#### 第七章 进程之间的通信

1. 进程之间的通信有哪些方法？分别由哪些优缺点？

进程间通信方法：

命名管道、匿名管道、消息队列、信号量、共享内存、socket、邮槽

优缺点：

命名管道可方便地用于不同计算机、不同操作系统的不同进程间的通信，有完善的权限控制机制，但是在多个进程间使用命名管道通信不够方便；

匿名管道的优势在于简洁占用小，但是仅局限与本地的父子进程间通讯；

消息队列是内核对象，独立于发送与接收消息的进程，可有效实现异步通信，以避免进程消息同步问题，但消息队列需要一定的资源加以维护；

信号量常用于进程间简单消息的传递，用于实现进程的同步操作，缺点是无法传输大量数据；

共享内存是在内存区域中开辟一块空间，并将其映射到多个进程的内存空间，利于大量数据的交换操作，但在程序设计时必须考虑进程同步的问题；

邮槽只是单向通信的，但邮槽可以跨机器间通信；

socket层是运输层上的一个抽象层，可以方便地用于本地进程通信与网络中进程的通信，但socket编程相对复杂。

2. 编写一个多进程程序，进程之间通过管道通信

1）利用无名管道，父进程与子进程之间双向通信，进程写成无限循环的形式

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#define TRUE 1

int main(void)

{

int AnonymousPipe\_1[2] = { 0 }, AnonymousPipe\_2[2] = { 0 };

int readchnum;

unsigned char PipeBuf[1024] = { 0 };

pid\_t pid;

//try to create anonymous pipe

if (pipe(AnonymousPipe\_1) < 0 || pipe(AnonymousPipe\_2) < 0)

{

printf("Create Anonymous Pipe Error!\n");

return -1;

}

pid = fork();

//fork failed

if (pid < 0)

{

printf("Fork Process Error!\n");

return -1;

}

//parent process

else if (pid)

{

close(AnonymousPipe\_1[0]);

while (TRUE)

{

printf("This is parent process, say something to child:\n");

scanf("%s", PipeBuf);

//send to first pipe

write(AnonymousPipe\_1[1], PipeBuf, strlen(PipeBuf));

//recv from second pipe

readchnum = read(AnonymousPipe\_2[0], PipeBuf, 1024);

PipeBuf[readchnum] = 0;//end of buffer

//show

printf("This is parent process, heard from child:\n%s\n\n", PipeBuf);

}

}

//child process

else

{

close(AnonymousPipe\_2[0]);

while (TRUE)

{

//recv from first pipe

readchnum = read(AnonymousPipe\_1[0], PipeBuf, 1024);

PipeBuf[readchnum] = 0;//end of buffer

//show

printf("This is child process, heard from parent:\n%s\n\n", PipeBuf);

printf("This is child process, say something to parent:\n");

scanf("%s", PipeBuf);

//send to second pipe

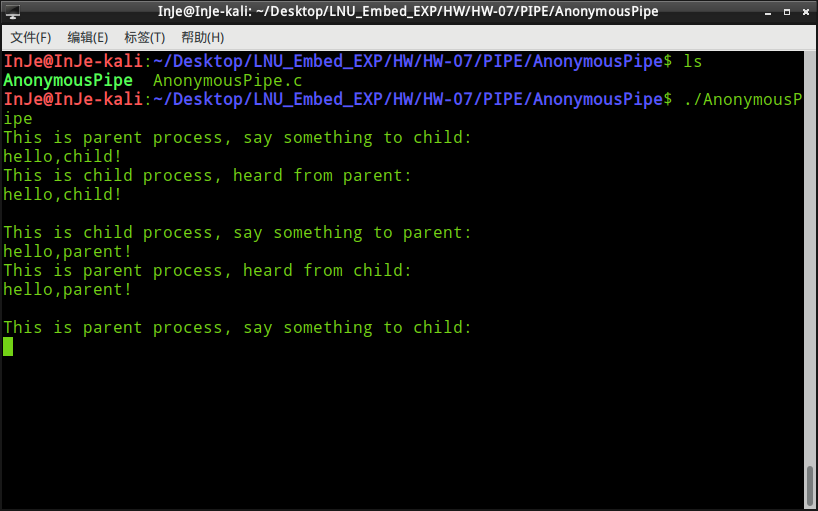
write(AnonymousPipe\_2[1], PipeBuf, strlen(PipeBuf));

