ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

Архитектура вычислительных систем
Микропроект №2, вариант 4

ПРОГРАММА ДЛЯ РЕШЕНИЯ ПРОБЛЕМЫ ОБЕДАЮЩИХ ФИЛОСОФОВ

С использованием pthread и semaphore

Пояснительная записка

Исполнитель:
студент группы БПИ191
В.И. Беловицкий/

2020 г.

СОДЕРЖАНИЕ

1. ПОСТАВЛЕННАЯ ЗАДАЧА

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, чтобы 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.

2. ИСПОЛЬЗОВАННЫЙ АЛГОРИТМ

При решении задачи был использован подход Таненбаума. Используются только бинарные семафоры:

– один общий мъютекс, необходимый для того, чтобы сделать процессы взятия и возврата вилок критическими секциями – для того, чтобы два философа не могли это сделать одновременно. Благодаря этому не может возникнуть ситуация, что каждый философ взял по одной вилке (дедлок).

– по семафору на каждую вилку.

У философа может быть три состояния – THINKING, HUNGRY, EATING.

Изначально все философы находятся в состоянии THINKING, затем начинают бесконечный цикл (рис 1).

```
// имитация деятельности философа
void* philospher(void* num) {

    while (true) {
        int* i = (int*)num;

        sleep(STIME);
        // Пытаемся взять вилки
        take_fork(*i);

        sleep(STIME);
        // Кладем вилки обратно на стол
        put_fork(*i);
    }
}
```

Рисунок 1. Деятельность философа

Критическими секциями являются процесс взятия вилки и процесс возврата вилки (рис 2, 3):

```
// процесс взятия вилок
void take_fork(int phnum) {
    sem_wait(&mutex);

    // меняем состояние на голодное
    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);

    // Пытаемся поесть
    test(phnum);

    sem_post(&mutex);

    // если нет возможности подождать, пока не будет получен сигнал
    sem_wait(&S[phnum]);
}
```

Рисунок 2. Критическая секция, взятие вилки

```
// процесс возврата вилок
void put_fork(int phnum) {
    sem_wait(&mutex);

    // переходим обратно в думающее состояние
    state[phnum] = THINKING;

    printf("Philosopher %d put back fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is Thinking\n", phnum + 1);

    // Вежливо предлагаем правому и левому соседам поесть
    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}
```

Рисунок 3. Критическая секция, возврат вилок

При взятии вилок философ пытается поесть, при возврате вилок философ предлагает (сигнализирует) своим соседам поесть. Попытка поесть удачна, если текущий философ голоден, а левы и правый соседы не едят (рис 4):

```
// попытка поесть
void test(int phnum) {
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {

        // меняем статус на "ем"
        state[phnum] = EATING;
        sleep(STIME);

        printf("Philosopher %d take fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);

        // при take_fork() не имеет эффекта
        // при put_fork() пробуждает голодных философов
        sem_post(&S[phnum]);
    }
}
```

Рисунок 4. Процесс попытки поесть

3. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входные данные определяется статически, в коде программы.

Изменяемыми являются параметры N – количество философов (по умолчанию 5), и STIME – время в миллисекундах для функции sleep (по умолчанию 3000).

Функция sleep является кроссплатформенной, ее реализация определяется в зависимости от операционной системы – Windows или Linux.

Выходными данными являются сообщения о деятельности философов (потоков).

4. ТЕСТИРОВАНИЕ

Запустим программу. Изначально философы находятся в состоянии THINKING, затем начинается бесконечный цикл.

```
Philosopher 1 is Thinking
Philosopher 2 is Thinking
Philosopher 3 is Thinking
Philosopher 4 is Thinking
Philosopher 5 is Thinking
Philosopher 2 is Hungry
Philosopher 2 take fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 4 take fork 3 and 4
Philosopher 4 is Eating
Philosopher 5 is Hungry
Philosopher 2 put back fork 1 and 2
Philosopher 2 is Thinking
Philosopher 1 take fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 put back fork 3 and 4
Philosopher 4 is Thinking
Philosopher 3 take fork 2 and 3
Philosopher 3 is Eating
Philosopher 2 is Hungry
Philosopher 1 put back fork 5 and 1
Philosopher 1 is Thinking
Philosopher 5 take fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 put back fork 2 and 3
```

Рисунок 5. Тестовый вывод программы

5. ЛИТЕРАТУРА

1. Э. Таненбаум. Современные операционные системы (4-е издание) 2015 г. (дата обращения: 13.12.2020).

