**操作系统大作业（三）**

**一、题目要求**

**目标：**

本次实验的目标是在Linux环境下实现一个多线程对临界资源的互斥操作，利用信号量实现对临界资源的保护，支持Linux下文件输入输出，提高对Linux环境下多进程、多线程、信号量机制和文件操作等知识的理解。

**问题描述：**

设有进程A、B、C，分别调用过程get、copy、put对缓冲区S和T进行操作。其中get负责从文件F\_in中把数据块读入并输出缓冲区S，copy负责从S中提取数据块并复制到缓冲区T中，put负责从缓冲区T中取出信息存入到本地文件F\_out中。实现get、copy、put的操作过程。

**功能需求：**

提供与用户交互的界面，用户可指定输入、输出文件以及缓冲区大小

利用信号量实现互斥

同时产生多个get、copy和put线程，利用信号量实现多个相同功能的线程间的通信，避免临界资源的非法访问，可参考读写者问题解决方案

支持文件输入、输出

**非功能需求：**

程序应有较好的容错性（即能对用户输入的命令进行判断，并对错误的命令进行错误处理）

**过程需求：**

使用vi进行代码的编写

使用make工具建立工程

将实现不同类别功能的函数写到不同的.c文件中，并使用makefile链接编译。

**二、程序实现**

**结构设计**

三个信号量实现线程互斥

sem\_tread\_sem; // read

sem\_twrite\_sem; // write

sem\_tcopy\_sem; // copy

**功能设计**

实现了同步多线程运行，文件读入，缓冲区COPY，文件写入三个线程同时运行，完成LINUX下文件的输入输出。

文件读入：用linux下open(),read(),lseek()函数实现

文件写出：用linux下open(),write()实现

缓冲区建立 malloc()函数

信号量建立sem\_init(),sem\_wait(),sem\_post()

互斥量建立pthread\_create(),pthread\_mutex\_lock(),pthread\_mutex\_unlock()

**测试**

使用./signalmove启动程序，启动参数必须包括，写入文件名，写出的文件名，和缓冲区大小，否则会报错退出。

缓冲区大小必须为数字，否则退出。

**程序源码**

**/\***

**\* signalmove.h**

**\*/**

#ifndef SIGNALMOVE\_H\_

#define SIGNALMOVE\_H\_

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <signal.h>

#include <fcntl.h>

#include <string.h>

#include <ctype.h>

//file works

void file\_in();

void file\_out();

//copy from s to t

void copy();

//put and get

void put();

void get();

sem\_t read\_sem; // read

sem\_t write\_sem; // write

sem\_t copy\_sem; // copy

pthread\_mutex\_t read\_mutex; // pthread mutex for read

pthread\_mutex\_t write\_mutex; // pthread mutex for write

char \*data\_s; //read buffer

char \*data\_t; //write buffer

int len; //buffer length

char \*filename\_out; //name\_out

char \*filename\_in; //name\_in

int result;

int result\_copy;

#endif /\* SIGNALMOVE\_H\_ \*/

**/\***

**\* main.c**

**\*/**

#include "signalmove.h"

int isdigit\_all(char \*str)

{

while (\*str != '\0')

{

if (!isdigit(\*str++))

return 0;

}

return 1;

}

//signal out

void Handlesignal(int signo) {

printf("end program!\n", signo);

exit(0);

}

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

if(argc!=4){

printf("error input! 1.filename\_in,2.filename\_out,3,length of buffer\n");

exit(0);

}

if(isdigit\_all(argv[3])==0){

printf("error input! 1.filename\_in,2.filename\_out,3,length of buffer\n");

printf("error length of buffer must be digit!\n");

exit(0);

}

result\_copy = 0;

filename\_in=argv[1];

filename\_out=argv[2];

len=atoi(argv[3]);

int thread\_i = 0;

data\_s = (char \*) malloc(len \* sizeof(char));

data\_t = (char \*) malloc(len \* sizeof(char));

if (signal(SIGINT, Handlesignal) == SIG\_ERR) {

printf("error signal out\n");

