一、预备知识

* 有 C 语言基础
* 掌握在 Linux 下常用编辑器的使用
* 掌握 Linux 下的程序编译过程
* 学习 pthread 库函数的使用
* 掌握共享锁和信号量 的使用方法
* 掌握make和makefile工具编译程序

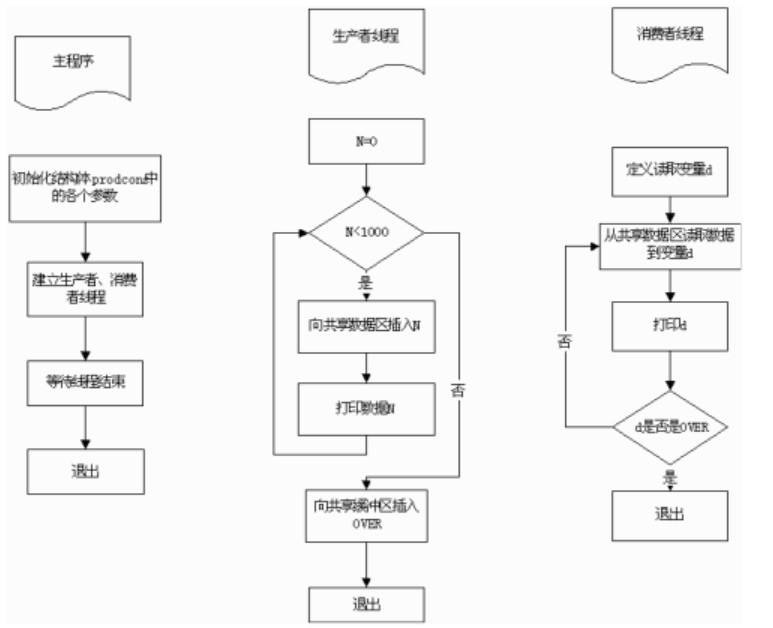
二、实验设备及工具

硬件： PC 机 Pentium 500 以上, 硬盘 40G 以上, 内存大于 128M。

软件：PC 机操作系统 REDHAT LINUX 9.0

三、实验流程

生产者写入缓冲区和消费者从缓冲区读数的具体流程，生产者首先要 获得互斥锁，并且判断写指针+1 后是否等于读指针，如果相等则进入等待状态，等候条件 变量 notfull；如果不等则向缓冲区中写一个整数，并且设置条件变量为 notempty，最后 释放互斥锁。消费者线程与生产者线程类似。流程图如下：



生产者-消费者实验源代码结构流程

四、实验源代码：

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "pthread.h"

#define BUFFER\_SIZE 16

/\* 设置一个整数的圆形缓冲区 \*/

struct prodcons {

int buffer[BUFFER\_SIZE]; /\* 缓冲区数组 \*/

pthread\_mutex\_t lock; /\* 互斥锁 \*/

int readpos, writepos; /\* 读写的位置\*/\*/

pthread\_cond\_t notempty; /\* 缓冲区非空信号 \*/

pthread\_cond\_t notfull; /\*缓冲区非满信号 \*/

};

/\*--------------------------------------------------------\*/

/\*初始化缓冲区\*/

void init(struct prodcons \* b)

{

pthread\_mutex\_init(&b->lock, NULL);

pthread\_cond\_init(&b->notempty, NULL);

pthread\_cond\_init(&b->notfull, NULL);

b->readpos = 0;

b->writepos = 0;

}

/\*--------------------------------------------------------\*/

/\* 向缓冲区中写入一个整数\*/

void put(struct prodcons \* b, int data)

{

pthread\_mutex\_lock(&b->lock); /\*获取互斥锁\*/

/\*等待缓冲区非满\*/

while ((b->writepos + 1) % BUFFER\_SIZE == b->readpos) {

printf("wait for not full\n");

pthread\_cond\_wait(&b->notfull, &b->lock); /\*等待状态变量 b->notfull，不满则跳出阻塞\*/

}

/\*写数据并且指针前移\*/

b->buffer[b->writepos] = data;

b->writepos++;

if (b->writepos >= BUFFER\_SIZE) b->writepos = 0;