2. The Dining Philosophers Problem [Электронный ресурс]. URL https://legacy.cs.indiana.edu/classes/p415-sjoh/hw/project/dining-philosophers/index.htm (дата обращения: 13.12.2020).

3. Solution of Dining Philosophers Problem using Semaphores [Электронный ресурс]. URL:

http://cs.gordon.edu/courses/cs322/lectures/transparencies/dining_phil.html (дата обращения: 13.12.2020).

6. ТЕКСТ ПРОГРАММЫ

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

// Кроссплатформенный sleep
#ifdef _WIN32
#include <windows.h>
void sleep(unsigned milliseconds)
{
    Sleep(milliseconds);
}
#else
#include <unistd.h>
void sleep(unsigned milliseconds)
{
    usleep(milliseconds * 1000);
}
#endif

// Количество философов
#define N 5
// Время для sleep в миллисекундах
#define STIME 3000

// Числовые значения состояний
#define THINKING 0
#define HUNGRY 1
#define EATING 2

// Удобные обозначения левого и правого соседов phnum-го философа
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

// Массив состояний философов
int state[N];
// Массив номеров философов
int phil[N] = { 0, 1, 2, 3, 4 };

// Общий бинарный семафор, необходимый для того, чтобы философы не могли одновременно
взять/положить вилки
sem_t mutex;
// Общие бинарные семафоры для каждой вилки
sem_t S[N];

// попытка поесть
void test(int phnum) {
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {

        // меняем статус на "ем"
        state[phnum] = EATING;
        sleep(STIME);

        printf("Philosopher %d take fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is Eating\n", phnum + 1);

        // при take_fork() не имеет эффекта
        // при put_fork() пробуждает голодных философов
        sem_post(&S[phnum]);
```

```
    }
}


// процесс взятия вилок
void take_fork(int phnum) {
    sem_wait(&mutex);

    // меняем состояние на голодное
    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);

    // Пытаемся поесть
    test(phnum);

    sem_post(&mutex);

    // если нет возможности подождать, пока не будет получен сигнал
    sem_wait(&S[phnum]);
}


// процесс возврата вилок
void put_fork(int phnum) {
    sem_wait(&mutex);

    // переходим обратно в думающее состояние
    state[phnum] = THINKING;

    printf("Philosopher %d put back fork %d and %d\n", phnum + 1, LEFT + 1, phnum + 1);
    printf("Philosopher %d is Thinking\n", phnum + 1);

    // Вежливо предлагаем правому и левому соседам поесть
    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}

// имитация деятельности философа
void* philospher(void* num) {

    while (true) {
        int* i = (int*)num;

        sleep(STIME);
        // Пытаемся взять вилки
        take_fork(*i);

        sleep(STIME);
        // Кладем вилки обратно на стол
        put_fork(*i);
    }
}

int main(){

    pthread_t thread_id[N];

    // инициализируем семафоры
    sem_init(&mutex, 0, 1);
    for (int i = 0; i < N; i++) {
        sem_init(&S[i], 0, 0);
```

```
    }

    for (int i = 0; i < N; i++) {
        // создаем и запускаем процессы философов
        pthread_create(&thread_id[i], NULL, philospher, &phil[i]);

        // перед началом трапезы философы думают
        printf("Philosopher %d is Thinking\n", i + 1);
    }

    // ждем завершения работы потоков
    // (но в данной реализации философы ведут свою деятельность бесконечно)
    for (int i = 0; i < N; i++) {
        pthread_join(thread_id[i], NULL);
    }

}
```