Правительство Российской Федерации

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

Факультет компьютерных наук

Пояснительная записка микропроекту №2

По дисциплине

“Архитектура вычислительных систем”

Исполнитель
Студентка группы БПИ194
А.И. Гилазутдинова
Вариант 6

Москва 2020

Оглавление

1. Постановка задачи

1) Задача о курильщиках. Есть три процесса-курильщика и один процесс-посредник. Курильщик непрерывно скручивает сигареты и курит их. Чтобы скрутить сигарету, нужны табак, бумага и спички. У одного процесса-курильщика есть табак, у второго – бумага, а у третьего – спички. Посредник кладет на стол по два разных случайных компонента. Тот процесс- курильщик, у которого есть третий компонент, забирает компоненты со стола, скручивает сигарету и курит. Посредник дожидается, пока курильщик закончит, затем процесс повторяется. Создать многопоточное приложение, моделирующее поведение курильщиков и посредника. При решении задачи использовать семафоры.
2) В процессе выводить результат работы программы в консоль.

2. Уточнение задачи

1) Реализовать программу, написанную на языке C++ 14.0.
2) При реализации проекта предлагается использовать библиотеку Posix thread (pthread) или стандартную библиотеку потоков (thread) языка программирования C++.
3) Количество повторов задается с помощью аргументов командной строки.

3. Теоретическая справка

Семафор — это механизм синхронизации, который можно использовать для управления отношениями между параллельно выполняющимися программными компонентами и реализации стратегий доступа к данным. Семафор — это переменная специального вида, которая может быть доступна только для выполнения узкого диапазона операций. Семафор используется для синхронизации доступа процессов и потоков к разделяемой модифицируемой памяти или для управления доступом к устройствам или другим ресурсам. Семафор можно рассматривать как ключ к ресурсам. Этим ключом может владеть в любой момент времени только один процесс или поток. Какая бы задача ни владела этим ключом, он надежно запирает (блокирует) нужные ей ресурсы для ее монопольного использования. Блокирование ресурсов заставляет другие задачи, которые желают воспользоваться этими ресурсами, ожидать до тех пор, пока они не будут разблокированы и снова станут доступными. После разблокирования ресурсов следующая задача, ожидающая семафор, получает его и доступ к ресурсам. Какая задача будет следующей, определяется стратегией планирования, действующей для данного потока или процесса.

Как упоминалось выше, к семафору можно получить доступ только с помощью специальных операций, подобных тем, которые выполняются с объектами. Это операции декремента, P(), и инкремента, V(). Если объект Mutex представляет собой семафор, то логика реализации операций **P (Mutex)** и **V (Mutex)** может выглядеть таким образом*:*

```
P(Mutex)
if (Mutex > 0) {
Mutex--;
} else {
Блокирование по объекту Mutex;
}
V(Mutex)
if(Очередь доступа к объекту Mutex не пуста){
Передача объекта Мьютекс следующей задаче;
} else {
Mutex++;
}
```

Реализация зависит от конкретной системы. Эти операции неделимы, т.е. их невозможно прервать. Если операцию P () попытаются выполнить сразу несколько задач, то лишь одна из них получит разрешение продолжить работу. Если объект Mutex был уже декрементирован, то задача будет заблокирована и займет место в очереди. Операция V () вызывается задачей, которая имеет доступ к объекту Mutex. Если получения доступа к объекту Мьютекс ожидают другие задачи, он «передается » следующей задаче из очереди. Если очередь задач пуста, объект Mutex инкрементируется. [1]

4. Реализация программы

Для реализации программы была выбрана библиотека Posix thread (pthread).
Для реализации программы были созданы 4 потока и 3 семафора.

