实验5 进程同步与互斥

1. 实验目的
2. 了解Linux 如何进行进程通信，以及进程通信的程序实现。
3. 通过生产者和消费者的实例理解进程互斥。
4. 理解死锁，以及安全序列和安全状态的概念。
5. 通过编程，掌握银行家算法分配资源的实现过程。
6. 预备知识
7. Linux 进程通讯的方法
8. 管道（Pipe）及有名管道（named pipe）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。
9. 信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件生，除了用于进程间通信外，进程还可以发送信号给进程本身；[linux](http://lib.csdn.net/base/linux" \o "Linux知识库)除了支持Unix早期 信号语义函数sigal外，还支持语义符合Posix.1标准的信号函数sigaction（实际上， 该函数是基于BSD的，BSD为了实现可靠信号机制，又能够统一对外接口，sigaction函数重新实现了signal函数。
10. 报文（Message）队列（消息队列）：消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。
11. 共享内存：使得多个进程可以访问同一块内存空间，是最快的可用IPC形式。是针其他通信机制运行效率较低设计的。往往与其它通信机制，如信号量结合使用， 来达到进程间的同步及互斥。
12. 信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。
13. 套接字（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的，但现在一般可以移植到其它类Unix 系统上：Linux和System V的变种都支持套接字。
14. 管道是进程间通信

管道通信是linux进程通信的方式之一，包括无名管道和有名管道2种。前者用于父进程和子进程间的通信，后者可用于运行于同一台机器的任意两个进程间的通信。

管道的局限性：

1. 数据自己读不能自己写。
2. 数据一旦被读走，便不在管道中存在，不可反复读取。
3. 由于管道采用半双工通信方式。因此，数据只能在一个方向上流动。
4. 只能在有公共祖先的进程间使用管道。

常见的通信方式有，单工通信、半双工通信、全双工通信。

进程可以通过调用函数pipe()创建一个管道。函数pipe()的原型如下：

**int pipe(int fildes[2]);**

与该函数pipe()相对应的系统调用sys\_pipe()的原型如下：

**asmlinkage int sys\_pipe(unsigned long \_\_user \* fildes);**

从本质上来说，pipe()函数的功能就是创建一个内存文件，但与创建普通文件的函数不同，函数pipe()将在参数fildes中为进程返回这个文件的两个文件描述符fildes[0]和fildes[1]。其中，fildes[0]是一个具有“只读”属性的文件描述符，fildes[1]是一个具有“只写”属性的文件描述符，即进程通过fildes[0]只能进行文件的读操作，而通过fildes[1]只能进行文件的写操作。

管道

只写

只读

进程B

进程A

共享文 件

通过文件系统的系统调用write和read进行管道的写和读；

* + 管道按FIFO（先进先出）方式传送消息
  + 只能单向传送消息，进程间双向通信，通常需要两个管道；
  + 只适用于父子进程之间或父进程安排的各个子进程之间。
* 发送进程利用文件系统的系统调用write(fd[1],buf,size), 把buf的长度为size字符的消息送入管道入fd[1]。
* 接收进程则使用系统调用read(fd[0],buf,size), 从管道出fd[0]读出size字符的消息置入buf中**。**
* 如同消息缓冲一样，在对pipe文件进行读写操作过程中要对发送进程和接送进程实施正确的同步与互斥以确保通信的正确性.

1. 实验内容

**例1：父子进程使用管道通信，父进程写入字符串，子进程读出并输出显示到屏幕。**

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define INPUT 0

#define OUTPUT 1

int main() {

int fd[2]; /\*定义子进程号 \*/

pid\_t pid;

char buf[256];

int returned\_count; /\*创建无名管道\*/

pipe(fd); /\*创建子进程\*/

pid=fork();

if(pid<0) {

printf("Error in fork\n");

exit(1);

}else if(pid == 0) { /\*执行子进程\*/

printf("in the child process...\n");

/\*子进程向父进程写数据，关闭管道的读端\*/

close(fd[INPUT]);

write(fd[OUTPUT], "hello world", strlen("hello world"));

exit(0);

}else { /\*执行父进程\*/

printf("in the parent process...\n");

/\*父进程从管道读取子进程写的数据，关闭管道的写端\*/

close(fd[OUTPUT]);

returned\_count = read(fd[INPUT], buf, sizeof(buf));

printf("%d bytes of data received from child process: %s\n", returned\_count, buf);

}

return 0;

}

运行结果：in the parent process...

in the child process...

