**Лабораторная работа №3**

**Работа с процессами в ОС Windows и Linux**

Задание:

**Windows**

Подготовка к работе

1. Изучить программу matrix.cpp. Ознакомиться с тем, как запускать эту программу, как она работает и что выводит в результате. Скрин и описание программы вставить в отчёт.
2. Скомпилировать программу для Windows в Visual Studio. Получить исполняемый файл matrix.exe.

Запуск, остановка, параметры процесса

1. Запустить программу matrix.exe через Power-Shell командой **./matrix.exe <n>**. Подобрать размерность матриц n так, чтобы процесс не завершался в течение хотя бы 5 минут. (У меня было значение 5000). Сделать скрин.
2. Найти процесс в диспетчере задач во вкладке Процессы. Сделать скрин.
3. Найти процесс в диспетчере задач во вкладке Подробности. Сделать отображение столбца «Базовый приоритет». Посмотреть, с каким приоритетом запущен процесс. Сделать скрин.
4. Открыть Power-Shell. Запустить команду **Get-Process -Name matrix**. (Процесс в это время должен всё ещё работать). Описать, что выведется в результате выполнения команды.
5. Запустить команду **Get-Process -Name matrix | Select-Object Name, PriorityClass**. Описать, что выведется в результате выполнения команды.
6. Остановить процесс командой **Stop-Process -Id <PID>**, где PID – ID запущенного процесса matrix.

Изучение разницы работы программы в зависимости от приоритетов

1. Запустить программу matrix.exe. Подобрать размерность матриц так, чтобы процесс отрабатывал за 20 секунд. (У меня было значение 1500). Сделать скрин результата выполнения программы (за сколько она отработала). Для точности повторить замеры 5 раз и зафиксировать всё в таблицу. Посчитать среднее значение.
2. Запустить программу matrix.exe с той же размерностью, но с повышенным приоритетом. Приоритет повышается двумя командами:  
   **$process = Get-Process -Name "matrix"  
   $process.PriorityClass = "High"**  
   Показать приоритет запущенной программы через Диспетчер задач или Power-Shell. Сделать скрин результата выполнения программы (за сколько она отработала). Для точности повторить замеры 5 раз и зафиксировать всё в таблицу.
3. Сделать выводы о результатах повышения приоритета у процесса. Как оно повлияло на работу программы? Стала ли быстрее работать программа? На сколько процентов?

**Linux**

Подготовка к работе

1. Скачиваем VS Code. Устанавливаем туда расширение для C/C++.
2. Открываем терминал, проверяем, установлен ли у нас компилятор g++ командой **g++ --version**. Если не установлен – устанавливаем командой **sudo apt install g++**. Команда попросит ввести пароль пользователя. После чего попросит подтверждения на установку – вводим Y и нажимаем Enter.
3. Компилируем программу через VS Code или при помощи команды **g++ -o matrix.out matrix.cpp**. Получаем исполняемый файл matrix.out. Выводим в консоли, что файл создался командой **ls -l**. Делаем скрин.

Запуск, остановка, параметры процесса

1. Запускаем программу командой **./matrix.out <n>**. Подобрать размерность матриц n так, чтобы процесс не завершался в течение хотя бы 5 минут. (У меня было значение 5000). Сделать скрин.
2. Открываем System Monitor и ищем matrix.out в процессах. Делаем скрин.
3. Открываем ещё один терминал. Вводим команду **top**. Ищем процесс matrix.out. Описать, что выведется в результате выполнения команды. Выяснить приоритет потока matrix.out. Выйти из top можно, нажав клавишу q.
4. Остановить процесс командой **kill <PID>**, где PID – ID запущенного процесса matrix.

Изучение разницы работы программы в зависимости от приоритетов

1. Запустить программу matrix.out. Подобрать размерность матриц так, чтобы процесс отрабатывал за 20 секунд. (У меня было значение 1500). Сделать скрин результата выполнения программы (за сколько она отработала). Для точности повторить замеры 5 раз и зафиксировать всё в таблицу. Посчитать среднее значение.
2. Запустить программу matrix.exe с той же размерностью, но с повышенным приоритетом. Приоритет повышается командой:  
   **sudo renice -n -20 -p $(ps -o pid,cmd -e | grep matrix.out | xargs | cut -d " " -f 1)**Показать приоритет запущенной программы через top. Сделать скрин результата выполнения программы (за сколько она отработала). Для точности повторить замеры 5 раз и зафиксировать всё в таблицу.
3. Сделать выводы о результатах повышения приоритета у процесса. Как оно повлияло на работу программы? Стала ли быстрее работать программа? На сколько процентов?

Код программы:

#include <iostream>

#include <sys/time.h>

#include <cstdlib>

#include <string.h>

using namespace std;

// Запускать программу так: matrix.exe <n>

// Где n - размерность матриц для перемножения

// Рекомендуется брать n где-то около 500-1000

int main(int argc, char \*\*argv) {

    // Говорим пользователю, что программа стартовала

    cout << "Started....";

    // Делаем отметку о начале выполнения программы

    struct timeval curr\_time{};

    gettimeofday(&curr\_time, nullptr);

    time\_t start\_time = (curr\_time.tv\_sec \* 1000) + (curr\_time.tv\_usec / 1000);

    // Получаем размерность матриц из консоли или используем дефолтную

    int n = 100;

    if (argc == 2) {

        n = atoi(argv[1]);

    }

    // Инициализируем модуль рандома с определённым начальным значением

    // Это позволит генерировать одинаковые массивы при каждом запуске

    srand(1);

    // Инициализируем первый и второй массивы со случайными числами

    int \*\*arr1 = new int\*[n];

    int \*\*arr2 = new int\*[n];

    long \*\*rez = new long\*[n];

    for (int i=0; i<n; i++) {

        arr1[i] = new int[n];

        arr2[i] = new int[n];

        rez[i] = new long[n];

    }

    for (int i=0; i<n; i++) {

        for (int j=0; j<n; j++) {

            arr1[i][j] = rand();

            arr2[i][j] = rand();

        }

    }

    // Перемножаем массивы

    for (int i=0; i<n; i++) {

        for (int j=0; j<n; j++) {

            rez[i][j] = 0;

            for (int k=0; k<n; k++) {

                rez[i][j] += arr1[i][k] \* arr2[k][j];

            }

        }

    }

    // Делаем отметку о завершении выполнения программы

    gettimeofday(&curr\_time, nullptr);

    time\_t end\_time = (curr\_time.tv\_sec \* 1000) + (curr\_time.tv\_usec / 1000);

    cout << "Execution time: " << (end\_time - start\_time) << " ms" << endl;

    // Освобождаем память

    for (int i=0; i<n; i++) {

        delete [] arr1[i];

        delete [] arr2[i];

        delete [] rez[i];

    }

    delete [] arr1;

    delete [] arr2;

    delete [] rez;

    return 0;

}

Требования к отчёту:

1. Название лабораторной: Основы работы с Linux.
2. Задание и вариант.
3. Скрины, необходимые пояснения и таблицы в соответствии с заданием.
4. Вывод о том, чему научились в лабораторной работе.