

Национальный исследовательский университет  
“Высшая Школа Экономики”,  
Факультет компьютерных наук  
Департамент программной инженерии

**«Микропроект №2»**

Пояснительная записка к разработке консольного приложения

Исполнитель:  
Студент группы БПИ199  
Волохов Никита Алексеевич

## Оглавление

<b>1. Текст задания.....</b>	<b>3</b>
<b>2. Применяемые расчетные методы.....</b>	<b>4</b>
а. Запуск программы .....	4
б. Работа и распараллеливание программы.....	4
с. Безопасность.....	6
<b>3. Тестовые примеры .....</b>	<b>7</b>
<b>4. Список используемых источников.....</b>	<b>9</b>

## 1. Текст задания

Формулировка задания: «*Задача о каннибалах*. Племя из  $n$  дикарей ест вместе из большого горшка, который вмещает  $m$  кусков тушеного миссионера. Когда дикарь хочет обедать, он ест из горшка один кусок, если только горшок не пуст, иначе дикарь будит повара и ждет, пока тот не наполнит горшок. Повар, сварив обед, засыпает. Создать многопоточное приложение, моделирующее обед дикарей. При решении задачи пользоваться семафорами.» [1].

## 2. Применяемые расчетные методы

### а. Запуск программы

Данная программа запускается при помощи следующей команды:  
`c++ ./main.cpp -o main -lpthread -std=c++11 && ./main [argv1] [argv2]`, где:

- `c++ ./main.cpp -o main -lpthread -std=c++11` – компиляция программы;
- `./main [argv1] [argv2]` – запуск скомпилированного файла. Аргументы `argv1` и `argv2` – обязательные для запуска программы (`argv1` задает количество каннибалов (количество потоков), а `argv2` – количество кусков мяса, помещающихся в один горшок (количество задач, которые выполняют потоки)). Положительные числа. Аргумент `argv1` сохраняется в переменную `threadsNum`, `argv2` – в `tasksNum`, `currPiecesNum` (глобальная переменная) и `BOWL_CAPACITY` (глобальная переменная). Аргумент `argv3` – необязательный. Задает количество итераций программы (сколько раз горшок с кусками мяса будет полон). Положительное число. Сохраняется в переменную `iterationsNum`, по умолчанию равной 2.

### б. Работа и распараллеливание программы

По условию задачи, решение данной задачи необходимо реализовать с использованием библиотеки POSIX Threads (PThreads) [2] при помощи семафоров. Данное требование может быть интерпретировано следующим образом: не факт, что каннибалы будут есть куски мяса из миски по очереди, скорее всего, они будут делать это в случайном порядке.

В начале выполнения программы, в консоль выводится строка формата “`threadsNum` каннибалов и `tasksNum` кусков помещается в горшке, горшок заполняется `iterationsNum` раз”.

Все потоки запускают стартовую функцию для потоков `func`, входной параметр которого – номер аборигена (номер потока), берущий кусок мяса из горшка.

Метод `func` выводит сообщение о действии каннибала под номером, переданным в данную функцию (внутри тела функции заносится в переменную `threadNum`): если мясо в горшке не закончилось, то есть если значение переменной `currPiecesNum` не равно нулю, то каннибал съедает кусок мяса и значение переменной `currPiecesNum` становится на единицу

меньше. Выводится соответствующее сообщение формата *“Каннибал threadNum съел кусок. Осталось currPiecesNum кусков”*.

Если мяса в горшке не осталось, то выполняется другое действие: каннибал, которому не досталось мясо, будит повара (в консоль выводится строка формата *“Каннибалу threadNum не досталось куска, он будит повара”*, значение переменной currPiecesNum становится равным BOWL\_CAPACITY, в консоль выводится строка формата *“Повар разбужен. Теперь в горшке снова BOWL\_CAPACITY кусков”*).

Все строки выводятся с установленной задержкой при помощи команды usleep из библиотеки встроенной в C++ библиотеки unistd [3].

Все тело функции func представляет собой критическую секцию (так как идет обращение и изменение глобальной переменной currPiecesNum, производится вывод сообщений в консоль), поэтому для синхронизации потоков было решено использовать двоичный семафор (mutex из библиотеки PThreads). Протокол входа в критическую секцию – вызов функции pthread\_mutex\_lock(&mutex1), закрывающей двоичный семафор и не позволяющей другим потоком зайти в данный участок кода, протокол выхода из критической секции – вызов функции pthread\_mutex\_unlock(&mutex1), открывающей двоичный семафор и позволяющий другим потоком зайти в данный участок кода, где mutex1 – глобальная переменная для хранения двоичного семафора.

Также существуют два сценария отработки программы: когда количество каннибалов (количество потоков) 1 и когда их больше 1.

Когда поток 1, программа (из функции main) выполняется последовательно вызовом функции func с входным параметром 1 через главный поток. Обернут в цикл for от int iterationNum = 0 до iterationsNum, чтобы выполнить фиксированное количество итераций программы.

