Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК

Расчетно-графическое задание
по дисциплине
«Операционные системы реального времени»

Выполнил: студент 4 курса
ИВТ, гр. ИП-813
Бурдуковский И.А.
Проверил: Преподователь кафедры ПМиК
Белевцова Екатерина Андреевна
.

Новосибирск 2021

Оглавление

Задание

1. Сравните время запуска (создания) нити и время активизации с помощью семафора заранее созданной нити.

2. Функции типа fwrite(), работающие через структуру FILE, используют внутреннюю буферизацию данных. Анализируя время выполнения функций, определите размер буфера

Выполнение

1. Для выполнения первого задания была реализована программа с использованием функций pthread_create, sem_init, sem_wait, sem_post, для отслеживания времени был выбран ClockCycles.

int pthread_create(pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine)(void*), void* arg) –создаёт нить на которой выполняется указанная при создании функция ;

int sem_init(sem_t * sem, int pshared, unsigned value) – инициализирует семафор для возможности синхронизации нитей с указанным значением value;

int sem_post (sem_t* sem) – увеличивает семафор на 1 и выводит одну нить из ожидания, если находится в очереди;

int sem_wait (sem_t* sem) – уменьшает семафор на 1. Однако: если семафор = 0, то нить блокируется до тех пор, пока кто-то не увеличит его. Если несколько нитей вызывают sem_wait на нулевом семафоре, то они ставятся в очередь, упорядоченную по приоритетам.

int ClockCycles() – функция из библиотеки sys/neutrino.h, возвращает текущее значение 64-битного счетчика циклов процессора. Чтобы вычислить количество прошедших секунд необходимо сначала вычислить количество циклов в секунду у системы с помощью SYSPAGE_ENTRY(qtime)->cycles_per_sec и уже поделить наши циклы на это количество.

Замер был произведен на создание нити с пустой функцией void *testThread(void *args). Также были созданы нити с функцией void *semaphoreThread(void *args), которая сразу при создании вставала в очередь ожидания с помощью sem_wait().

Второй замер был произведён при отработке функции sem_post(), которая выводила по очереди все нити из ожидания.

Как показали результаты замеров и сравнения времени: активизация с помощью семафора заранее созданной нити оказалась быстрее чем создание новой нити.

В зависимости от количества повторений создания и активизаций нитей увеличивалась точность и стабильность замеров между повторными запусками программ.

2. Для выполнения второго задания в программе также использовалась функция ClockCycles() для замера времени работы и функция fwrite.

size_t fwrite(const void *buf, size_t size, size_t count, FILE *stream) – функция записывает count объектов (каждый объект по size символов в длину) в поток - в нашем случае это указатель на открытый файл. В функции имеется внутренний буфер, в который первоначально передаются записываемые символы. Это позволяет сократить количество обращений к файлу и соответственно затрачиваемое на это время. Как только буфер заполняется – срабатывает прерывание, во время которого всё содержимое буфера сохраняется в файл.

В программе циклом производится замер времени записи одного символа char в файл. В тот момент, когда буфер полностью заполнится – сработает прерывание с записью, количество замеренного времени будет гораздо больше того, которое затрачивается для простой записи символа в буфер. В таком случае нам станет известен размер буфера, равный номеру итерации в цикле.

Результаты работы программ

```
# ./file1
Cycles per sec: 3594793400

Thread init: 896107.00 cycles
Thread init: 0.000249 secs

Semaphore: 164427.00 cycles
Semaphore: 0.000046 secs
# _
```

```
# ./file2
fwrite BUFSIZ: 1024
Cycles per sec: 3594793400

Analyzed buffer size: 1024
Control test: 548503.00 cycles
Control test: 0.0001525826 secs
# _
```

Листинг

File1.cpp

```
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <stdlib.h>
#include <inttypes.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>
#include <pthread.h>
#include <semaphore.h>
#include <errno.h>

using namespace std;

struct timespec start, stop;
sem_t* sem;
pthread_t last_threadId;
int thread_num = 1;

void *testThread(void *args){}

void *semaphoreThread(void *args){
    sem_wait(sem);
    sem_post(sem);
}

int main(){
    uint64_t cycles_per_sec = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    cout <<"Cycles per sec: " << cycles_per_sec << endl;
    unsigned long long t1, t2, d;
    t1 = ClockCycles();
    clock_gettime(CLOCK_REALTIME, &start);
    for(int i=0; i < thread_num; i++){
        pthread_create(0, 0, testThread, 0);
    }
    t2 = ClockCycles();
    d = t2-t1;

    printf("\nThread init: %.2f cycles\n", (double)d/thread_num);
    printf("Thread init: %f secs\n", (double)d/thread_num/cycles_per_sec);

    //Semaphore
    sem = new sem_t;
    int init_result = sem_init(sem, 0, 0);
    if(init_result != 0){
        printf("Semaphore init error: %d\n", errno);
        return 0;
    }
    for(int i=0; i < thread_num; i++){
        pthread_create(&last_threadId, 0, semaphoreThread, 0);
    }

    t1 = ClockCycles();
    sem_post(sem);
    pthread_join(last_threadId, 0);
    t2 = ClockCycles();

    d = t2-t1;
    printf("\nSemaphore: %.2f cycles\n", (double)d/thread_num);
    printf("Semaphore: %f secs\n", (double)d/thread_num/cycles_per_sec);

    return 0;
}
```

File2.cpp

```
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <stdlib.h>
#include <inttypes.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>
#include <pthread.h>
#include <semaphore.h>
#include <errno.h>

using namespace std;

sem_t* sem;
pthread_t last_threadId;
int thread_num = 1;

void *testThread(void *args){}

void *semaphoreThread(void *args){
    sem_wait(sem);
    sem_post(sem);
}
```

```
int main(){
    cout<<"fwrite BUFSIZ: " << BUFSIZ << endl;
    uint64_t cycles_per_sec = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    cout <<"Cycles per sec: " << cycles_per_sec << endl;

    char f[test_repeats];
    for(int i = 0; i < test_repeats; i++){
        strcat(f, "a");
    }

    if((fp = fopen("test.txt","wb")) == NULL){
        printf("Cannot open file.\n");
        exit(1);
    }

    for(int i = 1; i < test_repeats; i++){
        start = ClockCycles();
        fwrite(&f, sizeof(char), 1, fp);
        stop = ClockCycles();
        dif = stop - start;
        cycles = (double)dif;

        if(last_cycles != 0 && last_cycles * 10 < cycles){
            printf("\nAnalyzed buffer size: %d\n", i-1);
            printf("Control test: %.2f cycles\n", cycles);
            printf("Control test: %.10f secs\n", cycles/cycles_per_sec);
            break;
        }else{
            last_cycles = cycles;
        }

    }
    return 0;
}
```

Вывод

В рамках данного курса я узнал о операционных системах реального времени и научился работать в одной из них, QNX, узнал о принципе системы, чем она отличается от других систем.

В процессе выполнения расчетно-графического задания я углубил свои теоретические знания об ОСРВ, в частности о QNX. Больше узнал об алгоритме планирования и распределения ресурсов системы. Узнал о том, как измерить время работы части программы в QNX.

А также благодаря практической реализации подтвердил известную мне теорию, касающуюся предложенных 2ух расчетно-графических заданий.