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

КАФЕДРА 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

асс.		С.В. Осмоловский
должность	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

МЕХАНИЗМЫ СИНХРОНИЗАЦИИ И ВЗАИМОДЕЙСТВИЯ (IPC):
МЕЖПОТОКОВАЯ СИНХРОНИЗАЦИЯ.
по курсу: СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

РАБОТУ ВЫПОЛНИЛ
СТУДЕНТ ГР. 1441

	А.А. Протасов
подпись, дата	инициалы, фамилия

Санкт-Петербург 2017

1. Постановка задачи

Решить одну из классических задач синхронизации в информатике: с классическим или модифицированным условием. Для решения задачи необходимо использовать определенный механизм синхронизации, предоставляемой QNX: мьютексы, семафоры, спинлоки, блокировки чтения-записи, условные переменные, барьеры, или любой (любые) на собственное усмотрение, если в задании не указан конкретный механизм для решения. В данном случае для решения запрещается использовать способы, которые указаны в других вариантах с данной задачей.

Вариант 5 Задача о читателях-писателях. Приоритет писателя. 10 читателей, 5 писателей.

2. Пояснение

Задача о читателях-писателях — одна из важнейших задач параллельного программирования. Формулируется она так:

“Есть область памяти, позволяющая чтение и запись. Несколько потоков имеют к ней доступ, при этом одновременно могут читать сколько угодно потоков, но писать — только один. Как обеспечить такой режим доступа?”

2.1. Первая задача о читателях-писателях (приоритет читателя)

Формулировка:

“Пока память открыта на чтение, давать читателям беспрепятственный доступ. Писатели могут ждать сколько угодно.”

2.2. Вторая задача о читателях-писателях (приоритет писателя)

Формулировка:

“Как только появился хоть один писатель, читателей больше не пускать. При этом читатели могут простаивать.”

2.3. Третья задача о читателях-писателях (честное распределение ресурсов)

Формулировка:

“Не допускать простоев. Другими словами: независимо от действий других потоков, читатель или писатель должен пройти барьер за конечное время.”

3. Листинги

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sched.h>

#define READERS 10
#define WRITERS 5

struct sched_param  param;
struct sched_param  main_param;
pthread_mutex_t     read_mutex      = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t     write_mutex     = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t     readTry_mutex   = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t     resource_mutex  = PTHREAD_MUTEX_INITIALIZER;
pthread_attr_t      attribute_writer;
pthread_attr_t      attribute_reader;
pthread_t           reader_thread[READERS];
pthread_t           writer_thread[WRITERS];
static int          readcount;
static int          writecount;
static int          value;

void* writers();
void* readers();

void proc_rt(int prio){
    int min  = sched_get_priority_min(SCHED_RR);
    int max  = sched_get_priority_max(SCHED_RR);
    prio = prio > min ? prio : prio < max ? max : prio;
    main_param.sched_priority = prio;
    sched_setscheduler(pthread_self(), SCHED_RR, &main_param);
}

int main(){
    int ret;
    printf("Readers-writers problem solve algorithm - writers-preference\n");

    proc_rt(50);

    pthread_attr_init(&attribute_writer);
    pthread_attr_init(&attribute_reader);
    ret = pthread_attr_getschedparam(&attribute_reader, &param);
    param.sched_priority++;
    ret = pthread_attr_setschedparam(&attribute_writer, &param);

    for (int i = 0; i < WRITERS; i++)
        pthread_create(&writer_thread[i], &attribute_writer, writers, NULL);
    for (int i = 0; i < READERS; i++)
        pthread_create(&reader_thread[i], &attribute_reader, readers, NULL);

    for (int i = 0; i < WRITERS; i++)
```

```
53          pthread_join(writer_thread[i], NULL);
54      for (int i = 0; i < READERS; i++)
55          pthread_join(reader_thread[i], NULL);
56
57      return 0;
58  }
59
60  void* readers(){
61      pthread_mutex_lock(&readTry_mutex);
62      pthread_mutex_lock(&read_mutex);
63      readcount++;
64      if (readcount == 1)
65          pthread_mutex_lock(&resource_mutex);
66      pthread_mutex_unlock(&read_mutex);
67      pthread_mutex_unlock(&readTry_mutex);
68
69      printf("\033[35;1m Im reader number: %d\033[0m\n", readcount);
70      sleep(1);
71
72      pthread_mutex_lock(&read_mutex);
73      readcount--;
74      if (readcount == 0)
75          pthread_mutex_unlock(&resource_mutex);
76      pthread_mutex_unlock(&read_mutex);
77
78      pthread_exit(0);
79  }
80
81  void* writers(){
82      pthread_mutex_lock(&write_mutex);
83      writecount++;
84      if (writecount == 1)
85          pthread_mutex_lock(&readTry_mutex);
86      pthread_mutex_unlock(&write_mutex);
87
88      pthread_mutex_lock(&resource_mutex);
89      printf("\033[31;1m Im writer number: %d\033[0m\n", writecount);
90      pthread_mutex_unlock(&resource_mutex);
91      sleep(1);
92
93      pthread_mutex_lock(&write_mutex);
94      writecount--;
95      if (writecount == 0)
96          pthread_mutex_unlock(&readTry_mutex);
97      pthread_mutex_unlock(&write_mutex);
98
99      pthread_exit(0);
100 }
```

4. Выводы

```
toshiki@mangaka~/D/R/lab2-readers-writers_problem> ./main
Readers-writers problem solve algorithm - writers-preference
 I'm writer number: 1
 I'm writer number: 2
 I'm writer number: 3
 I'm writer number: 4
 I'm writer number: 5
 I'm reader number: 1
 I'm reader number: 2
 I'm reader number: 3
 I'm reader number: 4
 I'm reader number: 5
 I'm reader number: 6
 I'm reader number: 7
 I'm reader number: 8
 I'm reader number: 9
 I'm reader number: 10
```