#### 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

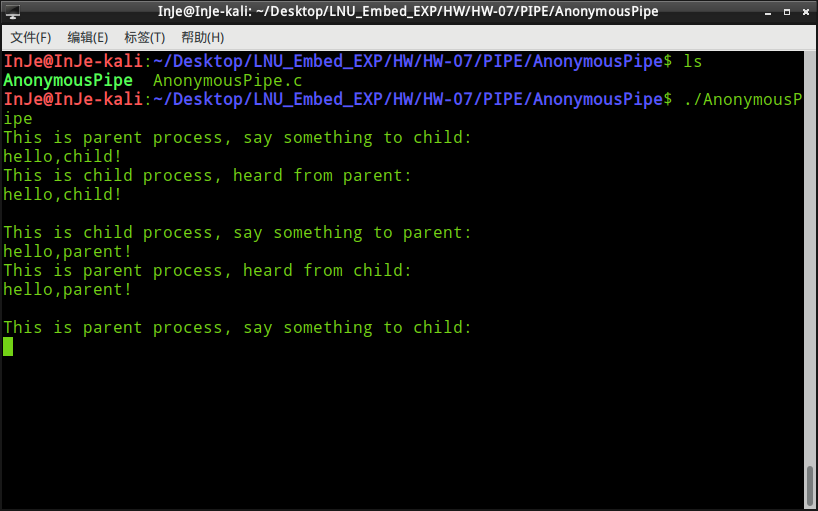
write(AnonymousPipe\_2[1], PipeBuf, strlen(PipeBuf));

}

}

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

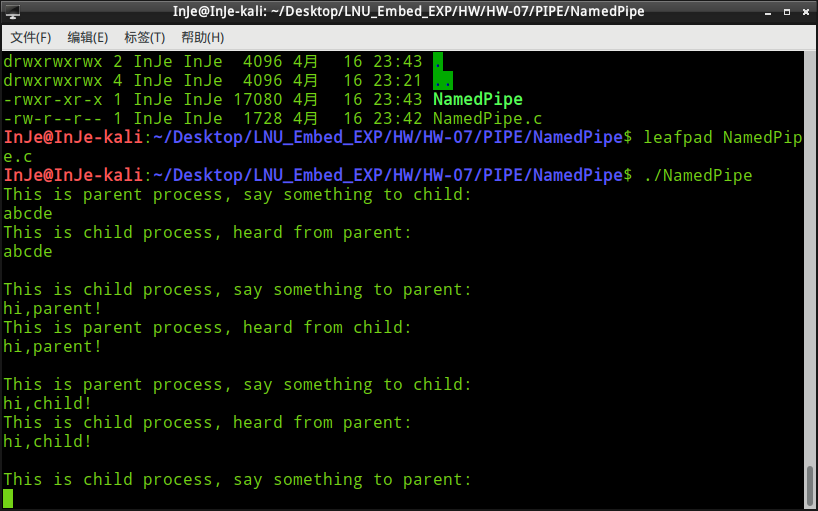
write(NamedPipe\_1, PipeBuf, strlen(PipeBuf));

}

}

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。

//mysig1.c

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

#include<sys/types.h>

#define TRUE 1

int main(void)

{

int PID, SIGNO;

printf("You can send signal to other process here.\n");

printf("Format:[PID] [SIGNO]\n");

while (TRUE)

{

scanf("%d %d", &PID, &SIGNO);

kill(PID, SIGNO);

}

return 0;

}

//mysig2.c

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

#include<sys/types.h>

#define TRUE 1

//to deal with signal recv

void CALLBACK\_signal(int SIG)

{

switch (SIG)

{

//suspend terminal

case SIGHUP:

printf("Catch [SIGHUP] signal!\n");

break;

//kbd INT

case SIGINT:

printf("Catch [SIGINT] signal!\n");

break;

//press quit key

case SIGQUIT:

printf("Catch [SIGQUIT] signal!\n");

break;

//others

default:

printf("Catch other signal!\n");

break;

}

return;

}

int main(void)

{

printf("Process ID of me is [ %d ]\n", getpid());

//deal with three signals

signal(SIGHUP, CALLBACK\_signal);

signal(SIGINT, CALLBACK\_signal);

signal(SIGQUIT, CALLBACK\_signal);

//loop show

while (TRUE)

