

**实 验（实训）报 告**

**项 目 名 称**  实验5 实验6

**所属课程名称**  操作系统

**项 目 类 型**  验证/设计型

**实验(实训)日期**  2024.10.16

**班 级**  22软工2班

**学 号**  220110900606

**姓 名**  贺烨强

**指导教师**  陈伟锋

浙江财经大学教务处制

|  |
| --- |
| **一、实验（实训）概述：** |
| **【目的及要求】**  **【基本原理】**  **【实施环境】** |
| **二、实验（实训）内容：** |
| **【实验（实训）过程】（步骤、记录、数据、程序等）**  **实验五**  **1.安装gcc**    2.完成hello.c  屏幕截图 2024-10-16 135156  3.完成fork.c  屏幕截图 2024-10-16 142248    **4.完成proceses.c**  屏幕截图 2024-10-16 144152    代码中创建了一个子进程，并且在父进程中也有一个无限循环。这意味着两个进程（父进程和子进程）都会无限地执行各自的循环，除非程序被外部中断或系统资源耗尽。  具体来说，在 fork() 调用后，如果返回值 pid 小于 0，则表示 fork() 失败；如果 pid 等于 0，则表示当前是在子进程中运行；如果 pid 大于 0，则表示当前是在父进程中运行，pid 的值为新创建的子进程的 PID。  **5.完成communication.c**        **子进程1：**  每次循环开始时关闭 filedis[0]，导致在第一次循环之后 filedis[0] 已经被关闭。  当接收到 SIGUSR1 信号时，再次关闭 filedis[0] 和 filedis[1]，但这不会影响已经关闭的 filedis[0]。  信号处理函数返回后，子进程1继续执行循环，但由于 filedis[0] 已经被关闭，这不会引起问题，但会导致子进程1无法正确退出。  **子进程2：**  每次循环开始时关闭 filedis[1]，导致在第一次循环之后 filedis[1] 已经被关闭。  当接收到 SIGUSR1 信号时，再次关闭 filedis[0] 和 filedis[1]，但这不会影响已经关闭的 filedis[1]。  信号处理函数返回后，子进程2继续执行循环，但由于 filedis[1] 已经被关闭，这不会引起问题，但会导致子进程2无法正确退出。  **父进程：**  父进程等待子进程1和子进程2结束后，关闭管道的读写端。  如果子进程1和子进程2没有正确退出，父进程会一直等待。  **实验六**  **#include <iostream>**  **#include <algorithm>**  **using namespace std;**  **enum State {ready,block,done};**  **struct PCB**  **{**  **int ID;**  **int priority;**  **int cputime;**  **int alltime;**  **int startblock;**  **int blocktime;**  **State state;**  **struct PCB \* next;**    **}pcb[5];**  **int main()**  **{**  **//定义五个进程的初始状态**  **pcb[0] = {0,9,0,3,2,3,ready};**  **pcb[1] = {1,38,0,3,1,1,ready};**  **pcb[2] = {2,30,0,6,4,2,ready};**  **pcb[3] = {3,29,0,3,1,1,ready};**  **pcb[4] = {4,0,0,4,1,2,ready};**  **int time=0;**  **Inttemp[5]; while(pcb[0].alltime||pcb[1].alltime||pcb[2].alltime||pcb[3].alltime||pcb[4].alltime)**  **{**    **for(int i=0; i<5; i++)**  **{**  **int array[]= {pcb[0].priority,pcb[1].priority,pcb[2].priority,pcb[3].priority,pcb[4].priority}; if((pcb[i].state==ready)&&(pcb[i].priority==\*max\_element(array,array+5)))**  **{**    **time++;**  **pcb[i].priority-=3;**  **pcb[i].cputime+=1;**  **pcb[i].alltime-=1;**  **for(int j=0; j<5; j++)**  **{**    **if(j==i) continue;**  **if(pcb[j].alltime>0&&pcb[j].state==ready)**  **{**  **pcb[j].priority+=1;**  **}**  **}**    **if(pcb[i].alltime==0)**  **{**  **pcb[i].state=done;**  **pcb[i].priority=-1; }**      **for(int j=0; j<5; j++) blocktime-1**  **{**    **if(pcb[j].state==block&&pcb[j].blocktime>0)**  **{**    **pcb[j].blocktime-=1;blocktime-1**  **if(pcb[j].blocktime==0)**  **{**  **pcb[j].state=ready;**  **pcb[j].priority=temp[j];**  **}**  **}**  **}**    **if(pcb[i].startblock>0)**  **{**  **pcb[i].startblock-=1;**  **if(pcb[i].startblock==0)**  **{**  **pcb[i].state=block;**  **temp[i]=pcb[i].priority;**  **pcb[i].priority=-1;**  **}**  **}**    **cout<<"时间片为: "<<time<<endl;**  **cout<<"执行进程: "<<pcb[i].ID<<endl;**  **void show\_PCB();**  **show\_PCB();**    **}**      **}**    **}**  **return 0;**  **}**  **void show\_PCB()**  **{**    **cout<<"就绪队列: ";**  **for(int j=0; j<5; j++)**  **{**  **if(pcb[j].state==ready)**  **{**  **cout<<"->"<<pcb[j].ID;**  **}**  **}**  **cout<<endl;**  **cout<<"阻塞队列: ";**  **for(int j=0; j<5; j++)**  **{**  **if(pcb[j].state==block)**  **{**  **cout<<"->"<<pcb[j].ID;**  **}**  **}**  **cout<<endl;**  **cout<<"ID\tpriority\tcputime\talltime\tstartblock\tblocktime\tstate"<<endl;**  **for(int j=0; j<5; j++)**  **{**  **cout<<pcb[j].ID<<'\t'<<pcb[j].priority<<'\t'<<'\t'<<pcb[j].cputime<<'\t'<<pcb[j].alltime<<'\t'<<pcb[j].startblock<<'\t'<<'\t'<<pcb[j].blocktime<<'\t'<<'\t'<<pcb[j].state<<endl;**  **}**  **cout<<"----------------------------------------------------------------------------"<<endl;**  **}**  **屏幕截图 2024-10-30 141500**  **屏幕截图 2024-10-30 141514**  **屏幕截图 2024-10-30 141523**  **屏幕截图 2024-10-30 141532**  **屏幕截图 2024-10-30 141540**  **屏幕截图 2024-10-30 141548**  1**.进程初始化**：创建了5个进程，每个进程有其独特的ID、优先级、已占用的CPU时间、需要占用的CPU时间、阻塞开始时间、阻塞时间以及状态。  2.**时间片轮转**：通过一个循环来模拟时间片的推进，每次循环代表一个时间单位。  3.**优先级调度**：在每个时间片内，选择具有最高优先级的就绪进程执行。正在执行的进程每执行一个时间片，其优先级降低；其他就绪进程的优先级增加。  4.**进程阻塞与恢复**：当一个进程达到特定条件时，它会被阻塞一段时间，在这段时间结束后，该进程会重新变为就绪状态。  5.**进程完成**：当一个进程所需的CPU时间全部用完后，该进程的状态被设置为完成。  6.**状态显示**：在每个时间片结束后，显示当前所有进程的状态，包括就绪队列和阻塞队列中的进程  **【结论与讨论】（结果、分析）** |
| **三、指导教师评语及成绩：** |
| **评语：**  **成绩： 指导教师签名：**  **批阅日期：** |