**《操作系统》**

**实 验 指 导 手 册**

**授课教师： 陈晋鹏**

**实验指导教师： 陈晋鹏**

**教学对象： 二年级本科生**

**开课时间： 春季学期**

**学生姓名： 田宇**

**学号： 2016212011**

**成绩：**

**北京邮电大学软件学院**

**2018年5月7日**

1. 实验目的

熟悉多线程操作

1. 实验内容

生产者消费者问题

1. 实验环境

Linux gcc

1. 实验要求

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。

1. 实验代码

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#define BUFFER\_SIZE 8

struct Products {

int buffer[BUFFER\_SIZE];

/\*保证存取操作的原子性 互斥性\*/

pthread\_mutex\_t locker;

/\*是否可读\*/

pthread\_cond\_t notEmpty;

/\*是否可写\*/

pthread\_cond\_t notFull;

int posReadFrom;

int posWriteTo;

};

int BufferIsFull(struct Products\* products) {

if ((products->posWriteTo + 1) % BUFFER\_SIZE == products->posReadFrom) {

return (1);

}

return (0);

}

int BufferIsEmpty(struct Products\* products) {

if (products->posWriteTo == products->posReadFrom) {

return (1);

}

return (0);

}

/\*制造产品\*/

void Produce(struct Products\* products, int item) {

/\*原子操作\*/

pthread\_mutex\_lock(&products->locker);

/\*无空间可写入\*/

while (BufferIsFull(products)) {

pthread\_cond\_wait(&products->notFull, &products->locker);

}

/\*写入数据\*/

products->buffer[products->posWriteTo] = item;

products->posWriteTo++;

if (products->posWriteTo >= BUFFER\_SIZE)

products->posWriteTo = 0;

/\*发信\*/

pthread\_cond\_signal(&products->notEmpty);

/\*解锁\*/

pthread\_mutex\_unlock(&products->locker);

}

int Consume(struct Products\* products) {

int item;

pthread\_mutex\_lock(&products->locker);

/\*为空时持续等待,无数据可读\*/

while (BufferIsEmpty(products)) {

pthread\_cond\_wait(&products->notEmpty, &products->locker);

}

/\*提取数据\*/

item = products->buffer[products->posReadFrom];

products->posReadFrom++;

/\*如果到末尾,从头读取\*/

if (products->posReadFrom >= BUFFER\_SIZE)

products->posReadFrom = 0;

pthread\_cond\_signal(&products->notFull);

pthread\_mutex\_unlock(&products->locker);

return item;

}

#define END\_FLAG (-1)

struct Products products;

void\* ProducerThread(void\* data) {

int i;

for (i = 0; i < 16; ++i) {

printf("producer: %d\n", i);

Produce(&products, i);

}

Produce(&products, END\_FLAG);

return NULL;

}

void\* ConsumerThread(void\* data) {

int item;

while (1) {

item = Consume(&products);

if (END\_FLAG == item)

break;

printf("consumer: %d\n", item);

}

return (NULL);

}

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

pthread\_t producer;

pthread\_t consumer;

int result;

pthread\_create(&producer, NULL, &ProducerThread, NULL);

pthread\_create(&consumer, NULL, &ConsumerThread, NULL);

pthread\_join(producer, (void \*) &result);

pthread\_join(consumer, (void \*) &result);

exit(EXIT\_SUCCESS);

}

1. 实验结果