}

}

return 0;

}

 2）利用有名管道，两个进程之间双向通信，进程写成无限循环的形式

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#define NAME\_NAMEDPIPE1 "/tmp/testFIFO\_1"

#define NAME\_NAMEDPIPE2 "/tmp/testFIFO\_2"

#define TRUE 1

int main(void)

{

int NamedPipe\_1, NamedPipe\_2;

int readchnum;

unsigned char PipeBuf[1024] = { 0 };

pid\_t pid;

//clear FIFO file

remove(NAME\_NAMEDPIPE1);

remove(NAME\_NAMEDPIPE2);

//try to create named pipe

if (mkfifo(NAME\_NAMEDPIPE1, 0666) || mkfifo(NAME\_NAMEDPIPE2, 0666))

{

printf("Create Named Pipe Error!\n");

return -1;

}

pid = fork();

//fork failed

if (pid < 0)

{

printf("Fork Process Error!\n");

return -1;

}

//parent process

else if (pid)

{

NamedPipe\_1 = open(NAME\_NAMEDPIPE1, O\_RDONLY);

NamedPipe\_2 = open(NAME\_NAMEDPIPE2, O\_WRONLY);

while (TRUE)

{

printf("This is parent process, say something to child:\n");

scanf("%s", PipeBuf);

//send to second pipe

write(NamedPipe\_2, PipeBuf, strlen(PipeBuf));

//recv from first pipe

readchnum = read(NamedPipe\_1, PipeBuf, 1024);

PipeBuf[readchnum] = 0;//end of buffer

//show

printf("This is parent process, heard from child:\n%s\n\n", PipeBuf);

}

}

//child process

else

{

NamedPipe\_1 = open(NAME\_NAMEDPIPE1, O\_WRONLY);

NamedPipe\_2 = open(NAME\_NAMEDPIPE2, O\_RDONLY);

while (TRUE)

{

//recv from second pipe

readchnum = read(NamedPipe\_2, PipeBuf, 1024);

PipeBuf[readchnum] = 0;//end of buffer

//show

printf("This is child process, heard from parent:\n%s\n\n", PipeBuf);

printf("This is child process, say something to parent:\n");

scanf("%s", PipeBuf);

//send to first pipe

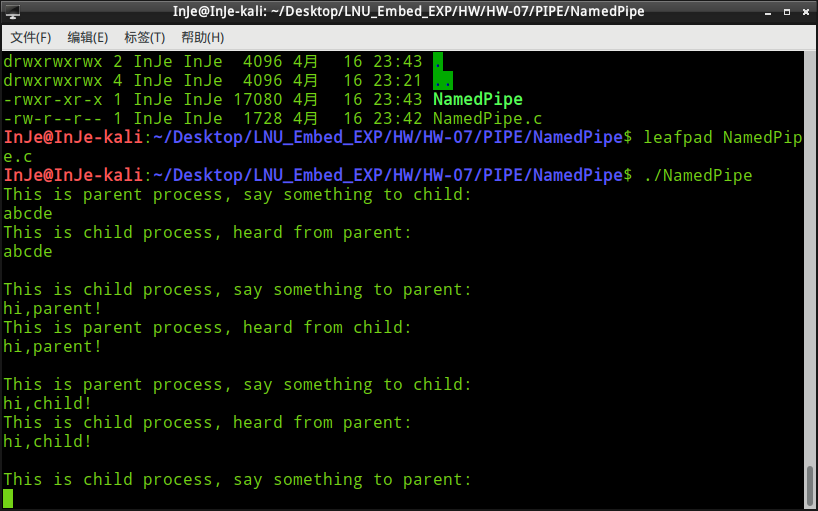
write(NamedPipe\_1, PipeBuf, strlen(PipeBuf));

}

}

return 0;

}



3. 编写两个进程程序，mysig1.c mysig2.c，mysig1向mysig2发信号，mysig2收到信号后，显示收到的信号。运行时，先运行mysig2，查看mysig2的pid，然后修改编译mysig1，再运行mysig1。

4. 编写一个多进程程序，进程之间通过消息队列通信

5. 编写一个多进程程序，进程之间通过信号量通信

6. 编写两个程序

1) hw0701.c， 打开一个已存在的文件，把文件内容复制到共享内存

2) hw0702.c，创建文件，从共享内存中读取数据，写入文件中，并且做对比