程完成后的返回值
    printf("CHILD : 再见! \n");
    exit(retval);  //子进程退出，返回值为用户给定的返回值
}
```

在父进程中，同样使用 getpid()获取当前进程的 pid，并输出子进程的 pid 和返回值

```
else if(childpid>0)  //fork()的返回值是一个新的 pid，表示当前在父进程中
{
    printf("PARENT: 我是父进程\n");
    printf("PARENT: 父进程的 pid 为: %d\n",getpid()); //输出当前进程的 pid
    printf("PARENT: 我的子进程的 pid 为: %d \n",childpid);  //输出当前进程的子进程的 pid
    printf("PARENT: 我现在会等待我的子进程结束\n");
    int status;
    wait(&status); //等待子进程结束运行，并保存其状态
    printf("PARENT: 子进程的返回值为: %d\n",WEXITSTATUS(status)); //输出子进程的返回值
     printf("PARENT: 再见! \n");
     exit(0); //父进程退出
     }
```

异常处理，fork()的返回值为-1，表示 fork()失败

```
else  //fork()返回-1，这表示进程创建失败
{
    perror("fork 出错");  //显示错误信息
    exit(0);
}
```

(2) exec 族函数执行 ls 指令

实验内容（2）的主要目的是学会在 Linux 下使用 fork()创建进程、并使用 exec 族函数来加载新进程的映像。同时，也可以试验 wait()函数的作用。

exec 族函数的定义：

exec 函数族提供了一个在进程中启动另一个程序执行的方法。它可以根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段，在执行完之后，原调用进程的内容除了进程号外，其他全部被新的进程替换了。另外，这里的可执行文件既可以是二进制文件，也可以是 Linux 下任何可执行的脚本文件。

exec 族函数的作用：

我们用 fork 函数创建新进程后，经常会在新进程中调用 exec 函数去执行另外一个程序。当进程调用 exec 函数时，该进程被完全替换为新程序。因为调用 exec 函数并不创建新进程，所以前后进程的 ID 并没有改变。

exec 函数族有这几个：execl，execlp，execle，execv，execvp，execvpe

函数原型：

```
#include <unistd.h>
extern char **environ;

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

exec 函数族的函数执行成功后不会返回，调用失败时，会设置 errno 并返回-1，然后从原程序的调用点接着往下执行。

参数说明

path：可执行文件的路径名字

arg：可执行程序所带的参数，第一个参数为可执行文件名字，没带路径且 arg 必须以 NULL 结束

file：如果参数 file 中包含/，则就将其视为路径名，否则就按 PATH 环境变量，在它所指定的各目录中搜寻可执行文件。

在本次实验中使用的为 execvp，应先构造一个指向各参数的指针数组，然后将该数组的地址作为这些函数的参数，关键代码如下：

```
char *argv[] = {"ls","-l",NULL};
if(execvp("ls",argv)==-1)
{
  perror("exec 出错!");
  exit(-1);
}
else exit(0);
```

2. **（实验 3.2 进程调度算法的模拟）**

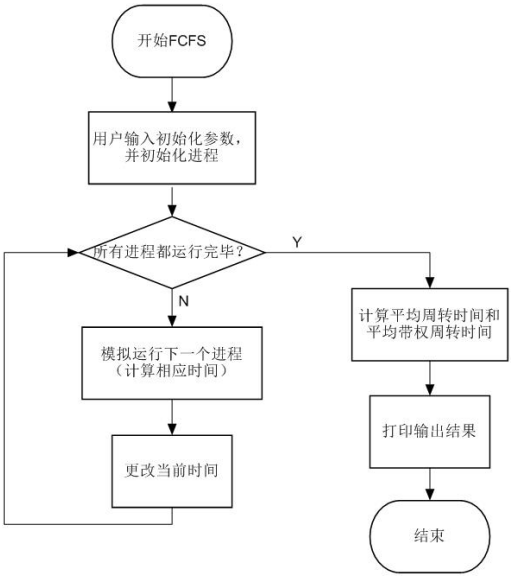
定义 PCB

对于 PCB 来说，我们需要的参数有：进程号，优先数，时间片长度，cpu 已经运行的时间，还需要运行的时间，已经运行完成的轮数和进程状态。PCB 是以链表形式进行存储的，所以在定义 PCB 时还需要加上一个指向 PCB 的指针。

```
typedef struct node{
  char name[10],state;
  int prio,round,cputime,needtime,count;
  struct node *next;
}PCB;
```

