Lab3源代码：

Lab3-1.c:

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(void)

{

pthread\_mutex\_t mutex;

pthread\_mutex\_init(&mutex,NULL);

printf("I'm father,PID is %d.\n",getpid());

pid\_t pid1 = fork();

if(pid1 == 0)

{

printf("I am the process P1. PID is %d.\n",getpid());

return 0;

}

waitpid(pid1, NULL, 0);

pid\_t pid2 = fork();

if(pid2 == 0)

{

pthread\_mutex\_lock(&mutex);

printf("I am the process P2. PID is %d.\n",getpid());

pthread\_mutex\_unlock(&mutex);

return 0;

}

waitpid(pid1, NULL, 0);

pid\_t pid3 = fork();

if(pid3 == 0)

{

pthread\_mutex\_lock(&mutex);

printf("I am the process P3. PID is %d.\n",getpid());

pthread\_mutex\_unlock(&mutex);

return 0;

}

waitpid(pid2, NULL, 0);

waitpid(pid3, NULL, 0);

pid\_t pid4 = fork();

if(pid4 == 0)

{

printf("I am the process P4. PID is %d.\n",getpid());

return 0;

}

else if(pid1 != -1)

{

waitpid(pid1, NULL, 0);

waitpid(pid2, NULL, 0);

waitpid(pid3, NULL, 0);

waitpid(pid4, NULL, 0);

printf("Process exited.\n");

}

pthread\_mutex\_destroy(&mutex);

return 0;

}

Lab3-2.c:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <unistd.h>

#include <assert.h>

int ticketcount = 1000; //共享变量，表示票的总数

int temp = 0;

sem\_t blanks; //信号量1，表示票余量

sem\_t datas; //信号量2，表示售票量

void \*worker1(void \*arg){

while(1){

sem\_wait(&blanks); //判断是否还有票

temp = ticketcount;

pthread\_yield(); //诱发并发错误，但是看不出来是否有效

temp = temp - 1;

pthread\_yield();

ticketcount = temp;

printf("Sale:ticketCount now is: %d\n",ticketcount);

usleep(500); //当一个进程执行完一次操作后，睡眠500微秒

sem\_post(&datas); //执行完一次操作后，售票数+1

}

return NULL;

}

void \*worker2(void \*arg){

while(1){

sem\_wait(&datas); //判断是否有退票

temp = ticketcount;

pthread\_yield();

temp = temp + 1;

pthread\_yield();

ticketcount = temp;

printf("Refund:ticketCount now is: %d\n",ticketcount);

usleep(500);

sem\_post(&blanks); //完成一次退票操作后，票余量+1

}

return NULL;

}

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

sem\_init(&blanks, 0, 1000);

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

pthread\_t p1, p2;

pthread\_create(&p1, NULL, worker1, NULL);

pthread\_create(&p2, NULL, worker2, NULL);

pthread\_join(p1, NULL);

pthread\_join(p2, NULL);

sem\_destroy(&blanks);

sem\_destroy(&datas);

return 0;

}

Lab3-3.c:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <unistd.h>

#include <assert.h>

char buf[10] = {0};

sem\_t blanks; //信号量1，表示buf缓存区中空闲单元的个数

sem\_t datas; //信号量2，表示buf缓存区中非空单元的个数

void \*worker1(void \*arg){

for (int i = 0; i < 10;) {

sem\_wait(&blanks); //判断是否有空闲单元可供输入

scanf("%c",&buf[i]);

i++;

i %= 10; //模拟环形队列，以防i++超过10导致数据丢失

sem\_post(&datas); //非空单元数+1

}

return NULL;

}

void \*worker2(void \*arg){

for (int i = 0; i < 10;) {

sem\_wait(&datas); //判断buf中是否有数据

printf("%c ",buf[i]);

sleep(1); //每次输出睡眠1s

i++;

i %= 10;

sem\_post(&blanks); //输出一个数据后，空闲单元数+1

}

return NULL;

