Правительство Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Департамент программной инженерии

Отчет к домашнему заданию
По дисциплине
«Архитектура вычислительных систем»

Работу выполнил:

Студент группы БПИ-195 Горбачев Д. Г.

Москва 2020

Задание, вариант 4

Задача об обедающих философах. Пять философов сидят возле круглого стола. Они проводят жизнь, чередуя приемы пищи и размышления. В центре стола находится большое блюдо спагетти. Спагетти длинные и запутанные, философам тяжело управляться с ними, поэтому каждый из них, что бы съесть порцию, должен пользоваться двумя вилками. К несчастью, философам дали только пять вилок. Между каждой парой философов лежит одна вилка, поэтому эти высококультурные и предельно вежливые люди договорились, что каждый будет пользоваться только теми вилками, которые лежат рядом с ним (слева и справа). Написать многопоточную программу, моделирующую поведение философов с помощью семафоров. Программа должна избегать фатальной ситуации, в которой все философы голодны, но ни один из них не может взять обе вилки (например, каждый из философов держит по одной вилки и не хочет отдавать ее). Решение должно быть симметричным, то есть все потоки-философы должны выполнять один и тот же код.

Решение

Имеем 5 философов, создаем для них семафоры.

Также создаем класс для философа, в котором имеем конструктор для вилок и имени.

Создаем 5 потоков с методом действий для философа:

1. Взять вилку, если она доступна (проверка через семафор).
2. Кушать спагетти
3. Положить вилку обратно оповестив об этом семафор.

Join-им их и получаем симметричное решение для философов.

Код:

```
/* Выполнил студент группы БПИ195
Горбачев Даниил Геннадьевич
Вариант 4*/

#include <iostream>
#define HAVE_STRUCT_TIMESPEC
#include <pthread.h>
#include <fstream>
#include <string>
#include <Windows.h>
#include <semaphore.h>

using namespace std;

pthread_mutex_t mutex;
sem_t* semaphoreForks;
sem_t semaphoreWaiter;
class Philosopher
{
public:
	string name;
	int leftFork, rightFork;

	void takeForks()
	{
		cout << name << " waits for forks " << leftFork << " and " << rightFork <<
endl;
		sem_wait(&semaphoreForks[leftFork - 1]);
```

```
		sem_wait(&semaphoreForks[rightFork - 1]);
		cout << name << " took forks " << leftFork << " and " << rightFork << endl;
	}

	void putBackForks()
	{
		sem_post(&semaphoreForks[leftFork - 1]);
		sem_post(&semaphoreForks[rightFork - 1]);
		cout << name << " gave forks " << leftFork << " and " << rightFork << endl;
	}

	void philosopherEat()
	{
		cout << name << " eats with forks " << leftFork << " and " << rightFork <<
endl;
	}

	Philosopher(char* name, int leftFork, int rightFork)
	{
		this->name = name;
		this->leftFork = leftFork;
		this->rightFork = rightFork;
	}

	//Философ ждет вилок, берет их и кушает, после чего кладет обратно и выходит
	static void* philosopherActions(void* arg)
	{
		Philosopher* f = (Philosopher*)arg;
		pthread_mutex_lock(&mutex);
		sem_wait(&semaphoreWaiter);
		f->takeForks();
		f->philosopherEat();
		f->putBackForks();
		sem_post(&semaphoreWaiter);
		pthread_mutex_unlock(&mutex);
		return NULL;
	};

};

void semafArray()
{
	semaphoreForks = new sem_t[5];
	for (int i = 0; i < 5; i++)
		sem_init(&semaphoreForks[i], 0, 1);
}

int main()
{
	semafArray();
	//семафор
	sem_init(&semaphoreWaiter, 0, 4);
	pthread_mutex_init(&mutex, NULL);
	//определениие философов и вилок рядом с ними
	Philosopher Phil1((char*)"Philosopher1", 1, 2);
	Philosopher Phil2((char*)"Philosopher2", 2, 3);
	Philosopher Phil3((char*)"Philosopher3", 3, 4);
	Philosopher Phil4((char*)"Philosopher4", 4, 5);
	Philosopher Phil5((char*)"Philosopher5", 5, 1);

	pthread_t* threadsPhilosopher = new pthread_t[5];
	pthread_create(&threadsPhilosopher[0], NULL, Philosopher::philosopherActions,
&Phil1);
```

```
	pthread_create(&threadsPhilosopher[1], NULL, Philosopher::philosopherActions,
&Phil2);
	pthread_create(&threadsPhilosopher[2], NULL, Philosopher::philosopherActions,
&Phil3);
	pthread_create(&threadsPhilosopher[3], NULL, Philosopher::philosopherActions,
&Phil4);
	pthread_create(&threadsPhilosopher[4], NULL, Philosopher::philosopherActions,
&Phil5);
	for (int i = 0; i < 5; i++)
		pthread_join(threadsPhilosopher[i], NULL);

	return 0;
}
```