先来先服务（FCFS）

系统将按照作业到达的先后次序来进行调度，优先从后备队列中，选择一个或多个位于队列头部的作业，把他们调入内存，分配所需资源、创建进程，然后放入“就绪队列”,直到该进程运行到完成或发生某事件堵塞后，进程调度程序才将处理机分配给其他进程。



在程序中，我们将所有进程按照读入的顺序依次放进 ready 链表，然后不断地将 ready 链表的进程依次放入 run 链表执行，同时将状态参数做出相应的改变，直到所有的进程都被执行完成。

```
1.  void FCFSrun(char alg)  //先来先服务调度算法
2.  {
3.    PCB *p;
4.    while(run!=NULL)
5.    {
6.      run->cputime += run->needtime;
7.      run->needtime =0;
8.      run->next = finish;
9.      finish = run;
10.     run->state = 'F';
11.     run = NULL;
12.     if(ready!=NULL)
13.       firstin();
14.       prt(alg);  }  }
```

时间片轮转（RR）

在早期的时间片轮转法中，系统将所有的就绪进程按先来先服务的原则，排成一个队列，每次调度时，把 CPU 分配给队首进程，并令其执行一个时间片。时间片的大小从几 ms 到几百 ms。当执行的时间片用完时，由一个计时器发出时钟中断请求，调度程序便据此信号来停止该进程的执行，并将它送往就绪队列的末尾;然后，再把处理机分配给就绪队列中新的队首进程，同时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程，在一给定的时间内，均能获得一时间片的处理机执行时间。

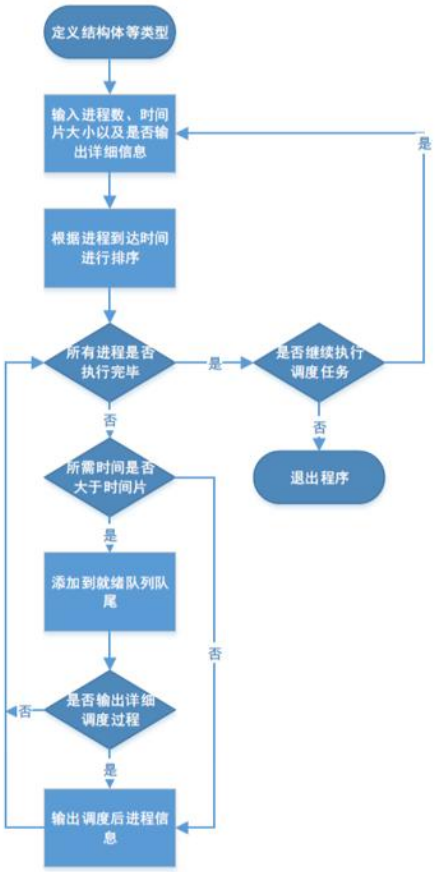
如果在时间片结束时进程还在运行，则 CPU 将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则 CPU 当即进行切换。调度程序所要做的就是维护一张就绪进程列表，当进程用完它的时间片后，它被移到队列的末尾。

在程序中，我们依次对读取到的进程进行 cpu 处理，每个进程都可以运行输入的指定时间片长度的时间。

如果发现当前进程剩余的所需时间小于等于时间片长度，那么表示这个进程可以在当前时间片中运行完，只需要让此进程**运行剩余所需的时间**即可，并不需要完整运行一个时间片，待其运行完后，将其放入 finish 队列中。

