**《操作系统》**

**实 验 指 导 手 册**

**授课教师： 刘知青**

**实验指导教师： 刘知青**

**教学对象： 二年级本科生**

**开课时间： 春季学期**

**学生姓名： 裴仪瑶**

**学号： 2014212052**

**成绩：**

**北京邮电大学软件学院**

**2016年4月22日**

1. 实验目的

熟悉多线程操作

1. 实验内容

一个生产者、一个消费者、一个消费者与生产者的混合。

1. 实验环境

Windows；visual studio

1. 实验要求
2. 生产者：四个线程，作用是产生1000000个随机整数（0-99），存储到buffer1中（容量100）.
3. 消费者：三个线程，每次从buffer1读取三个数字，并且求和。存入buffer2（容量200）。
4. 分解者：从buffer2读取数字，并且分解为质数并记录。统计每个数字出现的次数并画统计图。
5. 实验代码

#include <iostream>

#include<ctime>

#include<thread>

#include<queue>

#include<random>

#include<mutex>

#include <condition\_variable>

using namespace std;

#define TOTAL\_pro 100//目标生产总数

#define MAX\_buffer1 100//buffer1容量

#define MAX\_buffer2 200//buffer2容量

int buffer1\_num = 0;//buffer1现存数量

int buffer2\_num = 0;//buffer2现存数量

int pronum = 0;//实际生产总数

int connum = 0;//实际消费总数

int desnum = 0;//实际分解数量

queue<int> buffer1;//生产缓存

queue<int> buffer2;//消费缓存

int prime[62];//分解因子数组

int table[62] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,

31, 37, 41, 43, 47, 53, 59, 61, 67, 71,

73, 79, 83, 89, 97, 101, 103, 107, 109, 113,

127, 131, 137, 139, 149, 151, 157, 163, 167, 173,

179, 181, 191, 193, 197, 199, 211, 223, 227, 229,

233, 239, 241, 251, 257, 263, 269, 271, 277, 281,

283, 293 };//质数参考数组

mutex mtx;//三种线程间的互斥量

mutex produced\_mtx; //生产者线程间的互斥量

mutex consumed\_mtx; //消费者线程间的互斥量

condition\_variable buffer1\_not\_full; // 条件变量, 生产缓冲区不为满.

condition\_variable buffer1\_not\_empty; // 条件变量, 生产缓冲区不为空.

condition\_variable buffer2\_not\_full; // 条件变量, 生产缓冲区不为满.

condition\_variable buffer2\_not\_empty; // 条件变量, 生产缓冲区不为空.

//生产者

void fproducer(int seed){

srand(seed);//设置种子

bool ready\_to\_exit = false;

while (1) {

unique\_lock<mutex> lock(produced\_mtx);

if (pronum < TOTAL\_pro) {

unique\_lock<mutex> lock(mtx);

while (buffer1\_num >= MAX\_buffer1){ //产品库缓冲区满

cout << "wait...(buffer1 is full)" << endl;

buffer1\_not\_full.wait(lock); // 生产者等待"产品库缓冲区不为满"这一条件发生.

}

int i = rand() % 100;

buffer1.push(i);

pronum++;

buffer1\_num++;

cout << "Producer:" << i << endl;

buffer1\_not\_empty.notify\_all(); // 通知消费者产品库不为空.

lock.unlock(); //解锁

}

else {

cout << "生产完成" << endl;

ready\_to\_exit = true;

}

lock.unlock();

if (ready\_to\_exit == true)

break;

}

}

//消费者

void fconsumer(){

int i1, i2, i3;

bool ready\_to\_exit = false;

while (1) {

unique\_lock<mutex> lock(consumed\_mtx);

if (connum+3 < TOTAL\_pro){

unique\_lock<mutex> lock(mtx);

while (buffer1\_num <= 3) {//产品库缓冲区空

cout << "wait...(buffer1 is empty)" << endl;

buffer1\_not\_empty.wait(lock);//消费者等待"产品库缓冲区不为空"这一条件发生.

}

while (buffer2\_num >= MAX\_buffer2){ //消费缓冲区满

cout << "wait...(buffer2 is full)" << endl;

buffer2\_not\_full.wait(lock); // 消费者等待"消费缓冲区不为满"这一条件发生.

}

i1 = buffer1.front();

buffer1.pop();

i2 = buffer1.front();

buffer1.pop();

i3 = buffer1.front();

buffer1.pop();

int sum = i1 + i2 + i3;

cout <<"consumer:" << i1 <<"+" <<i2<<"+"<<i3<<"="<< sum << endl;

buffer1\_num -= 3;

buffer2.push(sum);

buffer2\_num++;

connum += 3;

buffer1\_not\_full.notify\_all(); // 通知消费者生产缓存区不为满.

buffer2\_not\_empty.notify\_all();//通知分解者消费缓存区不为空.

lock.unlock(); // 解锁.

}

else{

cout << "消费者完成" << endl;

ready\_to\_exit = true;

break;

}

if (ready\_to\_exit == true)

break;

lock.unlock();

}

}

//分解者

void fdecomposer(){

while (1) {

if (desnum < TOTAL\_pro / 3) {

unique\_lock<mutex> lock(mtx);

while (buffer2\_num == 0) {//消费缓冲区为空

cout << "wait...(buffer2 is empty)" << endl;

buffer2\_not\_empty.wait(lock);//分解者等待"消费缓冲区不为空"这一条件发生.

}

int number = buffer2.front();

int i = 0;

cout << "decomposer:" << number << endl;

while (number >= table[i]){

while (number%table[i] == 0){

number = number / table[i];

prime[i]++;

}

i++;

}

buffer2.pop();

buffer2\_num--;

buffer2\_not\_full.notify\_all();// 通知消费者消费缓存区不为满.

lock.unlock(); // 解锁.

}

else{

cout << "分解者完成" << endl;

break;

}

desnum++;

}

}

int main(){

thread producer1(fproducer, 33);

thread producer2(fproducer, 55);

thread producer3(fproducer, 66);

thread producer4(fproducer, 34);

thread consumer1(fconsumer);

thread consumer2(fconsumer);

thread consumer3(fconsumer);

thread decomposer(fdecomposer);

producer1.join();

producer2.join();

producer3.join();

producer4.join();

consumer1.join();

consumer2.join();

consumer3.join();

decomposer.join();

cout << "分解因子表：" << endl;

for (int i = 0; i < 62; i++)

cout << table[i] << ": " << prime[i] << endl;

system("pause");

return 0;

}

1. 实验结果



