

**实 验（实训）报 告**

**项 目 名 称**  （实验名称）

**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**

**班 级**  22软件工程1班

**学 号**  220110900725

**姓 名**  谭鸿杰

**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】**  1.安装gcc  2.完成hello.c  3.完成fork.c  4.完成process.c  5.完成communication.c  375f26aa996acd9cb84c07a7561859d  **【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）**  1.安装gcc  sudo apt update  sudo apt install gcc  gcc --version    安装成功  sudo apt install build-essential  2.编写程序hello.c    3.编写程序fork.c    代码如上  运行结果如下    4.编写程序procese.c        代码如上  运行结果如下    5.编写程序communication.c  代码如下：  #include<stdio.h>  #include<stdlib.h>  #include<string.h>  #include<unistd.h>  #include<sys/types.h>  #include <signal.h>  #include<sys/wait.h>  void signHandler1(int isigNo);  void signHandler2(int isigNo);  int child1,child2;  int filedis[2];  int main()  {  char buffer[40];  char info[40];  int status;  int counter=1;  printf("Process Parent pid=%d\n",getpid());  if(pipe(filedis)<0){  printf("Create Pipe failed\n");  return -1;  }  signal(SIGINT,signHandler1);  child1=fork();  printf("child1=%d\n",child1);  if(child1==0){  printf("child1 pid=%d\n",getpid());  signal(SIGINT, SIG\_IGN);  signal(SIGUSR1,signHandler2);  while(1){  close(filedis[0]);  sprintf(info,"I send you %d times",counter);  write(filedis[1],info,30);  counter++;  printf("PID:%d I have sent.\n",getpid() );  sleep(1);  }  }  else if(child1>0){  child2=fork();  if(child2==0){  printf("child2 pid=%d\n",getpid());  signal(SIGINT,SIG\_IGN);  signal(SIGUSR1,signHandler2);  while(1){  close(filedis[1]);  read(filedis[0],buffer,40);  printf("PID:%d I have received:%s\n",getpid(),buffer);  sleep(1);  }  }  waitpid(child1,NULL,0);  printf("child1 Process1 is over\n");  waitpid(child2,NULL,0);  printf("child2 Process2 is over\n");  close(filedis[0]);  close(filedis[1]);  printf("Parent Process is killed\n");    }  return 0;  }  void signHandler1(int isigNo){  printf("\nParent received signal Ctrl+C\n");  if(isigNo==SIGINT){  kill(child1,SIGINT);  kill(child2,SIGINT);  }  }  void signHandler2(int isigNo){  close(filedis[0]);  close(filedis[1]);  if(child1==0 && isigNo==SIGUSR1){  printf("child1 Process is killed by parent\n");  exit(0);  }  if(child2==0 && isigNo==SIGUSR1){  printf("child2 Process is killed by parent\n");  exit(0);  }  }    进程调度：  代码如下：  #include <stdio.h>  #define N 10  #define P proc  struct process{  int id;  int py; /\*priority\*/  int ct; /\*cputime\*/  int at; /\*alltime\*/  int sb; /\*startblock\*/  int bt; /\*blocktime\*/  int st; /\*state:1 ready;-1 block;0 finish\*/  };  struct process proc[N];  int main(){  int i,j,n;  int max\_pt,priority;  int timeslice=0;  int ready\_q[N],block\_q[N];  char ch;    for(i=0;i<=N;i++){  ready\_q[i]=0; block\_q[i]=0;  }  scanf("%d",&n);  for(int k=1;k<=n;k++){  P[k].id=k-1;  }  for(int k=1;k<=n;k++){  scanf("%d%d%d%d%d",&P[k].py,&P[k].ct,&P[k].at,&P[k].sb,&P[k].bt);  }  printf("\n\n RUNNING RPOC:%d\n",-1);  printf(" READY\_QUEUE :");  for(i=1;i<=n;i++)  printf("->id%d",i-1);  printf("\n");  printf(" BLOCK\_QUEUE :\n ");  