如果进程剩余的所需时间大于时间片长度，那么就让此进程运行完整的时间片长度，相应参数做出更改后等待下一次轮转。

代码详见 `roundrun(char alg)`函数



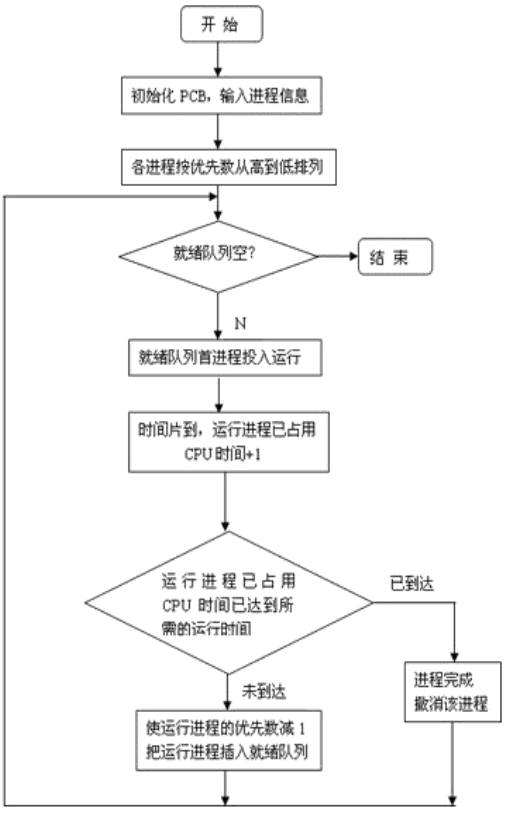
优先级调度算法（PR）

优先级进程调度算法，是把处理机分配给就绪队列中优先级最高的进程。这时，又可进一步把该算法分成抢占式和非抢占式，静态优先级和动态优先级。本次实验进行的是抢占式动态优先级调度算法。

在程序中，先根据进程的总运行时间从小到大排序，然后每个进程的优先数被初始化为 50-排序后的序号，然后循环进行以下过程，直到所有进程全部运行完

a. 找出优先数最大的进程放到队首并执行
b. 被执行的进程优先数-3，cpu 运行时间+1，需要时间-1
c. 如果需要时间为 0，表示已经运行完成，移出队列，状态修改为 F，表示结束

代码详见 `priority(char alg)`函数



四、 实验结果

1. （实验 3.1 进程的创建）

(1) fork 创建进程实验

```
PARENT: 我是父进程
PARENT: 父进程的pid为: 7651
PARENT: 我的子进程的pid为: 7652
PARENT: 我现在会等待我的子进程结束
CHILD : 我是子进程
CHILD : 子进程的pid为: 7652
CHILD : 我的父进程的pid为: 7651
CHILD : fork()的返回值是: 0
CHILD : 睡眠一秒
CHILD : 输入一个返回值(0~255): 66
CHILD : 再见!
PARENT: 子进程的返回值为: 66
PARENT: 再见!
```

(2) exec 族函数执行 ls 指令

```
PARENT: 当前处于父进程中
PARENT: 我现在会等待我的子进程结束
CHILD : 当前处于子进程中
总用量 60
-rwxrwxr-x 1 kh kh  8560 3月   8 18:01 1_exec
-rw-rw-r-- 1 kh kh  1105 3月   8 18:01 1_exec.c
-rwxrwxr-x 1 kh kh 12792 3月   8 17:57 1_pid
-rw-rw-r-- 1 kh kh  1735 3月   8 01:43 1_pid.c
-rwxrwxr-x 1 kh kh 13160 3月   8 18:00 2
-rw-rw-r-- 1 kh kh  6576 3月   8 18:00 2.c
PARENT: 子进程的返回值为: 0
PARENT: 再见!
```

2. （实验 3.2 进程调度算法的模拟）

(1) 先来先服务（FCFS）

```
输入进程数:
3
输入进程号和运行时间:
1 5
2 4
3 3
   先来先服务算法输出信息:
****************************************
进程号    所需时间    状态
3          3          W
2          4          W
1          5          W
进程号    所需时间    状态
2          4          R
1          5          W
3          0          F

进程号    所需时间    状态
1          5          R
2          0          F
3          0          F

进程号    所需时间    状态
1          0          F
2          0          F
3          0          F
```

