МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ВЕДЕРАЦИИ
федеральное государственное автономное образовательное
учреждение высшего образования
«Санкт-Петербургский Государственный университет Аэрокосмическо

Кафедра №14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

доц.,к.т.н.		К.А. Курицин
должность	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2
По курсу: «Технология программирования».

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГРУППЫ 1441		А.А. Протасов
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2016

1. Постановка задачи

Реализовать класс синхронизации семафор. Класс инкапсулирует работу объекта синхронизации, который содержит счетчик между нулем и заданным максимальным значением. Значение счетчика увеличивается каждый раз, когда поток завершает ожидание освобождения семафора, и уменьшается, когда поток освобождает семафор. В случае, если значение счетчика достигает максимального значения, потоки ожидают освобождение семафора. Если указано максимальное значение счетчика, равное 1, то семафор функционально должен вести себя как критическая секция (critical section).

2. Листинги

```
#include <pthread.h>
#include <iostream>
#include <cstddef>
#include <unistd.h>
#include <sys/types.h>
#include <sys/syscall.h>

using std::cout;
using std::endl;
using std::cin;

class Event{
    pthread_cond_t cond;
    pthread_mutex_t mutex;
    bool triggered;
public:
    Event(){
        pthread_cond_init(&cond, NULL);
        pthread_mutex_init(&mutex, NULL);
        cout << endl << "PthreadCreate ID: " << syscall(SYS_gettid) << endl;
    }
    ~Event(){
        cout << endl << "PthreadDestroy ID: " << syscall(SYS_gettid) << endl;
        pthread_cond_destroy(&cond);
        pthread_mutex_destroy(&mutex);
    }
    void set(){
        pthread_mutex_lock(&mutex);
        pthread_cond_signal(&cond);
        cout << endl << "ThreadSignal ID: " << syscall(SYS_gettid) << endl;
        triggered = true;
        pthread_mutex_unlock(&mutex);
    }
    void reset(){
        pthread_mutex_lock(&mutex);
        triggered = false;
        pthread_mutex_unlock(&mutex);
    }
    void wait(){
        cout << endl << "WaitThread ID: " << syscall(SYS_gettid) << endl;
        pthread_mutex_lock(&mutex);
```

```
42          pthread_cond_wait(&cond, &mutex);
43          pthread_mutex_unlock(&mutex);
44      }
45  };
46
47  class Lock{
48  public:
49      virtual void lock() = 0;
50      virtual void unlock() = 0;
51      virtual ~Lock() {}
52  };
53
54  class CriticalSection : public Lock{
55  private:
56      pthread_mutex_t CS;
57  public:
58      CriticalSection(){
59          pthread_mutex_init(&CS, NULL);
60      }
61      ~CriticalSection(){
62          pthread_mutex_destroy(&CS);
63      }
64      void lock(){
65          pthread_mutex_lock(&CS);
66      }
67      void unlock(){
68          pthread_mutex_unlock(&CS);
69      }
70  };
71
72  class Semaphore : public Lock{
73  private:
74      size_t count_max;
75      size_t count;
76      CriticalSection *tCS;
77      Event *event;
78  public:
79      Semaphore(int count_max) : count_max(count_max), count(0){
80          tCS = new CriticalSection;
81          event = new Event;
82      }
83      ~Semaphore(){
84          delete event;
85          delete tCS;
86      }
87      void lock(){
88          tCS->lock();
89          if(count++ < count_max){
90              tCS->unlock();
91              return;
92          }
93          tCS->unlock();
94          event->wait();
95      }
```

```
96      void unlock(){
97              tCS->lock();
98              if(count-- >= count_max){
99                  event->set();
100             }
101             tCS->unlock();
102     }
103 };
104
105 Semaphore *SEMAPHORE;
106 int number = 0;
107
108 pthread_mutex_t t;
109
110 void* foo(void*){
111     SEMAPHORE->lock();
112     pthread_mutex_lock(&t);
113     ++number;
114     cout << endl << "Job " << number << " started" << endl;
115     sleep(2);
116     cout << endl << "Job " << number << " finished" << endl;
117     SEMAPHORE->unlock();
118     pthread_mutex_unlock(&t);
119     return NULL;
120 }
121
122 int main(){
123     pthread_t tid[5];
124     SEMAPHORE = new Semaphore(5);
125
126     pthread_mutex_init(&t, NULL);
127     for (size_t i = 0; i < 5; i++){
128         pthread_create(&(tid[i]), NULL, &foo, NULL);
129     }
130     for(size_t i = 0; i < 5; i++)
131         pthread_join(tid[i], NULL);
132     pthread_mutex_destroy(&t);
133     delete SEMAPHORE;
134     SEMAPHORE = NULL;
135     return 0;
136 }
```