Когда потоков больше 1, командой pthread\_mutex\_init(&mutex1, NULL) инициализируется двоичный семафор, и инициализируется массив pthread\_t threads[threadsNum] для хранения дочерних потоков. Массив заполняется в цикле for от int i = 0 до threadsNum - 1 командой pthread\_create(&threads[i], NULL, func, (void\*)(i + 1)), где i + 1 – номер каннибала (номер потока). Все потоки вызываются в другом цикле for (тоже от int i = 0 до threadsNum - 1) командой pthread\_join(threads[i], NULL). Обернут в цикл do while (numOfTasksDone != iterationsNum \* tasksNum + iterationsNum - 1), внутри которого при помощи счетчиков numOfThreadsCreated и

`numOfTasksDone` идет подсчет количества созданных и отработанных потоков, не превышающее `iterationsNum * tasksNum + iterationsNum - 1`, чтобы не создать и не запустить их больше, чем требует количество итераций.

После выполнения программы, в консоль выводится сообщение *“Миска больше наполняться не будет, повар отказывается”*.

### с. Безопасность

В программе предусмотрен предупреждения выполнения программы с неправильными входными данными.

Перед тем, как входные параметры сохранять в соответствующие переменные, они проверяются на то, числа ли они. Для этого была написана функция `IsNumber`, проверяющая каждый символ входящей в функцию строки на то, цифра ли это (если первый символ строки – “-”, то есть знак минус, то проверка происходит со второго символа до конца). Если хоть один из введенных входных параметров – не число, то в консоль выводится сообщение *“Неверные входные параметры: параметры должны быть числами. Завершение программы”*, после чего программа завершает свое выполнение.

После сохранения входных параметров в соответствующие переменные, они проверяются на то, положительные ли это числа, при помощи оператора сравнения `>= 0`. Если хоть один из входных параметров – неположительное число, то в консоль выводится сообщение *“Неверные входные параметры: параметры должны быть числами. Завершение программы”*, после чего программа завершает свое выполнение.

### 3. Тестовые примеры

Программа корректно работает при использовании одного (главного) потока (см. Рисунок 1).

```
1 каннибалов и 4 кусков помещается в горшке, горшок заполняется 2 раз

Каннибал 1 съел кусок. Осталось 3 кусков
Каннибал 1 съел кусок. Осталось 2 кусков
Каннибал 1 съел кусок. Осталось 1 кусков
Каннибал 1 съел кусок. Осталось 0 кусков

Каннибалу 1 не досталось куска, он будит повара
Повар разбужен. Теперь в горшке снова 4 кусков

Каннибал 1 съел кусок. Осталось 3 кусков
Каннибал 1 съел кусок. Осталось 2 кусков
Каннибал 1 съел кусок. Осталось 1 кусков

Миска больше наполняться не будет, повар отказывается
```

Рисунок 1. Использование одного потока

Программа корректно работает при использовании нескольких дочерних потоков. Количество потоков больше количества задач (см. Рисунок 2).

```
5 каннибалов и 4 кусков помещается в горшке, горшок заполняется 2 раз

Каннибал 1 съел кусок. Осталось 3 кусков
Каннибал 3 съел кусок. Осталось 2 кусков
Каннибал 2 съел кусок. Осталось 1 кусков
Каннибал 4 съел кусок. Осталось 0 кусков

Каннибалу 5 не досталось куска, он будит повара
Повар разбужен. Теперь в горшке снова 4 кусков

Каннибал 1 съел кусок. Осталось 3 кусков
Каннибал 2 съел кусок. Осталось 2 кусков
Каннибал 3 съел кусок. Осталось 1 кусков
Каннибал 4 съел кусок. Осталось 0 кусков

Миска больше наполняться не будет, повар отказывается
```

Рисунок 2. Количество дочерних потоков больше количества задач

Программа корректно работает при использовании нескольких дочерних потоков. Количество потоков больше количества задач (см. Рисунок 2).

```
$ clang++ -Xpreprocessor -fopenmp -std=c++11 -I/usr/local/include -L/usr/local/lib -lomp main.cpp -o main && ./main 3 1
Enter matrix numbers for line #1
2
3
0

Enter matrix numbers for line #2
3
4
0

Enter matrix numbers for line #3
5
6
0

Thread #1 calculated rank of minor 1 x 1: 1
Thread #1 calculated rank of minor 2 x 2: 2
Thread #1 calculated rank of minor 3 x 3: 2
Matrix A rank is 2
```

Рисунок 3. Ранг матрицы не равен ее размерности



#### 4. Список используемых источников

1. Практические приемы построения многопоточных приложений. [Электронный ресурс]. // URL: <http://softcraft.ru/edu/comparch/tasks/mp02>. (Дата обращения: 09.12.2020, режим доступа: свободный).
2. POSIX Threads. [Электронный ресурс]. // URL: [https://ru.wikipedia.org/wiki/POSIX\\_Threads](https://ru.wikipedia.org/wiki/POSIX_Threads). (Дата обращения: 10.12.2020, режим доступа: свободный).
3. unistd.h [Электронный ресурс]. // URL: <https://pubs.opengroup.org/onlinepubs/7908799/xsh/unistd.h.html>. (Дата обращения: 10.12.2020, режим доступа: свободный).