山东大学 计算机科学与技术 学院

操作系统 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201900161140 | 姓名：张文浩 | | 班级： 19人工智能 |
| 实验题目：进程调度算法实验 | | | |
| 实验学时：2 | | 实验日期： 4.19 | |
| 实验目的：  加深对进程调度概念的理解，体验进程调度机制的功能，了解 Linux 系统中进程调度策略的使用方法。练习进程调度算法的编程和调试技术。 | | | |
| 硬件环境：  **自带笔记本win10** | | | |
| 软件环境：  vmware/ubuntu | | | |
| 实验步骤与内容：   1. 根据实验指导手册完成实例实验编程 2. 运行psched，指定3个子进程的优先数为10,5，-10，调度策略都是默认策略     根据运行的结果可以看出3个子进程在相同的调度策略下优先数小的仅呈现的到了执行   1. 再次运行psched，指定3个子进程的优先数为10,5,18，调度策略分别是：0,0,1，运行结果如下     可以看到4361进程虽然优先级最大，但由于其调度策略为先进先出，因此总是最先得到调度。   1. 独立实验   第一步：自定义子进程增加优先级、减小优先级、父进程增加优先级的三个函数    第二步：在主函数中注册三个终端信号，定义优先级结构，为下一步初始化优先级做准备    第三步：创建一个子进程，在子进程和父进程的调度策略都为0，优先级都为10。      第四步：在父子进程中都开启循环，每次循环中都输出各自的进程号、调度算法和优先级，并且在每次循环最后睡眠3秒，等待接收键盘指令。      第五步：创建makefile，执行程序，实现的功能是，键盘键入ctrl+c则子进程优先级+1，键入ctrl+z则子进程优先级-1，键入ctrl+\则父进程优先级-1.  测试结果如下      通过实验结果可以看出，一开始子进程的优先级比父进程大，所以每次输出的时候总是父进程先输出，子进程再输出。后面将子进程的优先级减到比父进程小之后，在循环输出的时候总是子进程先输出，父进程再输出。符合理论情况，即优先级小的进程先执行。 | | | |
| 结论分析与体会：  本次实验反映了操作系统的调度算法，包括先到先服务调度（FSFC）、最短作业调度（SJF）和轮转法调度（RR）。在是实例实验中，第一次将三个子进程的优先策略都设为0，即默认的分时调度策略，可以看到三个子进程每次都是按照优先级从小到大执行的。在第二次执行时，将第三个子进程的调度策略设定为1，即先来先服务策略。并设置该子进程的优先级最大，如果都在分时策略下执行，则该进程每次都会最后一个执行，但本次该进程调度策略为先来先服务，所以每次该进程都会先执行。该试验反映出的关于进程调度策略的知识有，在优先级调度算法中，优先级小的先被调用。因为SCHED\_OTHER 是分时调度策略，而SCHED\_FIFO是实时调度策略，实时调度策略将得到优先调用，所以在实例实验第二次执行时，被设置为SCHED\_FIFO调度策略的进程总是被先执行。当实时进程准备就绪后，如果当前cpu正在运行非实时进程，则实时进程立即抢占非实时进程 。  我在独立实验中设有两个并发执行的父子进程，不断循环输出各自进程号、优先数和调度策。进程初始调度策略均为系统默认策略和默认优先级。当子进程进程收到 SIGINT 信号时会自动将其优先数加 1，子进程收到 SIGTSTP 信号时会自动将其优先数减 1，父进程受到SIGQUIT信号会将其优先数减1。因为我只找到了这三个可以通过键盘输入的信号，所以无法实现父进程收到信号将其优先数加1的操作。这是我独立实验的不足，希望以后可用通过进一步的学习进行改进。  另外，在实验测试的时候我发现，偶尔会出现优先级大的优先级小的进程先输出的情况，经过助教的指点，我知道了出现这种情况原因是因为cpu在执行调度的时候是根据每个进程的nice来调度的，如果一个进程的优先级总是比别的进程的优先级大，每次都不被先执行，这个进程的nice会相应的减小，cpu这种机制能够适当的避免饥饿现象。 | | | |

附源代码

#include<stdio.h>

#include<stdlib.h>

#include<sched.h>

#include<sys/time.h>

#include<sys/resource.h>

#include<unistd.h>

#include<signal.h>

#include<wait.h>

int pid;

typedef void (\*sighandler)(int);

void plusc(){

if(pid>0) return;

setpriority(PRIO\_PROCESS,getpid(),getpriority(PRIO\_PROCESS,0)+1);

printf("process %d priority+1 = %d\n",getpid(),getpriority(PRIO\_PROCESS,0));

}

void minusc(){

if(pid>0) return;

setpriority(PRIO\_PROCESS,getpid(),getpriority(PRIO\_PROCESS,0)-1);

printf("process %d priority-1 = %d\n",getpid(),getpriority(PRIO\_PROCESS,0));

}

void minusp(){

if(pid==0) return;

setpriority(PRIO\_PROCESS,getpid(),getpriority(PRIO\_PROCESS,0)-1);

printf("process %d priority-1 = %d\n",getpid(),getpriority(PRIO\_PROCESS,0));

}

int main(int argc,char\* argv[])

{

signal(SIGINT,(sighandler)plusc);

signal(SIGTSTP,(sighandler)minusc);

signal(SIGQUIT,(sighandler)minusp);

struct sched\_param p[2];

for(int i=0;i<2;i++){

p[i].sched\_priority=10;

}

pid=fork();

if(pid>0){

sched\_setscheduler(getpid(),SCHED\_OTHER,&p[0]);

setpriority(PRIO\_PROCESS,getpid(),10);

sleep(1);

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

printf("parent pid = %d, priority = %d, scheduler = %d\n",getpid(),getpriority(PRIO\_PROCESS,0),sched\_getscheduler(getpid()));

sleep(3);

}

}else{

sched\_setscheduler(getpid(),SCHED\_OTHER,&p[1]);

setpriority(PRIO\_PROCESS,getpid(),10);

sleep(1);

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

printf("child pid = %d, priority = %d, scheduler = %d\n",getpid(),getpriority(PRIO\_PROCESS,0),sched\_getscheduler(getpid()));

sleep(3);

}

exit(EXIT\_SUCCESS);

}

return EXIT\_SUCCESS;

}