

Задание 1

Необходимые знания

1. Компилирование программ с помощью `gcc`.
2. Состояние гонки.
3. Критическая секция.
4. POSIX threads: как создавать, как дожидаться завершения.
5. Как линковаться на библиотеку `pthread`

Скомпилировать `mutex.c` без использования и с использованием мьютекса. Объяснить разницу в поведении программы.

Компилирование программ с помощью `gcc`

`gcc` — это компилятор для языка C. Он преобразует исходный код в исполняемый файл. Пример команды:

Флаг `-o` задаёт имя выходного файла. Можно также добавлять другие флаги (например, `-Wall` для предупреждений или `-pthread` для поддержки потоков).

Состояние гонки (Race Condition)

Состояние гонки возникает, когда два или более потока одновременно обращаются к общим данным, и порядок выполнения влияет на результат. Это может привести к ошибкам, которые сложно отследить и воспроизвести.

Критическая секция

Критическая секция — это участок кода, где происходит доступ к общим данным. Чтобы избежать состояния гонки, такой код защищают: например, с помощью мьютексов, чтобы только один поток мог выполнять его в данный момент.

POSIX Threads: как создавать, как дожидаться завершения

- **Создание потока:** Поток запускается с помощью системной функции, которая указывает, что он должен делать.
- **Ожидание завершения:** Чтобы основной поток дождался завершения других, используют специальную функцию ожидания. Это важно, чтобы программа не завершилась раньше времени.

Как линковаться на библиотеку `pthread`

Библиотека `pthread` реализует POSIX потоки. Чтобы её использовать, при компиляции нужно добавить флаг: `-pthread`

Он сообщает компилятору и компоновщику, что нужно подключить поддержку многопоточности.

Без мьютекса

Начало:

```
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ gcc -pthread mutex.c -o mutex
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ ./mutex
doing one thing
counter = 0
doing another thing
counter = 0
doing one thing
counter = 1
doing another thing
counter = 1
doing one thing
counter = 2
doing another thing
counter = 2
doing one thing
counter = 3
doing another thing
counter = 3
doing one thing
counter = 4
doing another thing
counter = 4
doing one thing
counter = 5
doing another thing
counter = 5
doing one thing
counter = 6
doing another thing
counter = 6
doing one thing
counter = 7
doing another thing
counter = 7
doing another thing
counter = 8
doing one thing
counter = 8
doing another thing
counter = 9
doing one thing
counter = 9
doing another thing
counter = 10
```

Конец

```
doing one thing
counter = 40
doing another thing
counter = 40
doing one thing
counter = 41
doing another thing
counter = 42
doing one thing
counter = 42
doing one thing
counter = 43
doing another thing
counter = 43
doing one thing
counter = 44
doing another thing
counter = 44
doing one thing
counter = 45
doing another thing
counter = 45
doing one thing
counter = 46
doing another thing
counter = 46
doing one thing
counter = 47
doing another thing
counter = 47
doing one thing
counter = 48
doing one thing
counter = 49
doing another thing
counter = 48
doing one thing
counter = 50
doing another thing
counter = 49
doing another thing
counter = 50
All done, counter = 51
```

С мьютексом

Начало:

```
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ gcc -pthread mutex.c -o mutex
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ ./mutex
doing one thing
counter = 0
doing one thing
counter = 1
doing one thing
counter = 2
doing one thing
counter = 3
doing one thing
counter = 4
doing one thing
counter = 5
doing one thing
counter = 6
doing one thing
counter = 7
doing one thing
counter = 8
doing one thing
counter = 9
doing one thing
counter = 10
doing one thing
counter = 11
doing one thing
counter = 12
doing one thing
counter = 13
doing one thing
counter = 14
doing one thing
counter = 15
doing one thing
counter = 16
doing one thing
counter = 17
doing one thing
counter = 18
doing one thing
counter = 19
doing one thing
counter = 20
```

Конец:

```
counter = 77
doing another thing
counter = 78
doing another thing
counter = 79
doing another thing
counter = 80
doing another thing
counter = 81
doing another thing
counter = 82
doing another thing
counter = 83
doing another thing
counter = 84
doing another thing
counter = 85
doing another thing
counter = 86
doing another thing
counter = 87
doing another thing
counter = 88
doing another thing
counter = 89
doing another thing
counter = 90
doing another thing
counter = 91
doing another thing
counter = 92
doing another thing
counter = 93
doing another thing
counter = 94
doing another thing
counter = 95
doing another thing
counter = 96
doing another thing
counter = 97
doing another thing
counter = 98
doing another thing
counter = 99
All done, counter = 100
```

Функции `dothing` и `doanotherthing` имеют по 50 итераций. Каждая из них выводит значение счетчика и увеличивает его на 1. В конце выполнения программы, по-хорошему, счетчик должен быть равен 100, но происходит гонка потоков. Оба потока пытаются одновременно считывать и изменять значение счетчика, что приводит к потере данных и в конечном итоге мы получаем число меньше. Использование мьютекса позволяет запретить одному из потоков работать со счетчиком, пока свою работу не завершит другой поток, что позволяет программе выполняться корректно. Стоит отметить, что использование мьютекса требует дополнительный ресурс компьютера, что увеличивает время выполнения программы.

Задание 2

Необходимые знания

1. POSIX threads: как создавать, как дожидаться завершения.
2. Как линковаться на библиотеку `pthread`
3. Как использовать мьютексы.

Написать программу для параллельного вычисления факториала по модулю `mod` ($k!$), которая будет принимать на вход следующие параметры (пример: `-k 10 --pnum=4 --mod=10`):

1. `k` - число, факториал которого необходимо вычислить.
2. `pnum` - количество потоков.
3. `mod` - модуль факториала

Для синхронизации результатов необходимо использовать мьютексы.

Мьютексы в POSIX Threads

Мьютекс (mutex — *mutual exclusion*, "взаимное исключение") — это механизм, который позволяет **только одному потоку** войти в критическую секцию в каждый момент времени. Он используется для **предотвращения состояния гонки** при доступе к общим данным.

Как использовать мьютекс (без кода)

1. **Создание мьютекса**
Перед использованием мьютекс нужно создать (инициализировать).
2. **Блокировка (lock)**
Перед входом в критическую секцию поток **захватывает** мьютекс. Если он уже занят другим потоком, текущий будет ждать.
3. **Разблокировка (unlock)**
После выхода из критической секции поток **освобождает** мьютекс, чтобы другие потоки могли продолжить.
4. **Удаление мьютекса**
После завершения работы с мьютексом его нужно уничтожить, чтобы освободить ресурсы.

Программу написал

```
Неверные параметры: k, pnum и mod должны быть положительными числами
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ ./factorial -k 5 -pnum 2 -mod 7
5! mod 7 = 1
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ ./factorial -k 5 -pnum 2 -mod 10000
5! mod 10000 = 120
○ @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $
```

Задание 3

Необходимые знания

1. Состояние deadlock

Напишите программу для демонстрации состояния deadlock.

```
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ touch dedlock.c
● @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ gcc -pthread deadlock.c -o deadlock
○ @ccurecc →/workspaces/os_lab_2019/lab5/src (master) $ ./deadlock
  Thread 1: Locked mutex1
  Thread 2: Locked mutex2
```

Deadlock (взаимная блокировка) в потоках — это ситуация, при которой **несколько потоков** находятся в состоянии ожидания ресурсов, занятых **друг другом**, и ни один из них не может продолжать выполнение.