{

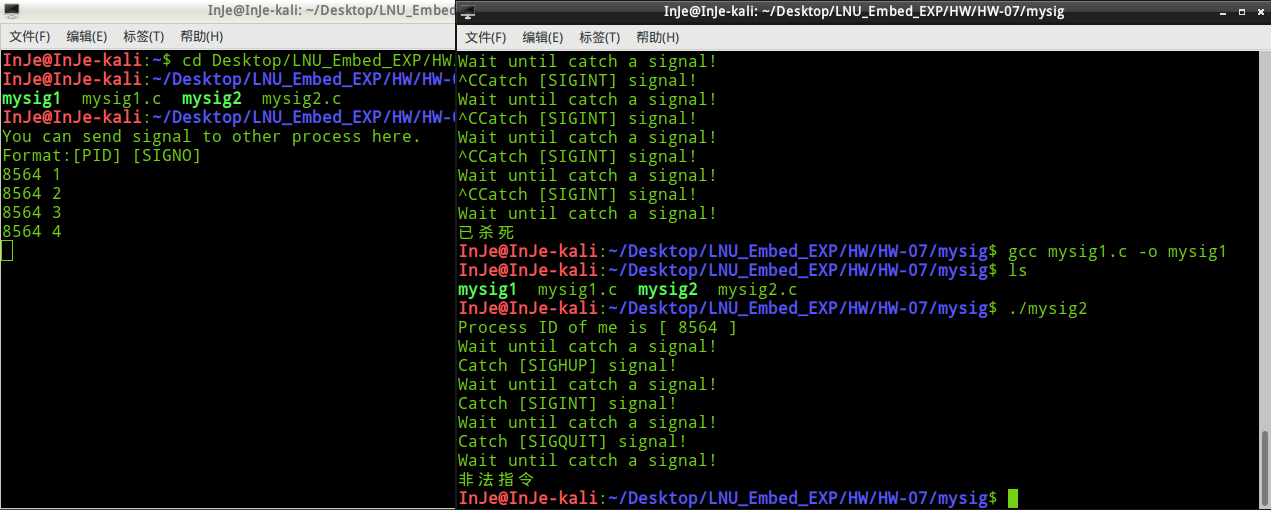
printf("Wait until catch a signal!\n");

pause();

}

return 0;

}



4. 编写一个多进程程序，进程之间通过消息队列通信

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/msg.h>

#define MAX\_MSGLEN 1024

#define ID\_MSGQUEUE 666

#define TRUE 1

typedef struct

{

long msg\_type;

unsigned char msg[MAX\_MSGLEN];

}MSG\_ATOM;

int main(void)

{

MSG\_ATOM msgpkg;

int msgQID;

unsigned char msgbuf[MAX\_MSGLEN] = { 0 };

pid\_t pid;

//try to create message queue

msgQID = msgget(ID\_MSGQUEUE, 0666 | IPC\_CREAT);

if (msgQID == -1)

{

printf("Create Message Queue Error!\n");

return -1;

}

printf("TIP: Send [QUIT] to quit.\n");

pid = fork();

//fork failed

if (pid < 0)

{

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

return -1;

}

//parent process

else if (pid)

{

while (TRUE)

{

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

scanf("%s", msgbuf);

strcpy(msgpkg.msg, msgbuf);

msgpkg.msg\_type = 1;

//send type1 msg

if (msgsnd(msgQID, &msgpkg, MAX\_MSGLEN, 0) == -1)

{

printf("Send message failed!\n");

return -1;

}

//recv type2 msg

if (msgrcv(msgQID, &msgpkg, MAX\_MSGLEN, 2, 0) == -1)

{

printf("Recv message failed!\n");

return -1;

}

//parent recv QUIT msg, delete msgQueue and exit.

if (!strcmp(msgpkg.msg, "QUIT"))

{

msgctl(msgQID, IPC\_RMID, 0);

printf("Parent process exit!\n");

return 0;

}

//show

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

}

}

//child process

else

{

while (TRUE)

{

//recv type1 msg

if (msgrcv(msgQID, &msgpkg, MAX\_MSGLEN, 1, 0) == -1)

{

printf("Recv message failed!\n");

return -1;

}

//child recv QUIT msg, send QUIT to parent and exit

if (!strcmp(msgpkg.msg, "QUIT"))

{

msgpkg.msg\_type = 2;

strcpy(msgpkg.msg, "QUIT");

if (msgsnd(msgQID, &msgpkg, MAX\_MSGLEN, 0) == -1)

