**操作系统大作业（二）**

**进程间通信**

有三个进程，分别完成如下任务

（1）接收键盘输入进程：负责接收用户键盘输入，并以适当的方式将数据交给其他进程处理。

（2）显示进程：负责全部数据显示任务，包括键盘输入显示和提示信息显示。

（3）分发数据进程：将键盘数据分为三类，字母、数字和其他字符，分别写入文件letter.txt，number.txt，其他数据丢弃。

进程1接收到输入后，将数据发送给进程3，进程3对齐进行处理，然后把要显示的数据发送给进程2，由进程2进行显示。

**1、使用消息队列方式**

**// msg.h头文件**

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/msg.h>

#include <errno.h>

// 消息结构体

#define MAX\_TEXT 128

struct msg\_st

{

long int msg\_type;

char text[BUFSIZ];

};

// 创建消息

int init(int id)

{

int r = msgget((key\_t) id, 0666 | IPC\_CREAT);

if(r == -1) {

fprintf(stderr, "msgget failed: %d\n", errno);

exit(EXIT\_FAILURE);

}

return r;

}

// 发送消息

int send(int id, void \*buf)

{

int r = msgsnd(id, buf, MAX\_TEXT, 0);

if(r == -1) {

fprintf(stderr, "msgsnd failed\n");

exit(EXIT\_FAILURE);

}

return r;

}

// 读取消息

int rev(int id, long int msgtype, void \*buf)

{

int r = msgrcv(id, buf, BUFSIZ, msgtype, 0);

if(r == -1) {

fprintf(stderr, "msgrcv failed: %d\n", errno);

exit(EXIT\_FAILURE);

}

return r;

}

// 删除消息

int del(int id)

{

int r = msgctl(id, IPC\_RMID, 0);

if(r == -1) {

fprintf(stderr, "msgctl(IPC\_RMID) failed\n");

exit(EXIT\_FAILURE);

}

return r;

}

**// input.c**

#include "msg.h"

int main()

{

struct msg\_st data;

char buffer[BUFSIZ];

int id = init(4567);

printf("input: --------start--------\n");

while(1) {

// 输入数据

printf("Enter some text: ");

fgets(buffer, BUFSIZ, stdin);

// 创建消息

data.msg\_type = 1;

strcpy(data.text, buffer);

// 发送消息

send(id, (void\*)&data);

if(strncmp(buffer, "quit", 4) == 0)

break;

sleep(1);

}

printf("receive: --------stop--------\n");

return 0;

}

**// data.c**

#include "msg.h"

int main()

{

struct msg\_st data;

char buffer[BUFSIZ]; char \*p;

int id = init(4567);

printf("data: --------start--------\n");

while(1) {

// 接收消息

rev(id, 1, (void\*)&data);

strcpy(buffer, data.text);

// test

printf("data: %s\n", buffer);

// quit

if(strncmp(buffer, "quit", 3) == 0) {

data.msg\_type = 2; // 如果是quit指令,直接转发给显示进程

send(id, (void \*)&data);

break;

}

// 分类

p = buffer;

while(\*p) {

if(\*p >= '0' && \*p <= '9') {

sprintf(data.text, "get num: %c\n", \*p);

} else if((\*p >= 'a' && \*p <= 'z') || (\*p >= 'A' && \*p <= 'Z')) {

sprintf(data.text, "get letter: %c\n", \*p);

} else {

sprintf(data.text, "throw char: %c\n", \*p);

}

//printf("data \_ sendmsg: %s", data.text);

data.msg\_type = 2;

send(id, (void \*)&data);

usleep(10000);

++p;

}

}

printf("data: --------stop--------\n");

return 0;

}

**// disp.c**

#include "msg.h"

int main()

{

struct msg\_st data;

char buffer[BUFSIZ];

int id = init(4567);

printf("disp: --------start--------\n");

while(1) {

// 接收消息

rev(id, 2, (void\*)&data);

strcpy(buffer, data.text);

// 显示

printf("disp: %s", buffer);

// quit

if(strncmp(buffer, "quit", 3) == 0)

break;

}

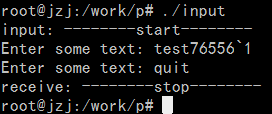
del(id);

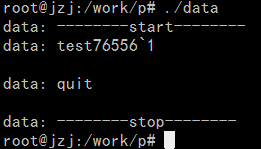
printf("disp: --------stop--------\n");

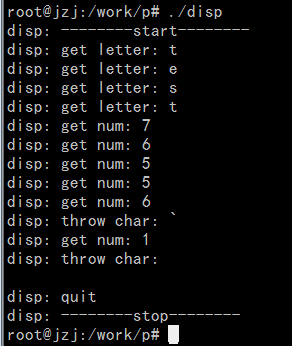
return 0;