11 bytes of data received from child process: hello world

1. **一个pipe有一个写端的验证。**

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#include <stdlib.h>

int main(void)

{

pid\_t pid;

int fd[2], i, n;

char buf[1024];

int ret = pipe(fd);

if(ret == -1){

perror("pipe error");

exit(1);

}

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

if((pid = fork()) == 0)

break;

else if(pid == -1){

perror("pipe error");

exit(1);

}

}

if (i == 0) {

close(fd[0]);

write(fd[1], "1.hello\n", strlen("1.hello\n"));

} else if(i == 1) {

close(fd[0]);

write(fd[1], "2.world\n", strlen("2.world\n"));

} else {

close(fd[1]); //父进程关闭写端,留读端读取数据

// sleep(1);

n = read(fd[0], buf, 1024); //从管道中读数据

write(STDOUT\_FILENO, buf, n);

for(i = 0; i < 2; i++) //两个儿子wait两次

wait(NULL);

}

return 0;

}

运行结果：1.hello

1. **生产者和消费者环形队列的实现**

预备知识

操作函数

**#include <semaphore.h>**

**int sem\_init(sem\_t \*sem, int pshared, unsigned int value);**

**int sem\_destroy(sem\_t \*sem);**

**int sem\_wait(sem\_t \*sem);**

**int sem\_trywait(sem\_t \*sem);**

**int sem\_post(sem\_t \*sem);**

初始化信号量sem\_init，参数value为信号量的值，参数pshared一般设为0，表示信号量用于同一进程内线程间同步。摧毁信号量sem\_destroy。P操作（申请资源）sem\_wait，使信号量的值-1。V操作（释放资源）sem\_post，使信号量的值+1。sem\_trywait是尝试申请资源。

代码实现：

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

#define \_SIZE\_ 5

sem\_t blanks; //表示格子的信号量

sem\_t datas; //表示商品的信号量

int buf[\_SIZE\_] ={ 0 };

//生产者

void\* product(void\* arg)

{

int i = 0;

int count=0;

while(count<=5)

{

usleep(500);

sem\_wait(&blanks); //生产者申请格子资源

int data = rand()%1000;

buf[i] = data;

printf("Product is:%d\n",data);

sem\_post(&datas); //每生产一个商品就需要对商品信号量+1

count++;

++i;

i %= \_SIZE\_;

}

}

//消费者

void\* consumer(void\* arg)

{

int i = 0;

int count=0;

while(1)

{

usleep(500);

sem\_wait(&datas); //消费者申请商品资源

printf("Consumer is%d\n",buf[i]);

sem\_post(&blanks); //买走一个商品，就多了一个空格子

count++;

++i;

i %= \_SIZE\_;

}

}

int main()

{

sem\_init(&blanks,0,\_SIZE\_);

sem\_init(&datas,0,0);

pthread\_t \_consumer;

pthread\_t \_product;

pthread\_create(&\_consumer,NULL,consumer,NULL);

pthread\_create(&\_product,NULL,product,NULL);

pthread\_join(\_consumer,NULL);

pthread\_join(\_product,NULL);

sem\_destroy(&blanks);

sem\_destroy(&datas);

return 0;

}

运行时输入gcc filename.c -pthread -o filename

运行结果：

Product is:383

Consumer is383

Product is:886

Consumer is886

Product is:777

Consumer is777

Product is:915

Consumer is915

Product is:793

Consumer is793

Product is:335

Consumer is335

Product is:386

Consumer is386

1. 实验题目
2. 进程间的管道通信实验要求：

编写程序实现进程间的管道通信。使用系统调用pipe（）建立一个管道，二个子进程分别向管道中各写一句话，例如：

Child 1 is sending a message!

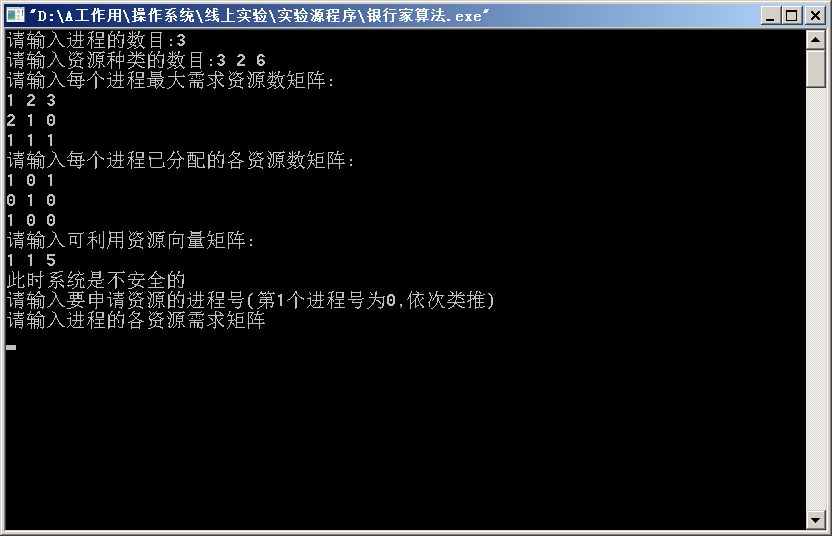
Child 2 is sending a message!

