## 《Linux 操作系统设计实践》实验二：进程通信

实验环境：Ubuntu 14.04.5 LTS

实验内容：

运行代码：

umsg.h 利用共享内存和信号量的封装来实现类消息队列的功能

#ifndef \_\_UMSG\_H\_\_

#define \_\_UMSG\_H\_\_

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <string.h>

#define LETTERNUM 8

#define LETTERSIZE 300

#define INT\_MAX 2147483647

struct mailbox {

long type[LETTERNUM];

char text[LETTERNUM][LETTERSIZE];

};

struct letter {

long type;

char text[LETTERSIZE];

};

struct msgid {

int semid;

int shmid;

struct mailbox\* mcont;

};

union semun {

int val;

struct semid\_ds \*buf;

unsigned short \*arry;

};

extern struct msgid umsgget(key\_t ukey, int umsgflg) {

struct msgid ret = { -1, -1, NULL};

union semun rec;

ret.semid = semget(ukey, 3, umsgflg);

if (ret.semid == -1) return ret;

ret.shmid = shmget(ukey, sizeof(struct mailbox), umsgflg);

if (ret.shmid == -1) {

semctl(ret.semid, 0, IPC\_RMID, NULL);

semctl(ret.semid, 1, IPC\_RMID, NULL);

semctl(ret.semid, 2, IPC\_RMID, NULL);

ret.semid = -1;

return ret;

}

void \*gshm = shmat(ret.shmid, NULL, 0);

if (gshm == (void\*) - 1) {

semctl(ret.semid, 0, IPC\_RMID, NULL);

semctl(ret.semid, 1, IPC\_RMID, NULL);

semctl(ret.semid, 2, IPC\_RMID, NULL);

shmctl(ret.shmid, IPC\_RMID, NULL);

ret.semid = -1;

ret.shmid = -1;

return ret;

}

rec.val = LETTERNUM;

semctl(ret.semid, 0, SETVAL, rec);

rec.val = 0;

semctl(ret.semid, 1, SETVAL, rec);

rec.val = 1;

semctl(ret.semid, 2, SETVAL, rec);

ret.mcont = (struct mailbox\*) gshm;

for (int i = 0; i < LETTERNUM; ++i)

ret.mcont->type[i] = -1;

return ret;

}

extern int umsgctl(struct msgid mid, int cmd) {

if (cmd != IPC\_RMID) return -1;

int flag;

flag = semctl(mid.semid, 0, cmd, NULL);

if (flag == -1) return -1;

flag = shmctl(mid.shmid, cmd, NULL);

return flag;

}

extern int umsgsnd(struct msgid mid, const struct letter \* scont) {

int semstat, i;

struct sembuf sp = {0, -1, IPC\_NOWAIT};

struct sembuf rv = {1, +1, IPC\_NOWAIT};

struct sembuf mp = {2, -1, IPC\_NOWAIT};

struct sembuf mv = {2, +1, IPC\_NOWAIT};

semstat = semop(mid.semid, &sp, 1);

if (semstat == -1)return -1;

semstat = semop(mid.semid, &mp, 1);

if (semstat == -1)return -1;

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

if(mid.mcont->type[i] == -1)break;

}

if (i == LETTERNUM)return -1;

mid.mcont->type[i] = scont->type;

strcpy(mid.mcont->text[i], scont->text);

semstat = semop(mid.semid, &mv, 1);

if (semstat == -1)return -1;

semstat = semop(mid.semid, &rv, 1);

return semstat;

}

extern int umsgrcv(struct msgid mid, struct letter \* scont) {

int semstat, i, min = INT\_MAX, mini;

struct sembuf rp = {1, -1, IPC\_NOWAIT};

struct sembuf sv = {0, +1, IPC\_NOWAIT};

struct sembuf mp = {2, -1, IPC\_NOWAIT};

struct sembuf mv = {2, +1, IPC\_NOWAIT};

semstat = semop(mid.semid, &rp, 1);

if(semstat == -1) return -1;

semstat = semop(mid.semid, &mp, 1);

if(semstat == -1) return -1;

for (i = 0, mini = -1; i < LETTERNUM; ++i) {

if ((mid.mcont->type[i] > 0) && (min > mid.mcont->type[i])) {

mini = i;

min = mid.mcont->type[i];

}

}

if (mini == -1) return -1;

scont->type = mid.mcont->type[mini];

strcpy(scont->text, mid.mcont->text[mini]);

mid.mcont->type[mini] = -1;

semstat = semop(mid.semid, &mv, 1);

if (semstat == -1)return -1;

semstat = semop(mid.semid, &sv, 1);

return semstat;