}

通过不同的shell终端，同时执行三个进程，执行结果如下：







**2、使用内存共享方式**

**// mem.h头文件**

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/shm.h>

#include <errno.h>

// 共享内存结构体

#define TEXT\_SZ 128

#define MEM\_W 0

#define MEM\_R 1

struct mem\_st

{

int flag; // 标志，1：表示可读取，0表示可写入

char text[TEXT\_SZ]; // 保存写入和读取的文本

};

// 创建内存共享

int init(int id)

{

int shmid; // 共享内存标识符

// 创建共享内存

shmid = shmget((key\_t) id, sizeof(struct mem\_st), 0666|IPC\_CREAT);

if(shmid == -1) {

fprintf(stderr, "shmget failed\n");

exit(EXIT\_FAILURE);

}

return shmid;

}

// 连接到当前进程的地址空间

void\* mat(int id)

{

void \*shm = NULL; // 分配的共享内存的原始首地址

// 将共享内存连接到当前进程的地址空间

shm = shmat(id, (void\*)0, 0);

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

fprintf(stderr, "shmat failed\n");

exit(EXIT\_FAILURE);

}

return shm;

}

// 阻塞式写入

int send(struct mem\_st\* shared, void \*buf)

{

// 数据还没有被读取，则等待数据被读取,不能向共享内存中写入文本

while(shared->flag == MEM\_R) {

usleep(10000);

// printf("Waiting read finish...\n");

}

// 向共享内存中写入数据

strncpy(shared->text, buf, TEXT\_SZ);

// 写完数据，设置flag使共享内存段可读

shared->flag = MEM\_R;

return 0;

}

// 阻塞式读取

int rev(struct mem\_st\* shared, void \*buf)

{

// 有其他进程在写数据，不能读取数据

while(shared->flag == MEM\_W) {

usleep(10000);

// printf("Waiting read finish...\n");

}

// 没有进程向共享内存定数据有数据可读取

strcpy(buf, shared->text);

// sleep(rand() % 3);

// 读取完数据，设置flag使共享内存段可写

shared->flag = MEM\_W;

return 0;

}

// 分离

int release(struct mem\_st\* shared)

{

//把共享内存从当前进程中分离

int r = shmdt((void \*) shared);

if(r == -1) {

fprintf(stderr, "shmdt failed\n");

exit(EXIT\_FAILURE);

}

return r;

}

// 删除

int del(int id)

{

//删除共享内存

int r = shmctl(id, IPC\_RMID, 0);

if(r == -1) {

fprintf(stderr, "shmctl(IPC\_RMID) failed\n");

exit(EXIT\_FAILURE);

}

return r;

}

**// input.c**

#include "mem.h"

int main()

{

char buffer[BUFSIZ];

struct mem\_st \*data = init(1234, 0);

printf("input: --------start--------\n");

while(1) {

// 输入数据

printf("Enter some text: ");

fgets(buffer, BUFSIZ, stdin);

// 发送消息

send(data, buffer);

if(strncmp(buffer, "quit", 4) == 0)

break;

sleep(1);

}

release(data);

printf("receive: --------stop--------\n");

return 0;

}

**// data.c**

#include "mem.h"

int main()

{

char buffer[BUFSIZ]; char \*p;

char buffer2[BUFSIZ];

int id;

struct mem\_st \*data, \*data2;

data = init(1234, &id);

data2 = init(2345, 0);

printf("data: --------start--------\n");

while(1) {

// 接收消息

rev(data, buffer);

// test

printf("data: %s\n", buffer);

// quit

if(strncmp(buffer, "quit", 3) == 0) { // 如果是quit指令,直接转发给显示进程

send(data2, buffer);

break;

}

// 分类

p = buffer;

while(\*p) {

if(\*p >= '0' && \*p <= '9') {

sprintf(buffer2, "get num: %c\n", \*p);

} else if((\*p >= 'a' && \*p <= 'z') || (\*p >= 'A' && \*p <= 'Z')) {

sprintf(buffer2, "get letter: %c\n", \*p);

} else {

sprintf(buffer2, "throw char: %c\n", \*p);

}

send(data2, buffer2);

usleep(10000);

++p;

}

}

release(data);

release(data2);

del(id);

printf("data: --------stop--------\n");

return 0;

}

**// disp.c**

#include "mem.h"

int main()

{

char buffer2[BUFSIZ];

int id;

struct mem\_st \*data2 = init(2345, &id);

printf("disp: --------start--------\n");

while(1) {

// 接收消息

rev(data2, buffer2);

strcpy(buffer2, data2->text);

// 显示

printf("disp: %s", buffer2);

// quit

if(strncmp(buffer2, "quit", 3) == 0)

break;

}

release(data2);

del(id);

printf("disp: --------stop--------\n");

return 0;

}

执行结果如下：

