**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **Домашнее задание №3**  **Пояснительная записка** | | |
|  |  | |
| Исполнитель  студентка группы БПИ193  В.В.Степанова  16.11.2020 | |
|  | | |
|  | |  |

**Москва 2020**

**Формулировка задания**

*Вариант 20*

Задача о производстве булавок. В цехе по заточке булавок все необходимые операции осуществляются тремя рабочими. Первый из них берет и проверяет ее на предмет кривизны. Если булавка не кривая, то рабочий передает ее своему напарнику. Напарник осуществляет собственно заточку и передает заточенную булавку третьему рабочему, который осуществляет контроль качества операции. Требуется создать многопоточное приложение, моделирующее работу цеха. При решении использоватть парадигму “производитель-потребитель”.

**Список используемой литературы**

1. Грегори Р.Эндрюс. Основы многопоточного, параллельного и распределенного программирования. -М.: Издательский дом “Вильямс”, 2003.
2. Уильямс Энтони. С++. Практика многопоточного программирования. – СПб.: Питер, 2020. – 640с.
3. Posix Threads [Электронный ресурс] / Wikipedia. Режим доступа: <https://ru.wikipedia.org/wiki/POSIX_Threads>, свободный. (дата обращения: 15.11.2020)
4. Peter Van Roy. The principal programming paradigms, 2008 [Электронный ресурс]. Режим доступа: <https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf>, свободный. (дата обращения: 15.11.2020)

**Приложение. Текст программы.**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <iostream>

#define NEEDLES\_NUMBER 100

#define BUF\_SIZE 5

using namespace std;

struct count {

int buf[BUF\_SIZE]; // the buffer

size\_t len;

int counter;

pthread\_mutex\_t mutex;

pthread\_cond\_t can\_produce; // signaled when items are removed

pthread\_cond\_t can\_consume;

pthread\_cond\_t can\_check\_quality;

};

void \*doit(void \*);

void \*check\_curvature(void \*vptr);

void \*sharpen(void \*vptr);

void \*check\_quality(void \*vptr);

int main()

{

pthread\_t worker\_one, worker\_two, worker\_three;

struct count c;

c.len = 0;

pthread\_mutex\_init(&c.mutex, NULL);

pthread\_cond\_init(&c.can\_produce, NULL);

pthread\_cond\_init(&c.can\_consume, NULL);

c.counter = 0;

srandom(1000);

for (int i = 0; i < NEEDLES\_NUMBER; i++)

{

printf("\033[32;1m NEEDLE %d: \033[0m\n", i + 1);

pthread\_create(&worker\_one, NULL, &check\_curvature, &c);

pthread\_create(&worker\_two, NULL, &sharpen, &c);

pthread\_create(&worker\_three, NULL, &check\_quality, &c);

pthread\_join(worker\_one, NULL);

pthread\_join(worker\_two, NULL);

pthread\_join(worker\_three, NULL);

}

exit(0);

}

void \*check\_curvature(void \*vptr)

{

struct count\* c;

c = (struct count\*) vptr;

pthread\_mutex\_lock(&c->mutex);

if(c->len == BUF\_SIZE) {

pthread\_cond\_wait(&c->can\_produce, &c->mutex);

}

int t = rand();

cout << "Curvature is fine" << endl;

c->buf[c->len] = t;

++c->len;

pthread\_cond\_signal(&c->can\_consume);

pthread\_mutex\_unlock(&c->mutex);

}

void \*sharpen(void \*vptr)

{

struct count\* c;

c = (struct count\*) vptr;

pthread\_mutex\_lock(&c->mutex);

if(c->len == 0) {

pthread\_cond\_wait(&c->can\_consume, &c->mutex);

}

--c->len;

cout << "Sharped!" << endl;

pthread\_cond\_signal(&c->can\_produce);

pthread\_mutex\_unlock(&c->mutex);

}

void \*check\_quality(void \*vptr)

{

struct count\* c;

c = (struct count\*) vptr;

pthread\_mutex\_lock(&c->mutex);

cout << "Good quality!" << endl;

pthread\_mutex\_unlock(&c->mutex);

}