父进程从管道中读取来自二个子进程的信息并显示。要求先接收进程P1，再接收P2。

1. 银行家算法实验要求：
2. 给出系统可用资源向量，例如：系统可用资源=（5，3，2）。
3. 给出若干进程最大需求矩阵。
4. 采用时间片轮转法调度进程。
5. 进程执行时提出资源请求（可利用随机数给出或从键盘输入）。
6. 判断资源是否可以安全分配，要求进程每提出一个资源请求，都要进行安全判断并给出安全序列，否则给出提示。

可以参照下面的C++程序修改为linux下的C语言程序。

运行结果示例：



#include<iostream>

using namespace std;

#define MAXPROCESS 50 //最大进程数

#define MAXRESOURCE 100 //最大资源数

int Available[MAXRESOURCE]; //可用资源数组

int Max[MAXPROCESS][MAXRESOURCE]; //最大需求矩阵

int Allocation[MAXPROCESS][MAXRESOURCE]; //分配矩阵

int Need[MAXPROCESS][MAXRESOURCE]; //需求矩阵

int Request[MAXPROCESS][MAXRESOURCE]; //进程需要资源数

bool Finish[MAXPROCESS]; //系统是否有足够的资源分配

int p[MAXPROCESS]; //记录序列

int m,n; //m个进程,n个资源

void Init()

{ //初始化算法

int i,j;

cout<<"请输入进程的数目:";

cin>>m;

cout<<"请输入资源种类的数目:";

cin>>n;

cout<<"请输入每个进程最大需求资源数矩阵:"<<endl;

for(i=0;i<m;i++)

{

for(j=0;j<n;j++)

{

cin>>Max[i][j];

}

}

cout<<"请输入每个进程已分配的各资源数矩阵:"<<endl;

for(i=0;i<m;i++) //判断输入的资源数是否合理

{

for(j=0;j<n;j++)

{

cin>>Allocation[i][j];

Need[i][j]=Max[i][j]-Allocation[i][j];//各进程尚需要的资源数

if(Need[i][j]<0)

{

cout<<"您输入的第"<<i+1<<"个进程的第"<<j+1<<"个资源数有错误,请重新输入:"<<endl;

j--;

continue;

}

}

}

cout<<"请输入可利用资源向量矩阵:"<<endl;

for(i=0;i<n;i++)

cin>>Available[i];

}

bool Safe() //判断安全函数

{ int i,j,k,l=0;

int Work[MAXRESOURCE]; //系统可提供给进程继续运行所需要的各类资源数目

for(i=0;i<n;i++)

Work[i]=Available[i];

for(i=0;i<m;i++)

Finish[i]=false;

for(i=0;i<m;i++)

{ if(Finish[i]==true)

continue;

else

{

for(j=0;j<n;j++)

if(Need[i][j]>Work[j])

break;

if(j==n)

{

Finish[i]=true;

for(k=0;k<n;k++)

Work[k]+=Allocation[i][k];

p[l++]=i;

i=-1;

}

else continue;

}

if(l==m) //如果系统是安全的，将资源分配给进程

{

cout<<"此时系统是安全的"<<endl;

cout<<"安全序列为:"<<endl;

for(i=0;i<l;i++)

{

cout<<p[i];

if(i!=l-1)

cout<<"-->";

}

cout<<""<<endl;

return true;

}

}

cout<<"此时系统是不安全的"<<endl;

return false;

}

void Bank() /\*银行家算法\*/

{

int i,cusneed;

char again;

while(1)

{

cout<<"请输入要申请资源的进程号(第1个进程号为0,依次类推)"<<endl;

cin>>cusneed;

cout<<"请输入进程的各资源需求矩阵"<<endl;

for(i=0;i<n;i++)

{

cin>>Request[cusneed][i];

}

for(i=0;i<n;i++)

{

if(Request[cusneed][i]>Need[cusneed][i])

{

cout<<"您输入的请求数超过进程的需求量!请重新输入!"<<endl;

continue;

}

if(Request[cusneed][i]>Available[i])

{

cout<<"您输入的请求数超过系统拥有的资源数!请重新输入!"<<endl;

continue;

}

}

for(i=0;i<n;i++) //假如系统将资源分配给p[i]

{

Available[i]-=Request[cusneed][i];

Allocation[cusneed][i]+=Request[cusneed][i];

Need[cusneed][i]-=Request[cusneed][i];

}

if(Safe()) //执行判断是否安全函数

{

cout<<"同意分配请求!"<<endl;

}

else

{

cout<<"您的请求被拒绝!"<<endl;

for(i=0;i<n;i++) //不安全的，回复资源分配表

{

Available[i]+=Request[cusneed][i];

Allocation[cusneed][i]-=Request[cusneed][i];

Need[cusneed][i]+=Request[cusneed][i];

}

}

for(i=0;i<m;i++)

{

Finish[i]=false;

}

cout<<"您还想再次请求分配吗?是请按y/Y,否请按其它键"<<endl;

cin>>again;

if(again=='y'||again=='Y')

{

continue;

}

break;

}

}

void main()

{

Init();

Safe();

Bank();

}

五、实验报告

将调试运行成功的程序和写好的实验报告一起压缩打包，以实验X-学号-姓名.rar这样的形式命名，并上传提交。