{

printf("Send message failed!\n");

return -1;

}

printf("Child process exit!\n");

return 0;

}

//show

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

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

scanf("%s", msgbuf);

strcpy(msgpkg.msg, msgbuf);

msgpkg.msg\_type = 2;

//send type2 msg

if (msgsnd(msgQID, &msgpkg, MAX\_MSGLEN, 0) == -1)

{

printf("Send message failed!\n");

return -1;

}

//child send QUIT msg, exit

if (!strcmp(msgbuf, "QUIT"))

{

printf("Child process exit!\n");

return 0;

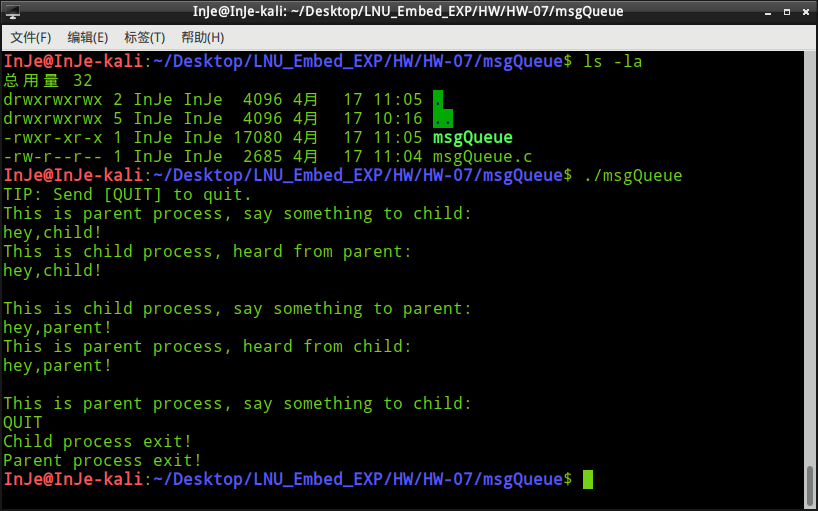
}

}

}

return 0;

}



5. 编写一个多进程程序，进程之间通过信号量通信

//Consumer.c

/\*

Single Consumer

Execute Producer First to initialize semaphore!

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/sem.h>

#include<time.h>

#define ID\_SEM\_FULL 666

#define ID\_SEM\_EMPTY 777

#define ID\_SEM\_MUTEX 888

#define NUM\_SEM 5 //5 boxes to be used

#define TRUE 1

//P

int Wait(int SemID)

{

struct sembuf sem;

sem.sem\_num = 0;

sem.sem\_op = -1;

sem.sem\_flg = SEM\_UNDO;

semop(SemID, &sem, 1);

return 0;

}

//V

int Signal(int SemID)

{

struct sembuf sem;

sem.sem\_num = 0;

sem.sem\_op = 1;

sem.sem\_flg = SEM\_UNDO;

semop(SemID, &sem, 1);

return 0;

}

int main(void)

{

int SemID\_Full, SemID\_Empty, SemID\_Mutex;

time\_t t;

struct tm \*LocalTime = NULL;

//try to create or get semaphore

SemID\_Full = semget(ID\_SEM\_FULL, 1, 0666 | IPC\_CREAT);

SemID\_Empty = semget(ID\_SEM\_EMPTY, 1, 0666 | IPC\_CREAT);

SemID\_Mutex = semget(ID\_SEM\_MUTEX, 1, 0666 | IPC\_CREAT);

if (SemID\_Full == -1 || SemID\_Empty == -1 || SemID\_Mutex == -1)

{

printf("Create / Get Semaphore failed!\n");

return -1;

}

printf("Get Semaphore!\n");

//consume loop

while (TRUE)

{

time(&t);

Wait(SemID\_Mutex);

Wait(SemID\_Empty);

printf("%s - Get\n", asctime(localtime(&t)));

Signal(SemID\_Full);

Signal(SemID\_Mutex);

sleep(1);

}

return 0;

}

//Producer.c

/\*

Single Producer

Execute Producer First to initialize semaphore!