Потоки:

Название потока	**Реализуемый метод**	**Назначение**
smoker_w_paper_th	smoker_look_at_table (курильщик смотрит на стол)	Имитация процесса-курильщика с бумагой
smoker_w_tobacco_th	smoker_look_at_table (курильщик смотрит на стол)	Имитация процесса-курильщика с табаком
smoker_w_matches_th	smoker_look_at_table (курильщик смотрит на стол)	Имитация процесса-курильщика со спичками
mediator_th	getIngredients (выбор двух ингредиентов)	Имитация процесса-посредника

Семафоры:

Название семафора	**Начальное значение**	**Назначение**
mediator	0	Оповещает процессы-курильщики, выбрал ли посредник 2 рандомных ингредиента
print	1	Для контроля вывода информации в консоль
found	0	с помощью него процесс-курильщик, который нашел недостающие ингредиенты оповещает процесс-посредник и двух других процессов-курильщиков, что он докурил

Ниже представлена блок-схема работы одного цикла программы. Программу можно зациклить любое количество раз, так как значения семафоров в начале и в конце одного цикла одинаковые.

Сам код программы предоставлен в Приложении.

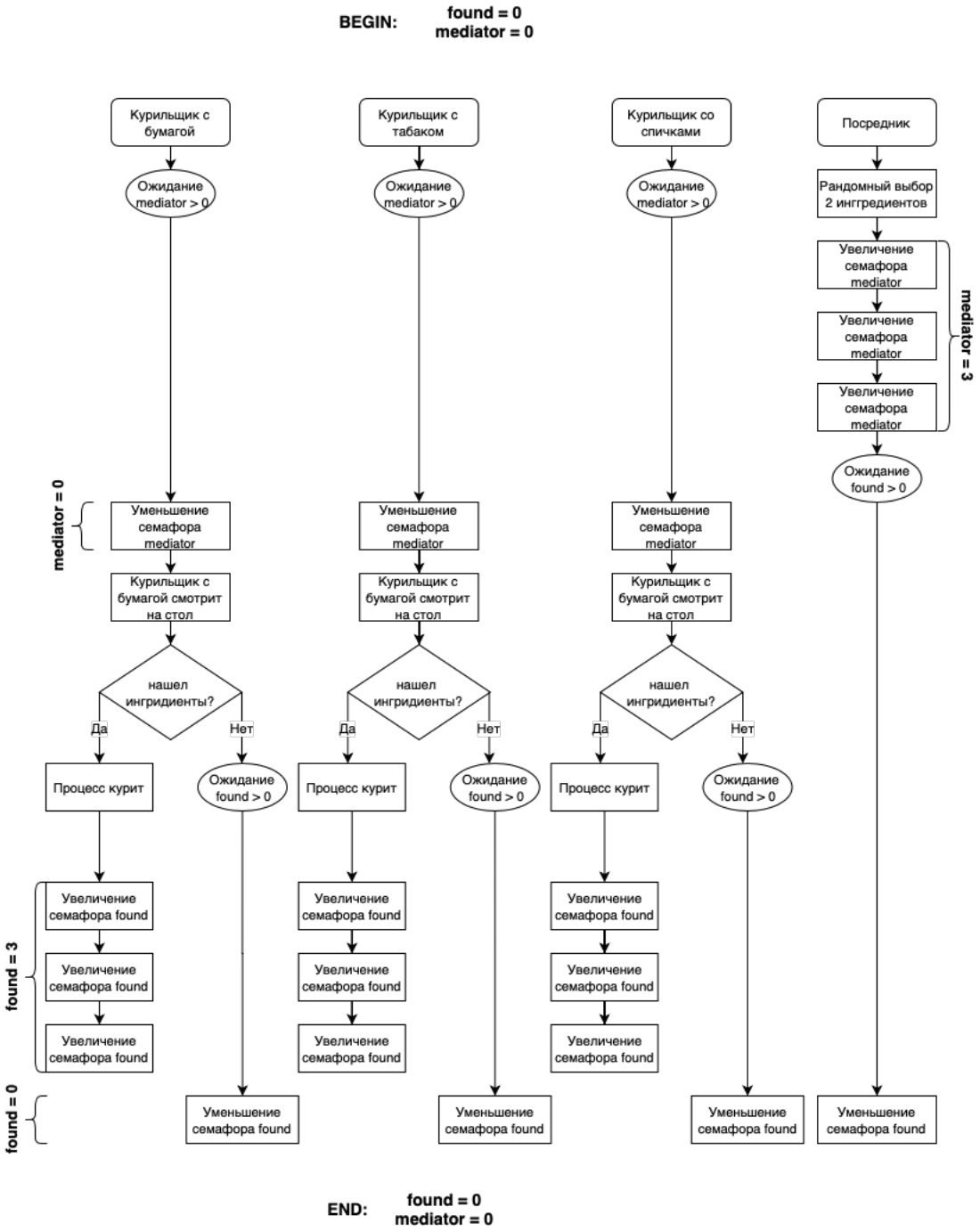

Рисунок 1 – блок-схема

5. Тестирование программы

Запуск программы осуществляется с помощью командной строки. При запуске программы передается 1 аргумент – количество повторов процесса. При вводе пользователем некорректных значений в консоль выводится сообщение об ошибке и программа завершается. При отсутствии ввода пользователем значения процессы повторяются до тех пор, пока пользователь не остановит программу.

1) Пример работы программы со значением 7.

```
mediator is choosing ingredients . . .
mediator puts tobacco and matches on the table . . .
smoker with matches is looking at the table . . .
smoker with paper is looking at the table . . .
smoker with paper found the missing ingredients . . .
smoker with paper is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts matches and tobacco on the table . . .
smoker with paper is looking at the table . . .
smoker with paper found the missing ingredients . . .
smoker with paper is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts matches and paper on the table . . .
smoker with matches is looking at the table . . .
smoker with tobacco is looking at the table . . .
smoker with tobacco found the missing ingredients . . .
smoker with tobacco is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts matches and tobacco on the table . . .
smoker with matches is looking at the table . . .
smoker with tobacco is looking at the table . . .
smoker with paper is looking at the table . . .
