## 实验8 多线程“生产者消费者”实验

学生姓名： 李亚军 学 号： 6100412196

专业班级： 卓越计科121班

#### 1．实验目的

“生产者消费者”问题是一个著名的同时性编程问题的集合。通过学习经典的“生产者消费者”问题的实验，读者可以进一步熟悉Linux中的多线程编程，并且掌握用信号量处理线程间的同步和互斥问题。

#### 2．实验内容

“生产者—消费者”问题描述如下。

有一个有限缓冲区和两个线程：生产者和消费者。他们分别不停地把产品放入缓冲区和从缓冲区中拿走产品。一个生产者在缓冲区满的时候必须等待，一个消费者在缓冲区空的时候也必须等待。另外，因为缓冲区是临界资源，所以生产者和消费者之间必须互斥执行。它们之间的关系如图1所示。



图1 生产者消费者问题描述

这里要求使用有名管道（将在下一章介绍，可提前了解）来模拟有限缓冲区，并且使用信号量来解决“生产者—消费者”问题中的同步和互斥问题。

#### 3．实验步骤

（1）信号量的考虑。

这里使用3个信号量，其中两个信号量avail和full分别用于解决生产者和消费者线程之间的同步问题，mutex是用于这两个线程之间的互斥问题。其中avail表示有界缓冲区中的空单元数，初始值为N；full表示有界缓冲区中非空单元数，初始值为0；mutex是互斥信号量，初始值为1。

（2）画出流程图。

本实验流程图如图2所示。



图2 “生产者—消费者”实验流程图

（3）编写代码

本实验的代码中采用的有界缓冲区拥有3个单元，每个单元为5个字节。为了尽量体现每个信号量的意义，在程序中生产过程和消费过程是随机（采取0～5s的随机时间间隔）进行的，而且生产者的速度比消费者的速度平均快两倍左右（这种关系可以相反）。生产者一次生产一个单元的产品（放入“hello”字符串），消费者一次消费一个单元的产品。

/\*producer-customer.c\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <pthread.h>

#include <errno.h>

#include <semaphore.h>

#include <sys/ipc.h>

#define MYFIFO "myfifo" /\* 缓冲区有名管道的名字 \*/

#define BUFFER\_SIZE 3 /\* 缓冲区的单元数 \*/

#define UNIT\_SIZE 5 /\* 每个单元的大小 \*/

#define RUN\_TIME 30 /\* 运行时间 \*/

#define DELAY\_TIME\_LEVELS 5.0 /\* 周期的最大值 \*/

int fd;

time\_t end\_time;

sem\_t mutex, full, avail; /\* 3个信号量 \*/

/\*生产者线程\*/

void \*producer(void \*arg)

{

int real\_write;

int delay\_time = 0;

while(time(NULL) < end\_time)

{

delay\_time = (int)(rand() \* DELAY\_TIME\_LEVELS/(RAND\_MAX) / 2.0) + 1;

sleep(delay\_time);

/\*P操作信号量avail和mutex\*/

sem\_wait(&avail);

sem\_wait(&mutex);

printf("\nProducer: delay = %d\n", delay\_time);

/\*生产者写入数据\*/

if ((real\_write = write(fd, "hello", UNIT\_SIZE)) == -1)

{

if(errno == EAGAIN)

{

printf("The FIFO has not been read yet.Please try later\n");

}

}

else

{

printf("Write %d to the FIFO\n", real\_write);

}

/\*V操作信号量full和mutex\*/

sem\_post(&full);

sem\_post(&mutex);

}

pthread\_exit(NULL);

}

/\* 消费者线程\*/

void \*customer(void \*arg)

{

unsigned char read\_buffer[UNIT\_SIZE];

int real\_read;

int delay\_time;

while(time(NULL) < end\_time)

{

delay\_time = (int)(rand() \* DELAY\_TIME\_LEVELS/(RAND\_MAX)) + 1;

sleep(delay\_time);

/\*P操作信号量full和mutex\*/

sem\_wait(&full);

sem\_wait(&mutex);

memset(read\_buffer, 0, UNIT\_SIZE);

printf("\nCustomer: delay = %d\n", delay\_time);

if ((real\_read = read(fd, read\_buffer, UNIT\_SIZE)) == -1)

{

if (errno == EAGAIN)

{

printf("No data yet\n");

}

}

printf("Read %s from FIFO\n", read\_buffer);

/\*V操作信号量avail和mutex\*/

sem\_post(&avail);

sem\_post(&mutex);

}

pthread\_exit(NULL);

}

int main()

{

pthread\_t thrd\_prd\_id,thrd\_cst\_id;

pthread\_t mon\_th\_id;

int ret;

srand(time(NULL));

end\_time = time(NULL) + RUN\_TIME;

/\*创建有名管道\*/

if((mkfifo(MYFIFO, O\_CREAT|O\_EXCL) < 0) && (errno != EEXIST))

{

printf("Cannot create fifo\n");

return errno;

}

/\*打开管道\*/

fd = open(MYFIFO, O\_RDWR);

if (fd == -1)

{

printf("Open fifo error\n");

return fd;

}

/\*初始化互斥信号量为1\*/

ret = sem\_init(&mutex, 0, 1);

/\*初始化avail信号量为N\*/

ret += sem\_init(&avail, 0, BUFFER\_SIZE);

/\*初始化full信号量为0\*/

ret += sem\_init(&full, 0, 0);

if (ret != 0)

{

printf("Any semaphore initialization failed\n");

return ret;

}

/\*创建两个线程\*/

ret = pthread\_create(&thrd\_prd\_id, NULL, producer, NULL);

if (ret != 0)

{

printf("Create producer thread error\n");

return ret;

}

ret = pthread\_create(&thrd\_cst\_id, NULL, customer, NULL);

if(ret != 0)

{

printf("Create customer thread error\n");

return ret;

}

pthread\_join(thrd\_prd\_id, NULL);

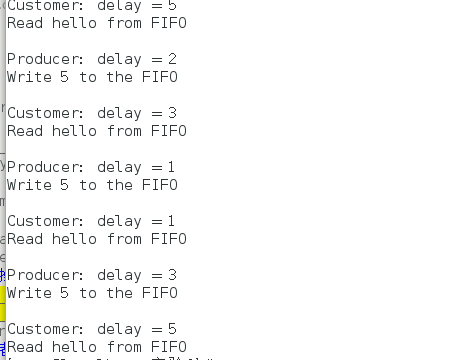
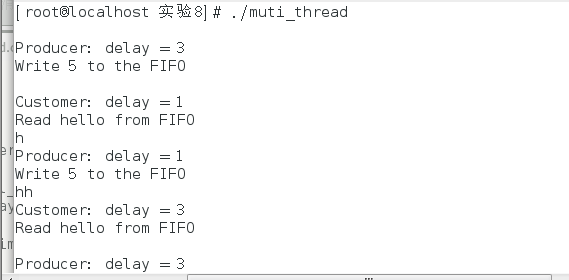
pthread\_join(thrd\_cst\_id, NULL);

close(fd);

unlink(MYFIFO);

return 0;

}



#### 根据实验结果完成实验报告，写出实验总结。

整个实验比较简单，但是开始老是在编译的时候出现sem\_wait等未定义的错误，经上网查询得知c库没有使能semaphore的支持.，故而需要添加-lpthread参数。