(2) 时间片轮转（RR）

```
输入进程数:
3
请输入时间片: 2
输入进程号和运行时间:
1 2
2 3
3 3
     时间片轮转算法输出信息:
****************************************
进程号   cpu时间    所需时间    记数    时间片    状态
1          0          2          0          2          W
2          0          3          0          2          W
3          0          3          0          2          W
进程号   cpu时间    所需时间    记数    时间片    状态
2          0          3          0          2          R
3          0          3          0          2          W
1          2          0          1          2          F

进程号   cpu时间    所需时间    记数    时间片    状态
3          0          3          0          2          R
2          2          1          1          2          W
1          2          0          1          2          F

进程号   cpu时间    所需时间    记数    时间片    状态
2          2          1          1          2          R
3          2          1          1          2          W
1          2          0          1          2          F

进程号   cpu时间    所需时间    记数    时间片    状态
3          2          1          1          2          R
2          3          0          2          2          F
1          2          0          1          2          F

进程号   cpu时间    所需时间    记数    时间片    状态
3          3          0          2          2          F
2          3          0          2          2          F
1          2          0          1          2          F
```

(3) 优先级调度（PR）

```
输入进程数:
3
输入进程号和运行时间:
4 3
5 2
6 1
   优先数算法输出信息:
****************************************
进程号   cpu时间    所需时间    优先数    状态
6          0          1          49         W
5          0          2          48         W
4          0          3          47         W
进程号   cpu时间    所需时间    优先数    状态
5          0          2          48         R
4          0          3          47         W
6          1          0          46         F

进程号   cpu时间    所需时间    优先数    状态
4          0          3          47         R
5          1          1          45         W
6          1          0          46         F

进程号   cpu时间    所需时间    优先数    状态
5          1          1          45         R
4          1          2          44         W
6          1          0          46         F

进程号   cpu时间    所需时间    优先数    状态
4          1          2          44         R
5          2          0          42         F
6          1          0          46         F

进程号   cpu时间    所需时间    优先数    状态
4          2          1          41         R
5          2          0          42         F
6          1          0          46         F

进程号   cpu时间    所需时间    优先数    状态
4          3          0          38         F
5          2          0          42         F
6          1          0          46         F
```

教务处制

五、 实验思考与总结

1. 实验 3.1 进程的创建

（1）总结调用 fork()函数后的三种返回情况。

fork()系统调用有 3 种返回情况：

返回值大于 0，表示当前进程是父进程，这个返回值为子进程的进程 ID 值；

等于 0，表示当前进程是子进程；

小于 0，表示进程创建失败，需要报错。

（2）总结 fork()和 wait()配合使用的情况，并尝试在父进程中取消 wait()函数，观察进程的运行情况。

当父进程中同时使用 fork()和 wait()/waitpid()函数时,父进程会处于阻塞状态等待子进程的运行结束。如果父进程中没有调用 wait()/waitpid()函数，则父进程和其创建的子进程属于并发进程，也就是父进程和子进程几乎是独立运行的。

2. 实验 3.2 进程调度算法的模拟

（1）书本示例调度程序中的调度模式是抢占式还是非抢占式？

非抢占式

（2）若要将书本里示例调度程序中的进程运行方式改为每运行一次就将优先数减 2，同时将运行时间加 1，其他条件不变，则该如何修改？

在 running()函数中，将(p->nice)--; 改为 p->nice = p->nice - 2;

（3）如何将书本中的调度算法改为固定优先数调度算法？

需要保持示例代码中进程的 p->nice 值保持不变，即在 running()函数中，删除(p->nice)--;

3. 实验总结

本次实验中，实验 1 主要使用了 fork()和 exec 族函数来创建进程，在实验 2 中编写并模拟了三种常见的调度算法：FCFS，RR，PR，掌握了 Linux 操作系统中进程的创建和终止操作，加深了对进程概念的理解，明确进程和程序的区别，对于操作系统的进程调度与组织有了更深的理解，同时理解了常用进程调度算法的具体实现。