}

pthread\_t read\_pthread;

pthread\_t copy\_pthread;

pthread\_t write\_pthread;

//sem lock

int ini1 = sem\_init(&read\_sem, 0, 1);

int ini2 = sem\_init(&write\_sem, 0, 0);

int ini5 = sem\_init(?\_sem, 0, 0);

if (ini1 && ini2 && ini5 != 0) {

printf("sem error\n");

exit(1);

}

//mutex lock

int ini3 = pthread\_mutex\_init(&read\_mutex, NULL);

int ini4 = pthread\_mutex\_init(&write\_mutex, NULL);

if (ini3 && ini4 != 0) {

printf("mutex error\ n");

exit(1);

}

int thread\_1 = pthread\_create(&read\_pthread, NULL, (void \*)&get,

(void \*) (&thread\_i));

if (thread\_1 != 0) {

printf("read thread create error!\n");

exit(1);

}

thread\_i++;

int thread\_2 = pthread\_create(?\_pthread, NULL, (void \*)?,

(void \*) (&thread\_i));

if (thread\_2 != 0) {

printf("copy thread create error!\n");

exit(1);

}

thread\_i++;

int thread\_3 = pthread\_create(&write\_pthread, NULL, (void \*)&put,

(void \*) (&thread\_i));

if (thread\_3 != 0) {

printf("write thread create error!\n");

exit(1);

}

pthread\_join(read\_pthread, NULL);

pthread\_join(copy\_pthread, NULL);

pthread\_join(write\_pthread, NULL);

free(data\_s);

free(data\_t);

exit(0);

}

**/\***

**\* file.c**

**\*/**

#include "signalmove.h"

void file\_in(int fd, char \*data, int len) {

while (1) {

sem\_wait(&read\_sem);

pthread\_mutex\_lock(&read\_mutex);

printf("data\_in..........\n");

if (lseek(fd, 0, SEEK\_CUR) == -1) {

printf("lseek failed!\n");

}

if ((result = read(fd, data, len)) == -1) {

printf("read error! \n");

}

int i=0,j=0;

printf("data\_in..........complete\n");

pthread\_mutex\_unlock(&read\_mutex);

sem\_post(?\_sem);

if(result<len){

len=result;

result=0;

break;

}

}

}

void file\_out(int fd, char \*data, int len) {

while (1) {

sem\_wait(&write\_sem);

pthread\_mutex\_lock(&write\_mutex);

printf("data\_out..........\n");

if (write(fd, data, len) == -1) {

printf("write error! \n");

}

printf("data\_out..........complete\n");

pthread\_mutex\_unlock(&write\_mutex);

if(result\_copy==1){

break;

}

}

}

**/\***

**\* put.c**

**\*/**

#include "signalmove.h"

void put() {

int fd = open(filename\_out, O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

printf("open file wrong!\n");

exit(0);

}

file\_out(fd, data\_t, len);

}

**/\***

**\* get.c**

**\*/**

#include "signalmove.h"

void get() {

int fd = open(filename\_in, O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

printf("open file wrong!\n");

exit(0);

}

file\_in(fd, data\_s, len);

}

**/\***

**\* copy.c**

**\*/**

#include "signalmove.h"

void copy() {

while (1) {

sem\_wait(?\_sem);

pthread\_mutex\_lock(&read\_mutex);

pthread\_mutex\_lock(&write\_mutex);

printf("copy..........\n");

strcpy(data\_t,data\_s);

int i=0;

for(;i<len;i++){

data\_s[i]=' ';

}

printf("copy..........complete\n");

pthread\_mutex\_unlock(&write\_mutex);

pthread\_mutex\_unlock(&read\_mutex);

sem\_post(&write\_sem);

sem\_post(&read\_sem);

if (result == 0) {

result\_copy = 1;

break;

}

}

}