****

**操作系统实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | 胡聿鑫 | **专业班级** | | 计科3班 | | **指导老师** | 王小妹 |
| **学号** | 2220192813 | | **实验日期** | | 2021.10.4 | | |
| **实验**  **名称** | 消息通信扩展实验 | | | | | | |
| **实验目的** | * 基本掌握 Linux 进程软中断通信的基本原理和实现方法。 * 了解消息队列的通信机制及原理，掌握消息通信相关系统调用的使用方法。 | | | | | | |
| **实验内容** | * 理解Linux进程间通信——软中断信号处理的实现方式，尝试改写程序以跳出程序3\_13的死循环，得到预期结果。 | | | | | | |
| **程**  **序**  **3\_13** | #include<unistd.h>  #include<stdio.h>  #include<signal.h>  #include<stdlib.h>  int wait\_mark;  void waiting( ) //自定义 waiting 函数与 wait(0)作用不同。其作用是通过循环使子进程停止。  {  while(wait\_mark!=0);  }  void stop( ) //调用此函数时，使 wait\_mark=0，结束 waiting 函数中的循环，使子进程继续执行。  {  wait\_mark=0;  }  int main( )  {  int p1, p2;  while((p1=fork())==-1);  if(p1>0)  {  while((p2=fork())==-1);  if(p2>0)  { //在父进程中  printf("parent\n");  **sleep(1);**  **//强制要求用户按下DEL或CTRL+C才使得父进程可以捕捉中断信号**  **/\*printf("按del捕捉中断信号\n");**  **wait\_mark=1;**  **signal(SIGINT,stop);\*/**  kill(p1,16); //发送信号 16  kill(p2,17); //发送信号 17  wait(0); //等待 p1、p2 中的某个子进程结束  wait(0); //等待 p1、p2 中的某个子进程结束  printf("parent process id killed!\n");  exit(0);  }  else  {  printf("p2\n");  wait\_mark=1;  signal(17,stop); //将信号 17 和函数 stop 关联起来  **//signal(SIGINT,SIG\_IGN);//忽略父进程发出的中断信号**  waiting(); //循环，直到父进程发送信号 17 时，调用 stop 结束循环继续执行  printf("child process 2 is killed by parent!\n");  exit(0);  }  }  else  { //子进程 p1  printf("p1\n");  wait\_mark=1;  signal(16,stop); //将信号 16 和 stop 关联起来。  **//signal(SIGINT,SIG\_IGN);//忽略父进程发出的中断信号**  waiting(); //循环，直到父进程发送信号 17 时，调用 stop 后结束循环继续执行  printf("child process 1 is killed by parent!\n");  exit(0);  }  } | | | | | | |
| **程**  **序**  **3\_14** | #include <sys/types.h>  #include <sys/msg.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  int main( )  {  int msgid; int status;  char str1[ ]={"test message:hello!"};  char str2[ ]={"test message:goodbye!"};  char str3[ ]={"The third message!"};  struct msgbuf {  long msgtype;  char msgtext[1024];  }sndmsg, rcvmsg;  if((msgid=msgget(IPC\_PRIVATE,0666)) ==-1)  {  printf("msgget error!\n");  exit(254);  }  sndmsg.msgtype=111;  sprintf(sndmsg.msgtext,str1);  if( msgsnd(msgid,&sndmsg,sizeof(str1)+1,0)==-1)  {  printf("msgsnd error!\n");  exit(254);  }  sndmsg.msgtype=222;  sprintf(sndmsg.msgtext,str2);  if( msgsnd(msgid,&sndmsg,sizeof(str2)+1,0)==-1)  {  printf("msgsnd error!\n");  exit(254);  }  sndmsg.msgtype=333;  sprintf(sndmsg.msgtext,str3);  if( msgsnd(msgid,&sndmsg,sizeof(str3)+1,0)==-1)  {  printf("msgsnd error!\n");  exit(254);  }  **if(status= msgrcv(msgid,&rcvmsg,80,222,IPC\_NOWAIT ) ==-1)**  **// if(status= msgrcv(msgid,&rcvmsg,80,333,IPC\_NOWAIT )==-1)**  **// if(status= msgrcv(msgid,&rcvmsg,80,0,IPC\_NOWAIT ) ==-1)**  {  printf("msg rcv error!\n");  exit(254);  }  printf("The receieved message: %s.\n",rcvmsg.msgtext);  msgctl(msgid,IPC\_RMID,0);  exit(0);  } | | | | | | |