smoker with paper found the missing ingredients . . .
smoker with paper is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts paper and matches on the table . . .
smoker with tobacco is looking at the table . . .
smoker with tobacco found the missing ingredients . . .
smoker with tobacco is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts paper and matches on the table . . .
smoker with matches is looking at the table . . .
smoker with tobacco is looking at the table . . .
smoker with tobacco found the missing ingredients . . .
smoker with tobacco is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts paper and matches on the table . . .
smoker with tobacco is looking at the table . . .
smoker with tobacco found the missing ingredients . . .
smoker with tobacco is smoking
--- ~
Program ended with exit code: 0
```

Рисунок 2 – программа со значением 7

2) Пример работы программы со значением 3.

```
mediator is choosing ingredients . . .
mediator puts tobacco and matches on the table . . .
smoker with tobacco is looking at the table . . .
smoker with matches is looking at the table . . .
smoker with paper is looking at the table . . .
smoker with paper found the missing ingredients . . .
smoker with paper is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts matches and tobacco on the table . . .
smoker with matches is looking at the table . . .
smoker with paper is looking at the table . . .
smoker with tobacco is looking at the table . . .
smoker with paper found the missing ingredients . . .
smoker with paper is smoking
--- ~
mediator is choosing ingredients . . .
mediator puts matches and paper on the table . . .
smoker with matches is looking at the table . . .
smoker with paper is looking at the table . . .
smoker with tobacco is looking at the table . . .
smoker with tobacco found the missing ingredients . . .
smoker with tobacco is smoking
--- ~
Program ended with exit code: 0
```

Рисунок 3 – программа со значением 3

3) Пример работы программы с некорректными данными.

```
Number of repetitions must be an integer number > 0
Program ended with exit code: 0
```

Рисунок 3 – программа с некорректным значением (ф)

Использованная литература

1) Камерон Хьюз. Параллельное и распределенное программирование с использованием C++/ Камерон Хьюз, Трейси Хьюз. пер. Н. М. Ручко. – Москва: Вильямс 2004. – 130 с.
2) https://man7.org [Электронный ресурс]/ man7.org Режим доступа: https://man7.org/linux/man-pages/man3/ , свободный (Дата обращения: 11.12.2020).

