

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ
к лабораторной работе №5
на тему

УПРАВЛЕНИЕ ПОТОКАМИ, СРЕДСТВА СИНХРОНИЗАЦИИ

Студент
Преподаватель

О. Л. Дайнович
Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

| | |
|--|---|
| 1 Цель работы | 3 |
| 2 Теоретические сведения | 4 |
| 3 Полученные результаты | 5 |
| Заключение | 7 |
| Список использованных источников | 8 |
| Приложение А (обязательное) Листинг кода | 9 |

1 ЦЕЛЬ РАБОТЫ

Изучение подсистемы потоков (pthread), основных особенностей функционирования и управления, средств взаимодействия потоков. Практическое проектирование, реализация и отладка программ с параллельными взаимодействующими (конкурирующими) потоками.

Требуется разработать многопоточную программу, реализующую операции ввода-вывода (дисковый файл, открытый для разделяемого доступа).

Количество потоков задается пользователем. Количество потоков выбирается не слишком большое, чтобы оставалось удобным для отображения и не провоцировало перегрузку системы.

Результат – сведения о времени выполнения для конкретной конфигурации, минимальный протокол выполнения.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

POSIX Threads — стандарт POSIX-реализации потоков (нитей) выполнения. Стандарт POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995) определяет API для управления потоками, их синхронизации и планирования.

Реализации данного API существуют для большого числа UNIX-подобных ОС (GNU/Linux, Solaris, FreeBSD, OpenBSD, NetBSD, OS X), а также для Microsoft Windows и других ОС.

Библиотеки, реализующие этот стандарт (и функции этого стандарта), обычно называются Pthreads (функции имеют приставку «pthread_»).

В конце 1980-х и начале 1990-х было несколько разных API, но в 1995 г. POSIX.1c стандартизовал потоки POSIX, позже это стало частью спецификаций SUSv3. В наше время многоядерные процессоры проникли даже в настольные ПК и смартфоны, так что у большинства машин есть низкоуровневая аппаратная поддержка, позволяющая им одновременно выполнять несколько потоков. В былые времена одновременное исполнение потоков на одноядерных ЦПУ было лишь впечатляюще изобретательной, но очень эффективной иллюзией.

Pthreads определяет набор типов и функций на Си.

В традиционном Unix API код последней ошибки errno является глобальной int переменной. Это однако не годится для программ с множественными нитями исполнения. В ситуации, когда вызов функции в одном из исполняемых потоков завершился ошибкой в глобальной переменной errno, может возникнуть состояние гонки из-за того, что и остальные потоки могут в данный момент проверять код ошибки и оконфузиться. В Unix и Linux эту проблему обошли тем, что errno определяется как макрос, задающий для каждой нити собственное изменяемое lvalue. [1]

Создание потока происходит с помощью функции `pthread_create(pthread_t *tid, const pthread_attr_t *attr, void*(*function)(void*), void* arg)`, где: `tid` - идентификатор потока, `attr` - параметры потока (NULL - атрибуты по умолчанию, подробности в man), `function` - указатель на потоковую функцию, в нашем случае `threadFunc` и `arg` - указатель на передаваемые данные в поток.

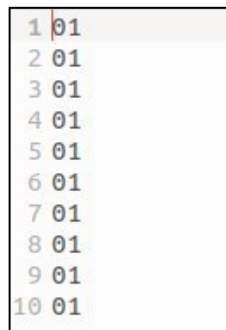
Функция `pthread_join` ожидает завершения потока `thread`. Второй параметр этой функции - результат, возвращаемый потоком. [2]

3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В результате выполнения лабораторной работы была разработана программа, реализующая многопоточную запись данных в файл и последующий анализ эффективности работы этой записи.

Основной код написан на языке C. Bash скрипт собирает программу, состоящую из файла main.cpp, с помощью файла Makefile и выполняет код.

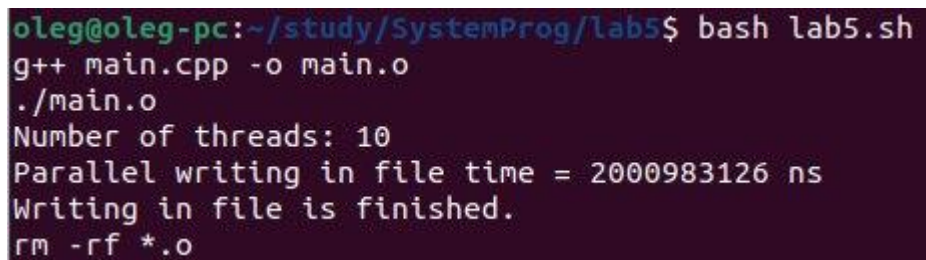
На вход программы дается число потоков для выполнения записи в файл. Сначала файл очищается, а потом наполняется новыми данными (рисунок 1).



| | |
|----|----|
| 1 | 01 |
| 2 | 01 |
| 3 | 01 |
| 4 | 01 |
| 5 | 01 |
| 6 | 01 |
| 7 | 01 |
| 8 | 01 |
| 9 | 01 |
| 10 | 01 |

