

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра безопасности информационных систем (БИС)

Отчет по лабораторной работе №6
по дисциплине «Системное программирование»
Синхронизация потоков и процессов

Выполнили

Студент группы 745:

_____ В.В. Мащенко

«__» _____ 2019 г.

Принял

Доцент, к.т.н кафедры БИС

_____ А.С. Романов

«__» _____ 2019 г.

1. Введение

Изучить средства синхронизации потоков и процессов. Познакомиться с соответствующими функциями WinAPI и POSIX API. В процессе изучения основного материала познакомиться также с функциями отображения файлов на память и директивами препроцессора.

2. Ход работы

Таблица 2.1 – Вариант задания

ФИО	Образ ОС для Docker	Язык программирования	Программа
Мащенко В.В.	3	2	16

В соответствие с вариантом было получено задание

- изучить краткие теоретические сведения и лекционный материал по теме практического задания;
- используя Docker и соответствующий образ подготовить среду для разработки (уже готовы если сделаны предыдущие лабораторные работы);
- предусмотреть в варианте задания использование разделяемого ресурса. Описать какие из средств синхронизации и как могут быть применены для решения новой задачи;
- реализовать один (или несколько) вариантов синхронизации на языке C++ для Linux;
- реализовать один (или несколько) вариантов синхронизации средствами встроенных высокоуровневых возможностей языка программирования;
- сравнить возможности обоих подходов, сделать выводы.

2.1 Язык программирования C++

Для реализации задачи в соответствии с вариантом, был написан код на языке программирования C++.

В данной работе был немного изменен код для поточного программирования на C++, используя мьютексы.

Мьютекс - базовый элемент синхронизации (`<mutex>`).

Мьютекс представлен в 4 формах:

- `mutex`: обеспечивает базовые функции `lock()` и `unlock()` и не блокируемый метод `try_lock()`;
- `recursive_mutex`: может войти «сам в себя»;
- `timed_mutex`: в отличие от обычного мьютекса, имеет еще два метода: `try_lock_for()` и `try_lock_until()`;
- `recursive_timed_mutex`: это комбинация `timed_mutex` и `recursive_mutex`.

```

std::mutex g_lock;

void threadFunction()
{
    g_lock.lock();

    std::cout << "runner start " << std::this_thread::get_id() << std::endl;

    std::this_thread::sleep_for(std::chrono::seconds(rand() % 5));
    for (int i = 0; i < 10; i++) {
        int barrier = rand() % 2;
        if (barrier != 0) {
            sleep(4);
            std::cout << "barrier " << i << std::endl;
        }
        else {
            continue;
        }
    }

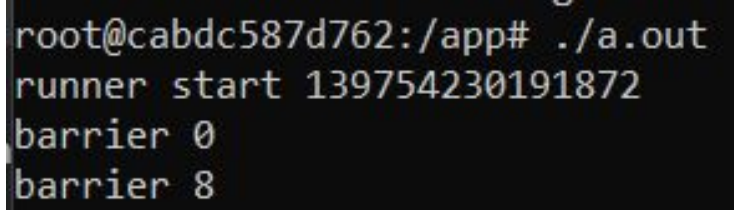
    std::cout << "runner finishes " << std::this_thread::get_id() << std::endl;

    g_lock.unlock();
}

int main()
{
    srand((unsigned int)time(0));
    std::thread t1(threadFunction);
    std::thread t2(threadFunction);
    std::thread t3(threadFunction);
    t1.join();
    t2.join();
    t3.join();
    return 0;
}

```

Рисунок 2.1 - Код программы

A terminal window with a black background and white text. The prompt is 'root@cabdc587d762:/app#'. The command './a.out' has been executed, resulting in four lines of output: 'runner start 139754230191872', 'barrier 0', and 'barrier 8'.

```
root@cabdc587d762:/app# ./a.out
runner start 139754230191872
barrier 0
barrier 8
```