| **程序3\_15** | #include <stdio.h>  #include <sys/types.h>  #include <sys/msg.h>  #include <sys/ipc.h>  #include <stdlib.h>  #define MSGKEY 75  struct msgform{  long mtype;  char msgtext[1030];  }msg;  int msgqid,i;  int main( )  {  **msgqid=msgget(MSGKEY,0777|IPC\_CREAT);**  while((i=fork())==-1);  if(!i) SERVER( );  while((i=fork())==-1);  if(!i) CLIENT( );  wait(0);  wait(0);  }  void SERVER( )  {  msgqid= msgget(MSGKEY,0777|IPC\_CREAT);  do  {  msgrcv(msgqid,&msg,1030,0,0);  printf("(server)received message %d \n", msg.mtype );  }while(msg.mtype!=1);  msgctl(msgqid,IPC\_RMID,0);  exit(0);  }  void CLIENT( )  {  int i;  msgqid=msgget(MSGKEY,0777);  for(i=10;i>=1;i--)  {  msg.mtype=i;  printf("(client)sent\n");  msgsnd(msgqid,&msg,1030,0);  }  exit(0);  } | | | | | | |
| **程序3\_16** | #include <stdio.h>  #include <sys/types.h>  #include <sys/msg.h>  #include <sys/ipc.h>  #include <stdlib.h>  **#include <string.h>**  **#include <unistd.h>**  **#include <sys/wait.h>**  **#include<signal.h>**  #define MSGKEY 75  struct msgform  {  long mtype;  char msgtext[1030]; //存放消息的内容  }msg;  int msgqid, i;  **int mark,n;**  **void waiting()**  **{**  **while(mark!=0);**  **}**  **void stop()**  **{**  **mark=0;**  **}**  void CLIENT( )  {  int i;  char string\_i[5]; //存放信息  msgqid=msgget(MSGKEY,0777); //打开一个消息队列；0777 是文件的存取权限  **n=getpid();**  **signal(SIGINT,SIG\_IGN);**  for(i=10;i>=1;i--)  {  msg.mtype=i;  printf("(client)sent\n");  sprintf(msg.msgtext,"the content of message is: ");  sprintf(string\_i, "message %d",i );  strcat(msg.msgtext,string\_i);  strcat(msg.msgtext,"\n");  msgsnd(msgqid,&msg,1030,0);  **mark=1;**  **signal(17,stop);**  **waiting();**  }  exit(0);  }  void SERVER( )  {  msgqid=msgget(MSGKEY,0777|IPC\_CREAT); //创建一个消息队列等  **mark=1;**  **signal(SIGINT,stop);**  do  {  msgrcv(msgqid,&msg,1030,0,0);  printf("(server)received message %d \n", msg.mtype );  printf("%s", msg.msgtext );  **kill(n,17);**  }while(msg.mtype!=1);  msgctl(msgqid,IPC\_RMID,0);  exit(0);  }  int main( )  {  while((i=fork())==-1);  **if(i>0)**  **{**  **SERVER();**  **}**  **else**  **{**  **CLIENT();**  **}**  wait(0);  wait(0);  } | | | | | | |
| **运行结果** | 程序3\_13运行结果：        程序3\_13(修改)运行结果：    程序3\_14运行结果：        程序3\_15运行结果：    程序3\_16运行结果： | | | | | | |
| **实**  **验**  **总**  **结** | 1. 并非只有图中所示的三种情况，还可能出现以下情况：      1. 程序中signal（）的作用是设置某一信号的对应动作，其第一个参数指明要处理的信号类型，第二个参数指明要执行的与信号关联的动作。 2. 若程序出现下图所示情况，说明未执行signal而先执行kill从而导致父进程提前结束。要解决该问题，只需要在父进程执行kill前，调用sleep函数。 3. 程序3\_16若不进行任何修改，则可能出现不同的运行结果，原因是接收和发送没有同步，而是在同时进行，既可能先完成所有发送任务才进行接收，也可能发送出一部分消息就已经开始执行接收操作。若想实现规律的一接一发，则需要与软中断结合，使得每发送一条消息，在没有接收完毕之前不会发送下一条消息。 | | | | | | |