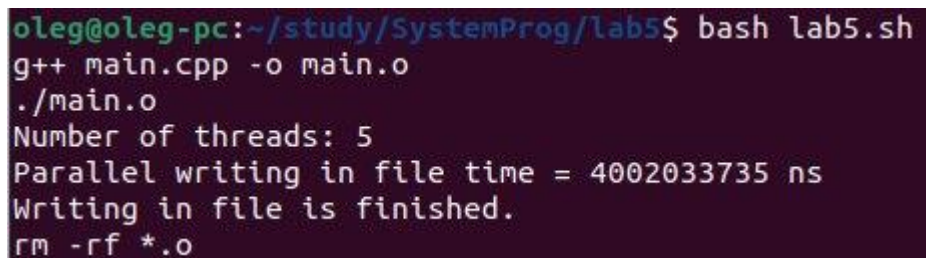
Рисунок 1 – Запись в файл (10 потоков)

После завершения программы в консоль выводятся данные о скорости проведения этой записи. Например, при количестве потоков равном 10, скорость записи в 2 раза больше, чем при выполнении данной операции с выделением пяти потоков (рисунки 2, 3).



```
oleg@oleg-pc:~/study/SystemProg/lab5$ bash lab5.sh
g++ main.cpp -o main.o
./main.o
Number of threads: 10
Parallel writing in file time = 2000983126 ns
Writing in file is finished.
rm -rf *.o
```

Рисунок 2 – Время записи при использовании 10 потоков



```
oleg@oleg-pc:~/study/SystemProg/lab5$ bash lab5.sh
g++ main.cpp -o main.o
./main.o
Number of threads: 5
Parallel writing in file time = 4002033735 ns
Writing in file is finished.
rm -rf *.o
```

Рисунок 3 – Время записи при использовании 5 потоков

Соответственно, при использовании только двух потоков, программа будет выполнена в 5 раз медленнее, чем при записи с выделением 10 потоков (рисунок 4).

```
oleg@oleg-pc:~/study/SystemProg/lab5$ bash lab5.sh
g++ main.cpp -o main.o
./main.o
Number of threads: 2
Parallel writing in file time = 10003508277 ns
Writing in file is finished.
rm -rf *.o
```

Рисунок 4 – Время записи при использовании 2 потоков

Сам результат записи в файл также будет отличаться (рисунок 5).

| | |
|---|------------|
| 1 | 0123456789 |
| 2 | 0123456789 |

Рисунок 1 – Запись в файл (2 потока)

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были изучены подсистемы потоков (pthread), основные особенности функционирования и управления, средства взаимодействия потоков. Практическое проектирование, реализация и отладка программ с параллельными взаимодействующими (конкурирующими) потоками.

Также была разработана программа, реализующая многопоточную запись данных в файл и последующий анализ эффективности работы этой записи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Pthreads: Потоки в русле POSIX [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/326138/> – Дата доступа: 24.03.2024.

[2] Программирование С в Linux — потоки pthreads [Электронный ресурс]. – Режим доступа <https://tetraquark.ru/archives/47> – Дата доступа: 24.03.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

Листинг 1 – Файл lab5.sh

```
#!/usr/bin/bash
make
./build/main
```

Листинг 2 – Файл makefile

```
all: compile run clean

compile:
    g++ main.cpp -o main.o

run: main.o
    ./main.o

clean:
    rm -rf *.o
```

Листинг 3 – Файл main.cpp

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

#define NUM_THREADS 2
#define FILENAME "output.txt"
#define BILLION 1000000000L
#define SYMBOL_NUM 20

void *thread_function(void *arg) {
    FILE *file = fopen(FILENAME, "a");
    if (file == NULL) {
        printf("Error in file opening: %s\n", FILENAME);
        pthread_exit(NULL);
    }

    for(int i = 0; i < SYMBOL_NUM / NUM_THREADS; i++){
        fprintf(file, "%d", i);
        sleep(1);
    }

    fprintf(file, "\n");

    fclose(file);
    pthread_exit(NULL);
}

int main() {
    FILE* file = fopen(FILENAME, "w");
```

```

if (file == NULL) {
    printf("Error in file opening: %s\n", FILENAME);
    exit(-1);
}
fclose(file);

if (NUM_THREADS > SYMBOL_NUM || SYMBOL_NUM % NUM_THREADS != 0) {
    printf("Error. Number of threads is invalid.\n");
    exit(-1);
}

pthread_t threads[NUM_THREADS];

int result;

unsigned long long diff;
struct timespec start, end;

clock_gettime(CLOCK_MONOTONIC_RAW , &start);

for (int i = 0; i < NUM_THREADS; i++) {
    result = pthread_create(&threads[i], NULL, thread_function, NULL);
    if (result) {
        printf("Error in thread opening: %d\n", result);
        exit(-1);
    }
}

for (int i = 0; i < NUM_THREADS; i++) {
    result = pthread_join(threads[i], NULL);
    if (result) {
        printf("Error in thread closing: %d\n", result);
        exit(-1);
    }
}

clock_gettime(CLOCK_MONOTONIC_RAW , &end);
diff = BILLION * (end.tv_sec - start.tv_sec) + end.tv_nsec - start.tv_nsec;
printf("Number of threads: %d \n", NUM_THREADS);
printf("Parallel writing in file time = %lld ns\n", diff);
printf("Writing in file is finished.\n");

return 0;
}

```