Приложение

```
/*
 * Гилазутдинова Аделя БПИ194 6 вар
 *
 * Задача о курильщиках.
 * Есть три процесса-курильщика и один процесс-посредник. Курильщик непрерывно
 * скручивает сигареты и курит их. Чтобы скрутить сигарету, нужны табак,
 * бумага и спички. У одного процесса- курильщика есть табак, у второго – бумага,
 * а у третьего – спички. Посредник кладет на стол по два разных случайных
компонента.
 * Тот процесс- курильщик, у которого есть третий компонент, забирает компоненты
со стола,
 * скручивает сигарету и курит. Посредник дожидается, пока курильщик закончит,
 * затем процесс повторяется. Создать многопоточное приложение, моделирующее
поведение
 * курильщиков и посредника. При решении задачи использовать семафоры.
 */
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <string>
#include <thread>
#include <zconf.h>
using namespace std;
enum Ingredients /* Enum representing the ingredients */
{
    paper,
    tobacco,
    matches
};
bool smoker_found;       //флаг (нашел ли курильщик недостающий ингредиент)
sem_t *mediator,         //семафор, обозначающий выбрал ли посредник 2 рандомных
ингредиента
*print,          //семафор для изменения глобальных переменных и печати в консоль
*found;          //семафор, обозначающий нашел ли курильщик недостающие
ингредиенты на столе
Ingredients ingredient1;//первый выбранный посредником ингредиент
Ingredients ingredient2;//второй выбранный посредником ингредиент
int iter_max, iter_temp;//переменные для итерирования повторений

/// Метод, переводящий элемент перечисления в строковвый формат
/// \param ingredient - элемент перечисления
/// \return - строковцый формат
string ToString(Ingredients ingredient) {
    switch (ingredient) {
        case paper:
            return "paper";
        case tobacco:
            return "tobacco";
        case matches:
            return "matches";
    }
}

/// Метод, имитирующий курение
/// \param ingredient - ингредиент, которым владеет курильщик
void smoke(Ingredients ingredient) {
    cout << "smoker with " + ToString(ingredient) + " is smoking" << endl;
    this_thread::sleep_for(chrono::milliseconds(500));
```

```
    cout << "-";
    this_thread::sleep_for(chrono::milliseconds(500));
    cout << "-";
    this_thread::sleep_for(chrono::milliseconds(500));
    cout << "-";
    this_thread::sleep_for(chrono::milliseconds(500));
    cout << " ~" << endl;
    this_thread::sleep_for(chrono::milliseconds(500));
}

/// Метод, имитирующий ожидание
/// \param message - выводимое сообщение
/// \param millisec - количество миллисекунд ожидания
void write_with_pause(string message, int millisec) {
    cout << message;
    this_thread::sleep_for(chrono::milliseconds(millisec));
    cout << " . ";
    this_thread::sleep_for(chrono::milliseconds(millisec));
    cout << ". ";
    this_thread::sleep_for(chrono::milliseconds(millisec));
    cout << ". " << endl;
    this_thread::sleep_for(chrono::milliseconds(millisec));
}

/// Метод, имитирующий то, как курильщики смотрят на стол и ищут недостающие
ингредиенты
/// \param ingr - ингредиент курильщика
/// \return - nullptr
void *smoker_look_at_table(void *ingr) {
    Ingredients ingredient = static_cast<Ingredients>((uintptr_t) ingr);
    //определенное количество повторений (заданное в аргументах ком. строки) либо
беск. цикл
    for (int i = 1; i <= iter_max; i += iter_temp) {
        //курильщик ожидает, пока посредник выбирает ингридиенты
        sem_wait(mediator);
        //если курильщик с недостающим ингридиентом не найден
        sem_wait(print);
        //есди курильщик с недостающим ингредиентом не наайден
