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

Отчет по РГР

Выполнил:

студент ИВТ,

гр. ИП-014

«\_\_» \_\_\_\_\_\_\_\_\_ 2023 г. / Обухов А.И./

(подпись)

Проверил:

Профессор кафедры ПМиК

«\_\_» \_\_\_\_\_\_\_\_\_ 2023 г. / Милешко А.В./

(подпись)

г. Новосибирск 2023

ЗАДАНИЕ

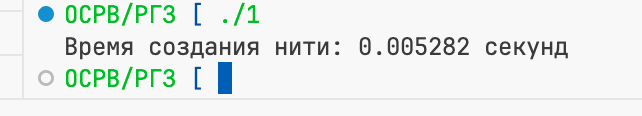
1. Основные характеристики системы. Сравните время запуска (создания) нити и время активизации с помощью семафора заранее созданной нити.

2. Особенности реализации операционной системы и аппаратуры. Функция создания процесса fork() может иметь недокументированные особенности.В каком режиме (разделяемом или частном) наследуются объекты, созданные с помощью malloc() и new?

ВЫПОЛНЕНИЕ РАБОТЫ

**Задание 1.**

Результат:



Исходный код:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <time.h>

#include <unistd.h>

// Глобальные переменные

pthread\_t thread;

sem\_t semaphore;

clock\_t start\_time;

clock\_t end\_time;

// Функция, которую выполнит нить

void\* worker\_function(void\* arg) {

// Засекаем время создания нити

start\_time = clock();

sem\_post(&semaphore); // Активизация семафора

return NULL;

}

int main() {

// Инициализация семафора

sem\_init(&semaphore, 0, 0);

// Инициализация потока

if (pthread\_create(&thread, NULL, worker\_function, NULL) != 0) {

perror("pthread\_create");

return 1;

}

// Ждем, пока семафор будет активирован

sem\_wait(&semaphore);

// Засекаем время активизации семафора

end\_time = clock();

double elapsed\_time\_create = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

printf("Время создания нити: %lf секунд\n", elapsed\_time\_create);

// Ожидание завершения нити

if (pthread\_join(thread, NULL) != 0) {

perror("pthread\_join");

return 1;

}

sem\_destroy(&semaphore); // Освобождение ресурсов семафора

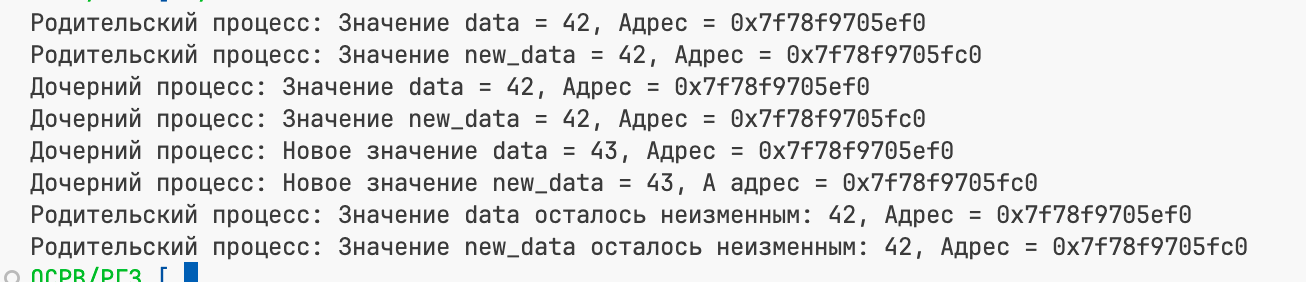
return 0;

}

**Задание 2.**

Объекты при использовании fork() наследуются в частном режиме

Результат:



Исходный код:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

int \*data = (int \*)malloc(sizeof(int)); // Выделяем динамическую память

int \*new\_data = new int;

if (data == NULL) {

perror("malloc");

return 1;

}

\*data = 42;

\*new\_data = 42;

pid\_t pid = fork(); // Создаем дочерний процесс

if (pid == -1) {

perror("fork");

return 1;

}

if (pid == 0) { // Это код, выполняемый дочерним процессом

printf("Дочерний процесс: Значение data = %d, Адрес = %p\n", \*data, data);

printf("Дочерний процесс: Значение new\_data = %d, Адрес = %p\n", \*new\_data, new\_data);

(\*data)++; // Меняем значение

(\*new\_data)++; // Меняем значение

printf("Дочерний процесс: Новое значение data = %d, Адрес = %p\n", \*data, data);

printf("Дочерний процесс: Новое значение new\_data = %d, А адрес = %p\n", \*new\_data, new\_data);

} else { // Это код, выполняемый родительским процессом

printf("Родительский процесс: Значение data = %d, Адрес = %p\n", \*data, data);

printf("Родительский процесс: Значение new\_data = %d, Адрес = %p\n", \*new\_data, new\_data);

sleep(1); // Даем дочернему процессу время для выполнения

printf("Родительский процесс: Значение data осталось неизменным: %d, Адрес = %p\n", \*data, data);

printf("Родительский процесс: Значение new\_data осталось неизменным: %d, Адрес = %p\n", \*new\_data, new\_data);

}

free(data); // Освобождаем выделенную память

delete new\_data;

return 0;

}