printf("===================================================== ===");  printf("\n %-12s","ID");  for(i=1;i<=n;i++)  printf("%8d",i-1);  printf("\n %-12s","PRIORITY");  for(i=1;i<=n;i++)  printf("%8d",P[i].py);  printf("\n %-12s","CPUTIME");  for(i=1;i<=n;i++)  printf("%8d",P[i].ct);  printf("\n %-12s","ALLTIME");  for(i=1;i<=n;i++)  printf("%8d",P[i].at);  printf("\n %-12s","STARTBLOCK");  for(i=1;i<=n;i++)  printf("%8d",P[i].sb);  printf("\n %-12s","BLOCKTIME");  for(i=1;i<=n;i++)  printf("%8d",P[i].bt);  printf("\n %-12s","STATE");  for(i=1;i<=n;i++){  P[i].st=1;  printf("%8s","READY");  }  for(i=1;i<=n;i++){  P[i].st=1;  ready\_q[i]=i;  }  ready\_q[0]=n;  /\* ready\_q[0]表示就绪队列中进程个数，block\_q[0]表示阻塞队列中进程个数\*/  /\* 进行运算\*/  do{  timeslice++; /\* 时间片加1 \*/  /\*  max\_pt记录优先级最高进程的pt,ready\_q数组记录就绪队列的顺序  priority记录优先级最高进程的ID  此程序段使ready\_q排序  \*/  for(i=1,max\_pt=-1;i<=ready\_q[0];i++)  if (P[ready\_q[i]].py>max\_pt){  max\_pt=P[ready\_q[i]].py;  j=i;  priority=ready\_q[i];  }  for(i=j;i<ready\_q[0];i++)  ready\_q[i]=ready\_q[i+1];  ready\_q[0]--;  /\* 按原则操作各进程，实质是按原则改变数组各值\*/  P[priority].py-=3;  if (P[priority].py<0) P[priority].py=0;  P[priority].ct++;  P[priority].at--;  /\* 除执行进程外，其他进程的操作\*/  for(i=1;i<=ready\_q[0];i++)  P[ready\_q[i]].py+=1;  /\* 对阻塞队列和就绪队列赋值\*/  /\*? 进程被阻塞的时间BLOCKTIME，表示已阻塞的进程再等待BLOCKTIME个时间片后，进程将转换成就绪状态；\*/  for(i=1;i<=block\_q[0];i++)  {  P[block\_q[i]].bt-=1; /\* 阻塞队列中的进程的blocktime－1 \*/  if (P[block\_q[i]].bt==0) /\* 如果该进程的blocktime＝0，说明该进程需要进入就绪队列\*/  { P[block\_q[i]].st=1; /\* 将该进程的st状态置1，将其从阻塞队列移入就绪队列\*/  block\_q[0]--;  ready\_q[0]++;  ready\_q[ready\_q[0]]=i;  }  }  if(P[priority].at==0) /\* 对当前执行进程，如果at＝0说明执行完毕\*/  {  P[priority].st=0;  P[priority].py=-1;  }  else if (P[priority].sb==P[priority].ct) /\* 进程的阻塞时间STARTBLOCK，\*/  /\* 表示当进程再运行STARTBLOCK个时间片后，\*/  /\* 进程将进入阻塞状态；\*/  {  P[priority].st=-1;  block\_q[0]++;  block\_q[block\_q[0]]=priority;  }  else /\* 以上两个条件都不符合，直接进入就绪队列\*/  {  ready\_q[0]++;  ready\_q[ready\_q[0]]=priority;  }  /\* 运行一次后输出运行结果\*/  printf("\n\n RUNNING RPOC:%d TIMELICE:%d\n",priority-1,timeslice);  printf(" READY\_QUEUE :");  for(i=1;i<=ready\_q[0];i++)  printf("->id%d",ready\_q[i]-1);  printf("\n");  printf(" BLOCK\_QUEUE :");  for(i=1;i<=block\_q[0];i++)  printf("->id%d",block\_q[i]-1);  printf("\n");  printf("===================================================== ===");  printf("\n %-12s","ID");  for(i=1;i<=n;i++)  printf("%8d",i-1);  printf("\n %-12s","PRIORITY");  for(i=1;i<=n;i++)  printf("%8d",P[i].py);  printf("\n %-12s","CPUTIME");  for(i=1;i<=n;i++)  printf("%8d",P[i].ct);  printf("\n %-12s","ALLTIME");  for(i=1;i<=n;i++)  printf("%8d",P[i].at);  printf("\n %-12s","STARTBLOCK");  for(i=1;i<=n;i++)  printf("%8d",P[i].sb);  printf("\n %-12s","BLOCKTIME");  for(i=1;i<=n;i++)  printf("%8d",P[i].bt);  printf("\n %-12s","STATE");  for(i=1;i<=n;i++)  {  if(P[i].st==1)  printf("%8s","READY");  else if (P[i].st==0)  printf("%8s","FINISH");  else  printf  ("%8s","BLOCK");  }  printf("\n");  }while((ready\_q[0]!=0)||(block\_q[0]!=0)); /\* 循环结束条件：就绪队列和阻塞队列全为空\*/  }  效果如下：    **【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：** |