        if (!smoker_found)
            write_with_pause("smoker with " + ToString(ingredient) + " is looking
at the table", 100);
        sem_post(print);
        //если у курильщика есть недостающий ингредиент
        if (ingredient != ingredient1 && ingredient != ingredient2) {
            smoker_found = 1;
            sem_wait(print);
            write_with_pause("smoker with " + ToString(ingredient) + " found the
missing ingredients", 500);
            sem_post(print);
            smoke(ingredient);
            //3 раза, так как его ждут остальные 2 курильщика и посредник
            sem_post(found);
            sem_post(found);
            sem_post(found);
        } else {
            //ожидание, когда курильщик покурит
            sem_wait(found);
        }
    }
    return nullptr;
}
```

```
void *getIngredients(void *args) {
    //определенное количество повторений (заданное в аргументах ком. строки) либо
беск. цикл
    for (int i = 1; i <= iter_max; i += iter_temp) {
        write_with_pause("mediator is choosing ingredients", 500);
        smoker_found = 0;
        //рандомный выбор ингредиентов
        int first = rand() % 3;
        int second = rand() % 3;
        while (second == first)
            second = rand() % 3;
        ingredient1 = static_cast<Ingredients>(first);
        ingredient2 = static_cast<Ingredients>(second);
        write_with_pause("mediator puts " + ToString(ingredient1) + " and " +
ToString(ingredient2) + " on the table", 500);
        //3 раза, потому что его ждут 3 курильщика
        sem_post(mediator);
        sem_post(mediator);
        sem_post(mediator);
        //ожидание, когда курильщик покурит
        sem_wait(found);
        //sem_post(found);
    }
    return nullptr;
}
int main(int argc, char **argv) {
    //если в агрументы не было передано количество повторений, то повториний
будет бесконечное количество
    if (argc == 1) {
        iter_max = 1;
        iter_temp = 0;
    } else {
        if(argc > 2){
            cout << "You should enter 1 integer number > 0" << endl;
            return 0;
        }
        iter_temp = 1;
        iter_max = atoi(argv[1]);
        if (iter_max <= 0) {
            cout << "Number of repetitions must be an integer number > 0" <<
endl;
            return 0;
        }
    }
    //инициализация семафоров
    sem_unlink("/mediator");
    sem_unlink("/print");
    sem_unlink("/found");
    mediator = sem_open("/mediator", O_CREAT, S_IRWXU, 0);
    print = sem_open("/print", O_CREAT, S_IRWXU, 1);
    found = sem_open("/found", O_CREAT, S_IRWXU, 0);
    if (!mediator || !print || !found) {
        cout << "Semaphores failed" << endl;
        return 0;
    }
    //создание потоков
    pthread_t smoker_w_paper_th;
    pthread_t smoker_w_tobacco_th;
    pthread_t smoker_w_matches_th;
    pthread_t mediator_th;
    pthread_create(&smoker_w_paper_th, NULL, smoker_look_at_table, (char *) 0 +
```

```
paper);
    pthread_create(&smoker_w_tobacco_th, NULL, smoker_look_at_table, (char *) 0 +
tobacco);
    pthread_create(&smoker_w_matches_th, NULL, smoker_look_at_table, (char *) 0 +
matches);
    pthread_create(&mediator_th, NULL, getIngredients, NULL);

    //синхронизация потоков
    pthread_join(smoker_w_paper_th, nullptr);
    pthread_join(smoker_w_tobacco_th, nullptr);
    pthread_join(smoker_w_matches_th, nullptr);
    pthread_join(mediator_th, nullptr);

    //удаление семафоров
    sem_unlink("mediator");
    sem_unlink("print");
    sem_unlink("found");
    sem_close(print);
    sem_close(mediator);
    sem_close(found);

    return 0;
}
```