/\*设置缓冲区非空信号\*/

pthread\_cond\_signal(&b->notempty); /\*设置状态变量\*/

pthread\_mutex\_unlock(&b->lock); /\*//释放互斥锁\*/

}

/\*--------------------------------------------------------\*/

/\*从缓冲区中读出一个整数 \*/

int get(struct prodcons \* b)

{

int data;

pthread\_mutex\_lock(&b->lock); /\*获取互斥锁\*/

/\* 等待缓冲区非空\*/

while (b->writepos == b->readpos) {

printf("wait for not empty\n");

pthread\_cond\_wait(&b->notempty, &b->lock);

/\*等待状态变量 b->notempty，不空则跳出阻塞。否则无数据可读 \*/

}

/\* 读数据并且指针前移 \*/

data = b->buffer[b->readpos];

b->readpos++;

if (b->readpos >= BUFFER\_SIZE) b->readpos = 0;

/\* 设置缓冲区非满信号\*/

pthread\_cond\_signal(&b->notfull); /\*设置状态变量\*/

pthread\_mutex\_unlock(&b->lock); /\*释放互斥锁\*/

return data;

}

/\*--------------------------------------------------------\*/

#define OVER (-1)

struct prodcons buffer;

/\*--------------------------------------------------------\*/

void \* producer(void \* data)

{

int n;

for (n = 0; n < 1000; n++) {

printf(" put-->%d\n", n);

put(&buffer, n);

}

put(&buffer, OVER);

printf("producer stopped!\n");

return NULL;

}

/\*--------------------------------------------------------\*/

void \* consumer(void \* data)

{

int d;

while (1) {

d = get(&buffer);

if (d == OVER ) break;

printf(" %d-->get\n", d);

}

printf("consumer stopped!\n");

return NULL;

}

/\*--------------------------------------------------------\*/

int main(void)

{

pthread\_t th\_a, th\_b;

void \* retval;

init(&buffer);

pthread\_create(&th\_a, NULL, producer, 0);

pthread\_create(&th\_b, NULL, consumer, 0);

/\* 等待生产者和消费者结束 \*/

pthread\_join(th\_a, &retval);

pthread\_join(th\_b, &retval);

return 0;

}

五、线程 **API**说明：

在程序的代码中大量的使用了线程函数，如 pthread\_cond\_signal、

pthread\_mutex\_init、pthread\_mutex\_lock 等。下面简单介绍，详细的说明请查阅资料。

* 线程创建函数：int pthread\_create (pthread\_t \* thread\_id, \_\_const pthread\_attr\_t \* \_\_attr, void \*(\*\_\_start\_routine) (void \*),void \*\_\_restrict \_\_arg)
* 获得父进程 ID：pthread\_t pthread\_self (void)
* 测试两个线程号是否相同：int pthread\_equal (pthread\_t \_\_thread1, pthread\_t \_\_thread2)
* 线程退出：void pthread\_exit (void \*\_\_retval)
* 等待指定的线程结束：int pthread\_join (pthread\_t \_\_th, void \*\*\_\_thread\_return)
* 互斥量初始化：pthread\_mutex\_init (pthread\_mutex\_t \*,\_\_const pthread\_mutexattr\_t \*)
* 销毁互斥量：int pthread\_mutex\_destroy (pthread\_mutex\_t \*\_\_mutex)
* 锁定互斥量（阻塞）：：int pthread\_mutex\_lock (pthread\_mutex\_t \*\_\_mutex)
* 解锁互斥量：int pthread\_mutex\_unlock (pthread\_mutex\_t \*\_\_mutex)
* 唤醒线程等待条件变量：int pthread\_cond\_signal (pthread\_cond\_t \*\_\_cond)
* 等待条件变量（阻塞）：：int pthread\_cond\_wait (pthread\_cond\_t \*\_\_restrict \_\_cond, pthread\_mutex\_t \*\_\_restrict \_\_mutex)

六、实验步骤

* 编辑源码，得到pthread.c
* 使用gcc编译（或者使用make工具）
* 运行

七、运行结果：

wait for not empty

put-->994

put-->995

put-->996

put-->997

put-->998

put-->999

producer stopped!

993-->get

994-->get

995-->get

996-->get

997-->get

998-->get

999-->get

consumer stopped!