МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное учреждение высшего образования
«Чувашский государственный университет И.Н. Ульянова»
Факультет информатики и вычислительной техники
Кафедра вычислительной техники

Системное операционные системы
Лабораторная работа 1
Вариант 5

Выполнил:

Студент группы ИВТ-41-20
Галкин Д.С.

Проверил:

Старший преподаватель
Первов С.Г.

Чебоксары, 2024

Цель работы:

Получение навыков многопоточного программирования.

Задание:

Проблема читателей и писателей, моделирующая доступ к базе данных. К базе данных (представленную в данном случае обычным файлом, пытаются получить доступ множество процессов. Можно разрешить одновременное считывание данных из базы, но если процесс записывает информацию в базу, доступ остальных процессов должен быть прекращен, даже доступ на чтение. Чтобы избежать голодания, необходимо сделать следующее: если пишущий процесс ждет доступа к базе, новый читающий процесс доступа не получает, а становится в очередь за пишущим процессом. Теперь пишущему процессу нужно подождать, пока базу данных покинут все читающие процессы, но не нужно пропускать вперед читающие процессы, пришедшие в базу после него. Можно реализовать и другое решение – предоставлять пишущим процессам более высокий приоритет. А лучше оба.

Ход работы:

Функция потока: reader

```
void *reader(void *arg) {
    while (1) {
        pthread_mutex_lock(&priority_mutex);
        while (write_count > 0 || (num_writers_waiting > 0 && read_count >=
patience_threshold_read)) {
            pthread_cond_wait(&read_priority, &priority_mutex);
        }
        read_count++;
        pthread_mutex_unlock(&priority_mutex);

        pthread_mutex_lock(&read_mutex);
        num_readers++;
        if (num_readers == 1) {
            pthread_mutex_lock(&write_mutex);
        }
        pthread_mutex_unlock(&read_mutex);

        // Чтение
        printf("Читатель %d читает в библиотеке\n", (int)arg);
        usleep(rand() % 1000000);
        printf("---> Читатель %d прочитал книгу в библиотеке\n", (int)arg);

        pthread_mutex_lock(&read_mutex);
        num_readers--;
        if (num_readers == 0) {
            pthread_mutex_unlock(&write_mutex);
        }
        pthread_mutex_unlock(&read_mutex);

        pthread_mutex_lock(&priority_mutex);
        read_count--;
        if (num_writers_waiting > 0 && read_count < patience_threshold_read) {
            pthread_cond_signal(&write_priority);
        }
        pthread_mutex_unlock(&priority_mutex);

        usleep(rand() % 1000000);
    }
}
```

Функция потока: writer

```
void *writer(void *arg) {
    while (1) {
        pthread_mutex_lock(&priority_mutex);
        num_writers_waiting++;
        while (num_readers > 0 || write_count > 0) {
            pthread_cond_wait(&write_priority, &priority_mutex);
        }
        num_writers_waiting--;
        write_count++;
        pthread_mutex_unlock(&priority_mutex);

        pthread_mutex_lock(&write_mutex);
        // Запись
        printf("Писатель %d пишет в библиотеке\n", (int)arg);
        usleep(rand() % 3000000);
        printf("---> Писатель %d закончил писать в библиотеке\n", (int)arg);
        pthread_mutex_unlock(&write_mutex);

        pthread_mutex_lock(&priority_mutex);
        write_count--;
        if (write_count == 0) {
            if (num_writers_waiting > 0 && read_count >= patience_threshold_read) {
                pthread_cond_signal(&write_priority);
            } else {
                pthread_cond_broadcast(&read_priority);
            }
        }
        pthread_mutex_unlock(&priority_mutex);

        usleep(rand() % 3000000);
    }
}
```

main.c

```c
#include "model_thread.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


// #define NUM_READERS 5
// #define NUM_WRITERS 5

pthread_mutex_t write_mutex;
pthread_mutex_t read_mutex;
pthread_mutex_t priority_mutex;
pthread_cond_t write_priority;
pthread_cond_t read_priority;

int num_readers = 0;
int num_writers_waiting = 0;
int read_count = 0;
int write_count = 0;
int patience_threshold_read = 3;
int patience_threshold_write = 10;

void main() {
    // pthread_t readers[NUM_READERS], writers[NUM_WRITERS];
    char queues[] = "rrwwrrrrrrrrrrwwwwrrrrwwwwrrrrwwwwrrrrwwwwrrrrwww";
    int queues_length = strlen(queues);
    pthread_t thread_queues[queues_length];

    pthread_mutex_init(&write_mutex, NULL);
    pthread_mutex_init(&read_mutex, NULL);
    pthread_mutex_init(&priority_mutex, NULL);
    pthread_cond_init(&write_priority, NULL);
    pthread_cond_init(&read_priority, NULL);

    // for (int i = 0; i < NUM_READERS; i++) {
    //     pthread_create(&readers[i], NULL, reader, (void *)(intptr_t)i);    // }
//    // for (int i = 0; i < NUM_WRITERS; i++) {    //     pthread_create(&writers[i],
NULL, writer, (void *)(intptr_t)i);    // }
    for (int i = 0; i < queues_length; i++) {
        // printf("xxxx> Создался поток %c \n", queues[i]);
        if (queues[i] == 'r')
            pthread_create(&thread_queues[i], NULL, reader, (void *)(intptr_t)i);
        else
            pthread_create(&thread_queues[i], NULL, writer, (void *)(intptr_t)i);
    }

    // for (int i = 0; i < NUM_READERS; i++) {
    //     pthread_join(readers[i], NULL);    // }    //    // for (int i = 0; i <
NUM_WRITERS; i++) {    //     pthread_join(writers[i], NULL);    // }
    for (int i = 0; i < queues_length; i++) {
        pthread_join(thread_queues[i], NULL);
    }
```

```
    pthread_mutex_destroy(&write_mutex);
    pthread_mutex_destroy(&read_mutex);
    pthread_mutex_destroy(&priority_mutex);
    pthread_cond_destroy(&write_priority);
    pthread_cond_destroy(&read_priority);
}
```

Вывод:

В ходе выполнения лабораторной работы я получил навыки многопоточного программирования.