}

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

sem\_init(&blanks, 0, 10);//在多线程之前初始化信号量

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

pthread\_t p1, p2;

pthread\_create(&p1, NULL, worker1, NULL); //建立多线程

pthread\_create(&p2, NULL, worker2, NULL);

pthread\_join(p1, NULL);

pthread\_join(p2, NULL);

sem\_destroy(&blanks);

sem\_destroy(&datas);

return 0;

}

Lab3-4-1(sender.c)

/\*

\* Filename: Sender.c

\* Description:

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/sem.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <string.h>

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

{

key\_t key;

int shm\_id;

int sem\_id;

int value = 0;

//1.Product the key

key = ftok(".", 0xFF);

//2. Creat semaphore for visit the shared memory

sem\_id = semget(key, 1, IPC\_CREAT|0644);

if(-1 == sem\_id)

{

perror("semget");

exit(EXIT\_FAILURE);

}

//3. init the semaphore, sem=0

if(-1 == (semctl(sem\_id, 0, SETVAL, value)))

{

perror("semctl");

exit(EXIT\_FAILURE);

}

//4. Creat the shared memory(1K bytes)

shm\_id = shmget(key, 1024, IPC\_CREAT|0644);

if(-1 == shm\_id)

{

perror("shmget");

exit(EXIT\_FAILURE);

}

//5. attach the shm\_id to this process

char \*shm\_ptr;

shm\_ptr = shmat(shm\_id, NULL, 0);

if(NULL == shm\_ptr)

{

perror("shmat");

exit(EXIT\_FAILURE);

}

//6. Operation procedure

struct sembuf sem\_b;

sem\_b.sem\_num = 0; //first sem(index=0)

sem\_b.sem\_flg = SEM\_UNDO;

sem\_b.sem\_op = 1; //Increase 1,make sem=1

while(1)

{

//if(0 == (value = semctl(sem\_id, 0, GETVAL)))

{

printf("\nNow, snd message process running:\n");

printf("\tInput the snd message: ");

scanf("%s", shm\_ptr);

if(-1 == semop(sem\_id, &sem\_b, 1))

{

perror("semop");

exit(EXIT\_FAILURE);

}

}

//if enter "end", then end the process

if(0 == (strcmp(shm\_ptr ,"end")))

{

printf("\nExit sender process now!\n");

break;

}

}

shmdt(shm\_ptr);

return 0;

}

Lab3-4-1(reciever.c)

/\*

\* Filename: Receiver.c

\* Description:

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/sem.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <string.h>

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

{

key\_t key;

int shm\_id;

int sem\_id;

int value = 0;

//1.Product the key

key = ftok(".", 0xFF);

//2. Creat semaphore for visit the shared memory

sem\_id = semget(key, 1, IPC\_CREAT|0644);

if(-1 == sem\_id)

{

perror("semget");

exit(EXIT\_FAILURE);

}

//3. init the semaphore, sem=0

if(-1 == (semctl(sem\_id, 0, SETVAL, value)))

{

perror("semctl");

exit(EXIT\_FAILURE);

}

//4. Creat the shared memory(1K bytes)

shm\_id = shmget(key, 1024, IPC\_CREAT|0644);

if(-1 == shm\_id)

{

perror("shmget");

exit(EXIT\_FAILURE);

}

//5. attach the shm\_id to this process

char \*shm\_ptr;

shm\_ptr = shmat(shm\_id, NULL, 0);

if(NULL == shm\_ptr)

{

perror("shmat");

exit(EXIT\_FAILURE);

}

//6. Operation procedure

struct sembuf sem\_b;

sem\_b.sem\_num = 0; //first sem(index=0)

sem\_b.sem\_flg = SEM\_UNDO;

sem\_b.sem\_op = -1; //Increase 1,make sem=1

while(1)

{

//if(1 == (value = semctl(sem\_id, 0, GETVAL)))

{

printf("\nNow, receive message process running:\n");

printf("\tThe message is : %s\n", shm\_ptr);

if(-1 == semop(sem\_id, &sem\_b, 1))

{

perror("semop");

exit(EXIT\_FAILURE);

