Lab7 通信机制

> 实习说明
>
> 本实习希望通过修改Nachos系统平台的底层源代码，达到“实现通信机制”的目标。

任务完成情况

Exercises	Y/N
Exercise1	*Y*
Exercise2	*Y*
Exercise3	*Y*

Exercise1 调研*Linux*中进程通信机制的实现

> 进程间通信*IPC* ——简书
>
> 每个进程各自有不同的用户地址空间，任何一个进程的全局变量在另一个进程中都看不到，所以进程之间要交换数据必须通过内核，在内核中开辟一块缓冲区，进程1把数据从用户空间拷到内核缓冲区，进程2再从内核缓冲区把数据读走，内核提供的这种机制称为进程间通信（*IPC*，*InterProcess Communication*）

*Linux*中进程通信的方式包括：管道，信号，消息队列，共享内存，信号量和套接字六种。

管道是一种半双工的通信机制，采用循环队列实现。管道包括有名管道和匿名管道两种，匿名管道在内存中开辟一个缓冲区，只能供父子或兄弟进程之间通信；而有名管道为了解决匿名管道的局限性，为管道赋予名字，有名管道可以支持两个无关进程之间的通信。

信号是软件层次上对中断的一种模拟，是一种异步通信方式，信号可以在用户空间和内核之间直接交互。

消息队列是存放在内核中的链表结构，用队列标识符表示。目前有*POSIX*和*System V*两种消息队列，后者被广泛使用。延伸阅读：消息队列C语言的实践

共享内存是内核维护的一片独立的内存区域，它允许多个进程进行读写，需要信号量机制来实现进程的同步和互斥。延伸阅读：Linux支持的主要三种共享内存方式：*mmap*()系统调用、*Posix*共享内存，以及*System V*共享内存实践

信号量是一种非负的资源计数器，用于进程对临界资源的同步和互斥。包括两个操作：*wakeup*()和*signal*()，AKA，*PV*操作。

套接字广泛用于网络通信，它支持客户/服务器之间不同的进程通信，套接字包括三个属性：域/端口号/协议类型。

> 其他参考资料
>
> *Linux*进程间通信的几种方式总结--*linux*内核剖析（七）

Exercise2 为*Nachos*设计并实现一种线程/进程间通信机制

> 基于已完善的线程、调度、同步机制部分，选择*Linux*中的一种进程通信机制，在*Nachos*上实现。

我选择实现管道。

有名管道

有名管道的实质就是一个文件，它实现了没有亲缘关系进 程之间的通信。本次测试之前创建了一个模板文件*file1*，里面存放了数据，因为我在给字符串*buffer*幅值的过程中一直会报错，所以我不得不采取从文件读入的方式来初始化*buffer*。

在*code/test/*目录下有两个用于测试的*c*程序*pipe_writer.c*和*pipe_reader.c*:

```
//lab7 pipe
//写者线程负责创建管道，并向管道中写入数据；然后调用Exec()执行读者线程
#include "syscall.h"
#define QUOTE_LEN 40    //诗的长度
int fileID1, fileID2;   //两个打开文件的id
int numBytes;           //实际读取的字符数
char buffer[QUOTE_LEN]; //缓冲区
int main()
{
    Create("file2");                                //创建文件
    fileID1 = Open("file1");                        //打开文件file1
    fileID2 = Open("file2");                        //打开文件file2
    numBytes = Read(buffer, QUOTE_LEN, fileID1); //从file1中读取字节到buffer中
    Write(buffer, numBytes, fileID2);            //再从buffer中写入file2中
    Exec("../test/pipe_reader");                 //执行读者线程
    Close(fileID1);                              //关闭两个文件
    Close(fileID2);
    Exit(0);
}
```

```
//pipe_reader.c
// 读者线程从管道中读取数据。
#include "syscall.h"
#define QUOTE_SIZE 40
char buffer[QUOTE_SIZE];
int pipeID;

int main()
{
    pipeID = Open("file2");             //打开管道
    Read(buffer, QUOTE_SIZE, pipeID); //从管道中读入数据
    Close(pipeID);                      //关闭管道
    Exit(0);                            //退出
}
```

匿名管道

匿名管道不需要创建一个文件

```
#include "syscall.h"    //匿名管道
#define QUOTE_LEN 40    //诗的长度
char buffer[QUOTE_LEN]; //缓冲区

void child()
{
    int dummy;                          //dummy继承自父亲,但是需要声明一下，否则会报错
    Read(buffer, QUOTE_LEN, dummy); //从管道中读入数据
```

```
    Close(dummy);                   //关闭文件
    Exit(0);
}

int main()
{
    int fileID1, fileID2, dummy;                //两个打开文件的id和一个dummy变量，
用于继承父线程的打开文件
    int numBytes;                               //实际读取的字符数
    Create("file2");                            //创建管道
    fileID1 = Open("file1");                    //打开文件file1
    fileID2 = Open("file2");                    //打开管道file2
    dummy = fileID2;                            //为dummy赋值，Nachos只支持这种笨拙
的方式
    numBytes = Read(buffer, QUOTE_LEN, fileID1); //从file1中读取字节到buffer中
    Write(buffer, numBytes, fileID2);           //再从buffer中写入file2中
    Fork(child);                                //执行读者线程
    Close(fileID1);
    Exit(0);
}
```

Exercise3 为实现的通信机制编写测试用例

有名管道测试

```
vagrant@precise32:/vagrant/nachos/nachos-3.4/code/code/userprog$ ./nachos -d t -x
../test/pipe_writer
Success create file name file2
Success open file name file1 with ID 161383400
Success open file name file2 with ID 161383416
Success read 25bytes from file ID 161383400
Success write 25bytes to file ID161383416
Forking thread "exec_thread" with func = 0x804f6e1, arg = 448
Success delete file ID 161383400
Success delete file ID 161383416
main exists with status 0.
Switching from thread "main" to thread "exec_thread"
Success open file name file2
Success read 25bytes from file ID 161383400
Success delete file ID 161383400
exec_thread exists with status 0.
```

结论

两个没有亲缘关系的线程顺利通过管道进行通信，实验成功。

匿名管道测试

```
vagrant@precise32:/vagrant/nachos/nachos-3.4/code/code/userprog$ ./nachos -d t -x
../test/pipe_writer
Success create file name file2
Success open file name file1 with ID157996008
Success open file name file2 with ID157996024
Success read 25bytes from file ID157996008
Success write 25bytes to file ID157996024
Forking thread "child_thread" with func = 0x804f7d9, arg = 208
Success delete file ID157996008
main exists with status 0.
Switching from thread "main" to thread "child_thread"
Success read 25bytes from file ID157996024
Success delete file ID157996024
fork_thread exists with status 0.
```

结论

使用*fork*()产生的子线程继承了父线程的打开文件，利用这个在内存中的“文件”实现了通信，符合匿名管道的定义，实验成功。

困难&解决

其实可以使用系统级线程来实现管道测试，更简单，也不会报奇奇怪怪的错。用用户级线程的体会就是：变量声明和赋值不能同时进行，不能*include*标准库以及没有*debug*手段，所以我的代码都写得很保守，不敢“长篇大论”，否则一旦报错，甚至都找不到原因。。。

参考文献

https://wenku.baidu.com/view/b4c693d1d05abe23482fb4daa58da0116c171f1e