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/sem.h>

#include<time.h>

#define ID\_SEM\_FULL 666

#define ID\_SEM\_EMPTY 777

#define ID\_SEM\_MUTEX 888

#define NUM\_SEM 5 //5 boxes to be used

#define TRUE 1

//P

int Wait(int SemID)

{

struct sembuf sem;

sem.sem\_num = 0;

sem.sem\_op = -1;

sem.sem\_flg = SEM\_UNDO;

semop(SemID, &sem, 1);

return 0;

}

//V

int Signal(int SemID)

{

struct sembuf sem;

sem.sem\_num = 0;

sem.sem\_op = 1;

sem.sem\_flg = SEM\_UNDO;

semop(SemID, &sem, 1);

return 0;

}

int main(void)

{

int SemID\_Full, SemID\_Empty, SemID\_Mutex;

time\_t t;

struct tm \*LocalTime = NULL;

//try to create or get semaphore

SemID\_Full = semget(ID\_SEM\_FULL, 1, 0666 | IPC\_CREAT);

SemID\_Empty = semget(ID\_SEM\_EMPTY, 1, 0666 | IPC\_CREAT);

SemID\_Mutex = semget(ID\_SEM\_MUTEX, 1, 0666 | IPC\_CREAT);

if (SemID\_Full == -1 || SemID\_Empty == -1 || SemID\_Mutex == -1)

{

printf("Create / Get Semaphore failed!\n");

return -1;

}

semctl(SemID\_Full, 0, SETVAL, NUM\_SEM);//Full = 5

semctl(SemID\_Empty, 0, SETVAL, 0);//Empty = 0

semctl(SemID\_Mutex, 0, SETVAL, 1);//Mutex = 1

printf("Semaphore initialized!\n");

//produce loop

while (TRUE)

{

time(&t);

printf("%s - Produce\n", asctime(localtime(&t)));

Wait(SemID\_Mutex);

Wait(SemID\_Full);

time(&t);

printf("%s - Put\n", asctime(localtime(&t)));

Signal(SemID\_Empty);

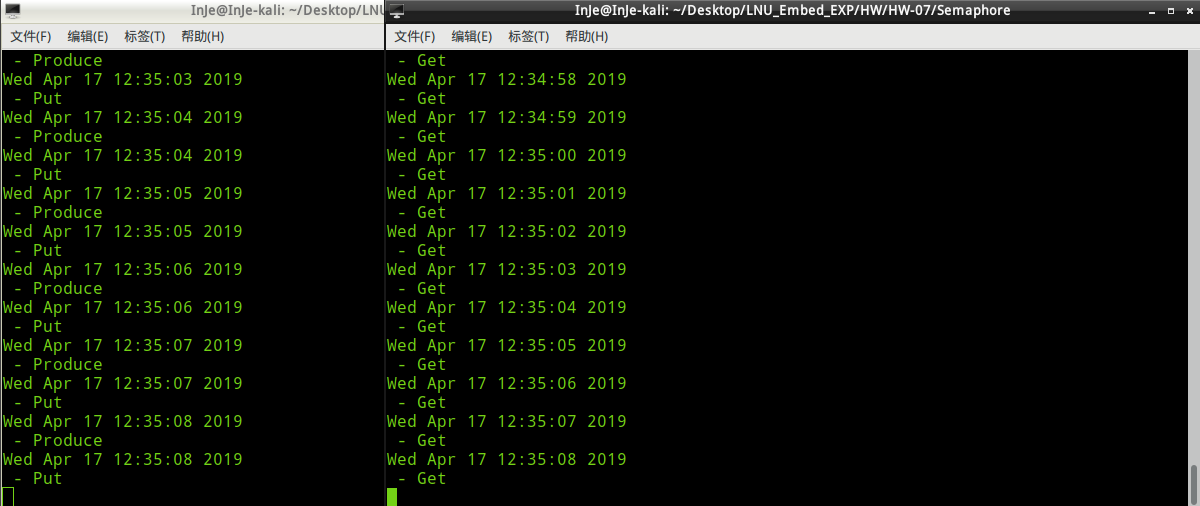
Signal(SemID\_Mutex);

sleep(1);

}

return 0;

}

生产者-消费者是个动态的过程，上图可能无法有效展示

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

#include<stdio.h>

#include<unistd.h>

#include<string.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#define BufferSize 8192

#define ID\_SHMem 666

int main(void)

{

int IDSHMem;

void \*ptr\_Mem = NULL;

FILE \*pF = NULL;

unsigned char \*ptr\_SharedData = NULL;

unsigned char FileData[BufferSize] = { 0 };