Рисунок 2.2 - Результат работы

2.2 Язык программирования Go

```
package main

import (
    "fmt"
    "time"
    "math/rand"
    "strconv"
)

func runner1(c chan string) {
    for i := 0; i < 25; i++ {
        var barrier int = rand.Intn(2);
        if (barrier != 0) {
            c <- "barrier number: " + strconv.Itoa(i);
        }
    }
}

func runner2(c chan string) {
    for i := 0; i < 25; i++ {
        var barrier int = rand.Intn(2);
        if (barrier != 0) {
            c <- "(2) barrier number: " + strconv.Itoa(i);
        }
    }
}

func printer(c chan string) {
    for {
        msg := <- c
        fmt.Println(msg)
        time.Sleep(time.Second * 1)
    }
}

func main() {
    var c chan string = make(chan string)

    go runner1(c)
    go runner2(c)
    go printer(c)

    var input string
    fmt.Scanln(&input)
}
```

Рисунок 2.3 - Код программы

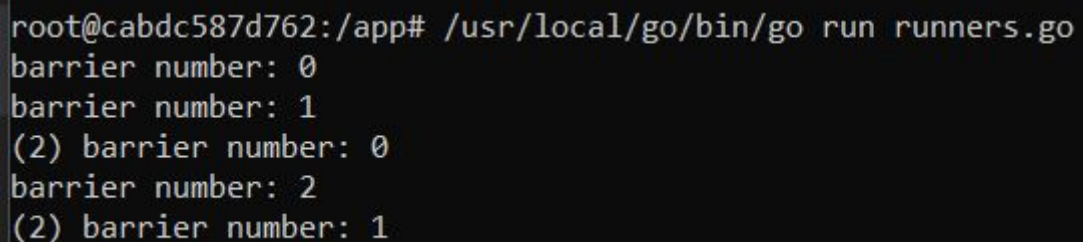
Обычно, при вызове функции, программа выполнит все конструкции внутри вызываемой функции, а только потом перейдет к следующей после вызова, строке. С горутиной программа немедленно перейдет к следующей строке, не дожидаясь, пока вызываемая функция завершится. Вот почему здесь присутствует вызов `Scanln`, без него программа завершится еще перед тем, как ей удастся вывести числа.

При запуске все горутины выполняются последовательно, а не одновременно, как ожидается. Поэтому необходимо добавить небольшую задержку функции с помощью функций `time.Sleep()` и `rand.Intn()`.

Каналы в языке `Golang` обеспечивают возможность общения нескольких горутин друг с другом.

Тип канала представлен ключевым словом `chan`, за которым следует тип, который будет передаваться по каналу. Оператор `<-` (стрелка влево) используется для отправки и получения сообщений по каналу.

Данное использование каналов позволяет синхронизировать две горутины. Программа будет выводить сообщение от двух потоков о барьере в поле равным 25 через блокирующее поведение.



```
root@cabdc587d762:/app# /usr/local/go/bin/go run runners.go
barrier number: 0
barrier number: 1
(2) barrier number: 0
barrier number: 2
(2) barrier number: 1
```

Рисунок 2.4 - Результат работы

2.3 Docker

Соберем наш образ с помощью докера в соответствии с заданием.

```
FROM debian:latest

RUN mkdir /app
COPY app/ /app

RUN apt-get update -qq && apt-get install -y build-essential gcc
ADD https://storage.googleapis.com/golang/go1.11.linux-amd64.tar.gz /opt/go.tar.gz
RUN tar -C /usr/local/ -xvf /opt/go.tar.gz
ENV PATH=$PATH:/usr/local/go/bin

RUN cd /app && gcc -pthread lab5.c -o lab6
```

Рисунок 2.5 - Код Dockerfile'a

3. Заключение

В результате работы было произведено ознакомление с потоками, с основными функциями WinAPI для работы с потоками в Windows и библиотекой Pthread для работы с потоками в Linux.