}

#endif

C2uCTL.c 同时包含发送和接收的通信程序，在开始运行时手动输入KEY来建立相互的连接。

#define \_XOPEN\_SOURCE 1

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include "umsg.h"

int main(int argc, char \*argv[])

{

struct msgid msg\_id1, msg\_id2;

int key1, key2, flag;

char inpt[5];

struct letter send, recv;

printf("LOCAL KEY:");

scanf("%d", &key1);

msg\_id1 = umsgget(key1, IPC\_CREAT | 0600);

printf("REMOTE KEY:");

scanf("%d", &key2);

msg\_id2 = umsgget(key2, IPC\_CREAT | 0600);

printf("MSGID1A:%d,MSGID2A:%d\n", msg\_id1.semid, msg\_id2.semid);

printf("MSGID1B:%d,MSGID2B:%d\n", msg\_id1.shmid, msg\_id2.shmid);

printf("OPERATE (H:HELP E:EXIT)\n");

for(int loop = 1; inpt[0] != 'E'; ++loop) {

printf("INPUT :");

scanf("%s", inpt);

switch(inpt[0]) {

case 'S':case 's':

send.type = loop;

scanf("%s", send.text);

flag = umsgsnd(msg\_id2, &send);

if(flag != -1)printf("SEND#%05ld %-32s [ O K ]\n", send.type, send.text);

else printf("SEND#%05ld %-32s [ FAILED ]\n", send.type, send.text);

break;

case 'R':case 'r':

flag = umsgrcv(msg\_id1, &recv);

if(flag != -1)printf("RECV#%05ld %-32s [ O K ]\n", recv.type, recv.text);

else printf("RECV#ERROR %-32s [ FAILED ]\n", " ");

break;

case 'E':case 'e':

inpt[0] = 'E';

break;

default:

printf("S:Send R:Receive E:Exit\n");

}

}

umsgctl(msg\_id1, IPC\_RMID);

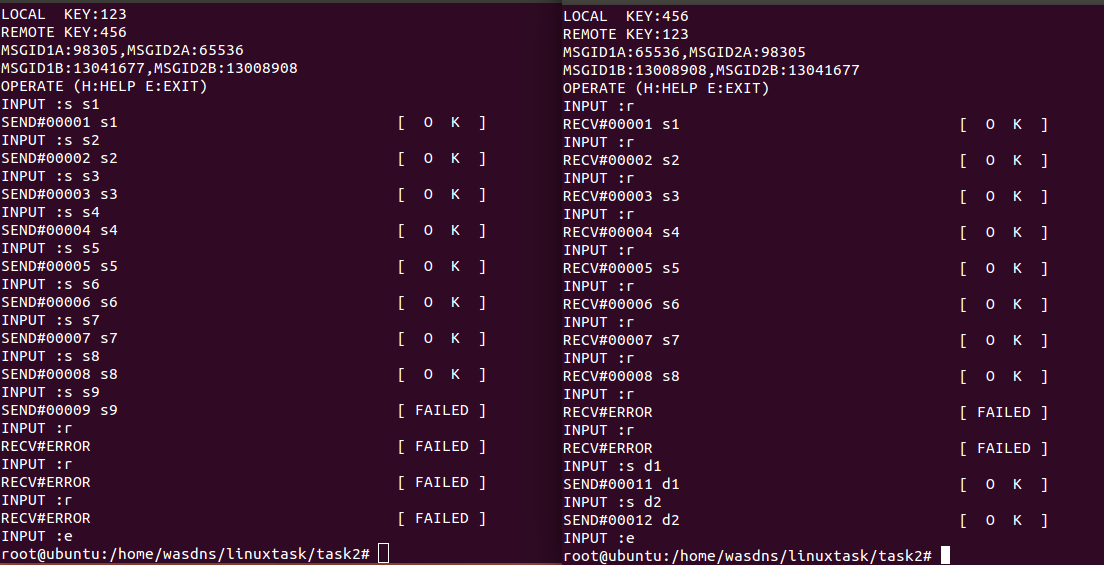
umsgctl(msg\_id2, IPC\_RMID);

return EXIT\_SUCCESS;

}

运行结果：

预留空间为每个进程各8封信，当空间满时发送失败，须等待对方接收才能继续发送。



实验总结：

利用信号量和共享内存实现通信较为复杂，可以参考消息队列的函数所实现的功能来编写。