//get file content

pF = fopen("AwesomeFile.bin", "r");

while (fread(FileData, 1, BufferSize, pF));

fclose(pF);

printf("Get file content succeeded!\n");

//try to create SHMem

IDSHMem = shmget(ID\_SHMem, BufferSize, 0666 | IPC\_CREAT);

if (IDSHMem == -1)

{

printf("Create SHMem segment failed!\n");

return -1;

}

printf("Create SHMem segment succeeded!\n");

//try to map memory segment

ptr\_Mem = shmat(IDSHMem, 0, 0);

if (ptr\_Mem == (void \*)-1)

{

printf("Map memory failed!\n");

return -1;

}

printf("Map shared memory succeeded!\n");

//point shared data segment to SHMem

ptr\_SharedData = (unsigned char\*)ptr\_Mem;

printf("Point SharedData segment to SHMem succeeded!\n");

//write filedata to SharedData segment

strcpy(ptr\_SharedData, FileData);

printf("Write file content to shared memory succeeded!\n");

//test SHMem

printf("Check the shared memory segment:\n[%s]\n", ptr\_SharedData);

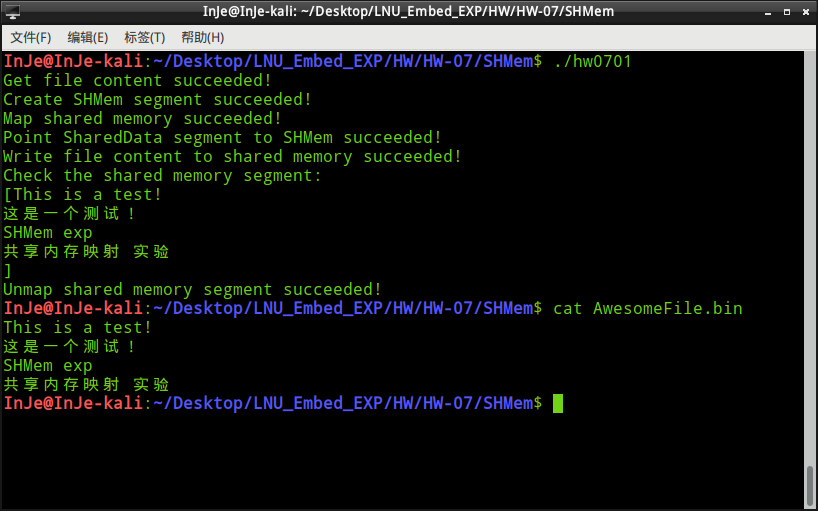
//unmap shared memory from this process

shmdt(ptr\_Mem);

printf("Unmap shared memory segment succeeded!\n");

return 0;

}



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

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#define BufferSize 8192

#define ID\_SHMem 666

int main(void)

{

int IDSHMem;

void \*ptr\_Mem = NULL;

FILE \*pF = NULL;

unsigned char \*ptr\_SharedData = NULL;

unsigned char FileData[BufferSize] = { 0 };

//try to get SHMem

IDSHMem = shmget(ID\_SHMem, BufferSize, 0666 | IPC\_CREAT);

if (IDSHMem == -1)

{

printf("Get SHMem segment failed!\n");

return -1;

}

printf("Get SHMem segment succeeded!\n");

//try to map memory segment

ptr\_Mem = shmat(IDSHMem, 0, 0);

if (ptr\_Mem == (void \*)-1)

{

printf("Map memory failed!\n");

return -1;

}

printf("Map shared memory succeeded!\n");

//point shared data segment to SHMem

ptr\_SharedData = (unsigned char\*)ptr\_Mem;

printf("Point SharedData segment to SHMem succeeded!\n");

//read filedata from SharedData segment

strcpy(FileData, ptr\_SharedData);

printf("Read data from shared memory succeeded!\n");

//test FileData

printf("Check the file data:\n[%s]\n", FileData);

//unmap shared memory from this process

shmdt(ptr\_Mem);

printf("Unmap shared memory segment succeeded!\n");

//delete shared memory

shmctl(IDSHMem, IPC\_RMID, 0);

printf("Delete shared memory segment succeeded!\n");

//write file content

pF = fopen("AwesomeFile\_cpy.bin", "w");

fwrite(FileData, 1, strlen(FileData) + 1, pF);

fclose(pF);

printf("Write file content succeeded!\n");

//test file content

printf("Check the file content:\n");

system("cat AwesomeFile\_cpy.bin");

return 0;

}