}

}

//if enter "end", then end the process

if(0 == (strcmp(shm\_ptr ,"end")))

{

printf("\nExit the receiver process now!\n");

break;

}

}

shmdt(shm\_ptr);

//7. delete the shared memory

if(-1 == shmctl(shm\_id, IPC\_RMID, NULL))

{

perror("shmctl");

exit(EXIT\_FAILURE);

}

//8. delete the semaphore

if(-1 == semctl(sem\_id, 0, IPC\_RMID))

{

perror("semctl");

exit(EXIT\_FAILURE);

}

return 0;

}

Lab3-4-3(client.c)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include <signal.h>

#define BUF\_SIZE 128

//Rebuild the strcut (must be)

struct msgbuf

{

long mtype;

char mtext[BUF\_SIZE];

};

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

{

//1. creat a mseg queue

key\_t key;

int msgId;

printf("THe process(%s),pid=%d started~\n", argv[0], getpid());

key = ftok(".", 0xFF);

msgId = msgget(key, IPC\_CREAT|0644);

if(-1 == msgId)

{

perror("msgget");

exit(EXIT\_FAILURE);

}

//2. creat a sub process, wait the server message

pid\_t pid;

if(-1 == (pid = fork()))

{

perror("vfork");

exit(EXIT\_FAILURE);

}

//In child process

if(0 == pid)

{

while(1)

{

alarm(0);

alarm(100); //if doesn't receive messge in 100s, timeout & exit

struct msgbuf rcvBuf;

memset(&rcvBuf, '\0', sizeof(struct msgbuf));

msgrcv(msgId, &rcvBuf, BUF\_SIZE, 2, 0);

printf("Server said: %s\n", rcvBuf.mtext);

}

exit(EXIT\_SUCCESS);

}

else //parent process

{

while(1)

{

usleep(100);

struct msgbuf sndBuf;

memset(&sndBuf, '\0', sizeof(sndBuf));

char buf[BUF\_SIZE] ;

memset(buf, '\0', sizeof(buf));

printf("\nInput snd mesg: ");

scanf("%s", buf);

strncpy(sndBuf.mtext, buf, strlen(buf)+1);

sndBuf.mtype = 1;

if(-1 == msgsnd(msgId, &sndBuf, strlen(buf)+1, 0))

{

perror("msgsnd");

exit(EXIT\_FAILURE);

}

//if scanf "end~", exit

if(!strcmp("end~", buf))

break;

}

printf("THe process(%s),pid=%d exit~\n", argv[0], getpid());

}

return 0;

}

Lab3-4-3(server.c)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include <signal.h>

#define BUF\_SIZE 128

//Rebuild the strcut (must be)

struct msgbuf

{

long mtype;

char mtext[BUF\_SIZE];

};

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

{

//1. creat a mseg queue

key\_t key;

int msgId;

key = ftok(".", 0xFF);

msgId = msgget(key, IPC\_CREAT|0644);

if(-1 == msgId)

{

perror("msgget");

exit(EXIT\_FAILURE);

}

printf("Process (%s) is started, pid=%d\n", argv[0], getpid());

while(1)

{

alarm(0);

alarm(600); //if doesn't receive messge in 600s, timeout & exit

struct msgbuf rcvBuf;

memset(&rcvBuf, '\0', sizeof(struct msgbuf));

msgrcv(msgId, &rcvBuf, BUF\_SIZE, 1, 0);

printf("Receive msg: %s\n", rcvBuf.mtext);

struct msgbuf sndBuf;

memset(&sndBuf, '\0', sizeof(sndBuf));

strncpy((sndBuf.mtext), (rcvBuf.mtext), strlen(rcvBuf.mtext)+1);

sndBuf.mtype = 2;

if(-1 == msgsnd(msgId, &sndBuf, strlen(rcvBuf.mtext)+1, 0))

{

perror("msgsnd");

exit(EXIT\_FAILURE);

}

//if scanf "end~", exit

if(!strcmp("end~", rcvBuf.mtext))

break;

}

printf("THe process(%s),pid=%d exit~\n", argv[0], getpid());

return 0;

}