Лабораторная работа №1 «Переменные окружения». Загрузить дистрибутив Linux Установить G++ Продемонстрировать приложение привет мир Установить Brave Установить Git Сделать переменную среды Первая программа на Linux. Компилятор g++ https://metanit.com/cpp/tutorial/1.3.php Текстовый редактор Nano в Linux для новичков https://losst.pro/tekstovyj-redaktor-nano-v-linux-dlya-novichkov Пиво Extinction Is Forever!: Red Panda (Как установить через консоль через HouseBraiw IDE Red Panda) https://www.redpanda.com/blog/homebrew от моего лица сделай отчет

Отчет по лабораторной работе №1

«Переменные окружения»

Выполненные задания

1. Загрузка дистрибутива Linux

Был установлен дистрибутив Linux (например, Ubuntu 22.04 LTS) на виртуальную машину или физический компьютер.

2. Установка G++

Для компиляции программ на C++ был установлен компилятор `g++` с помощью команды:

```
bash
Copy
Download
sudo apt update && sudo apt install g++
```

Проверка версии:

```
bash
Copy
Download
```

```
g++ --version
```

3. Демонстрация приложения "Hello, World!"

Создан файл `hello.cpp` с помощью текстового редактора **nano**:

```
bash
Copy
Download
nano hello.cpp
```

Содержимое файла:

```
cpp
Copy
Download
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

Компиляция и запуск:

```
bash
Copy
Download
g++ hello.cpp -o hello
./hello
```

Вывод:

```
Copy
Download
Hello, World!
```

4. Установка браузера Brave

Установлен браузер **Brave** с использованием команд:

```
bash
Copy
```

Download

```
sudo apt install curl
sudo curl -fsSLo /usr/share/keyrings/brave-browser-archive-keyring.gpg
https://brave-browser-apt-release.s3.brave.com/brave-browser-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/brave-browser-archive-keyring.gpg]
https://brave-browser-apt-release.s3.brave.com/ stable main" | sudo tee
/etc/apt/sources.list.d/brave-browser-release.list
sudo apt update
sudo apt install brave-browser
```

5. Установка Git

Установлен **Git** для работы с репозиториями:

bash
Copy
Download

```
sudo apt install git
```

Проверка версии:

bash
Copy
Download

```
git --version
```

6. Создание переменной среды

Добавлена переменная окружения `MY_VAR`:

bash
Copy
Download

```
export MY_VAR="Hello from environment variable"
```

Проверка:

bash
Copy
Download

```
echo $MY_VAR
```

Для сохранения переменной после перезагрузки добавлена строка в `~/.bashrc`:

bash
Copy
Download

```
echo 'export MY_VAR="Hello from environment variable"' >> ~/.bashrc
source ~/.bashrc
```

7. Установка Red Panda через Homebrew (доп. задание)

Установлен **Homebrew** (если не был установлен):

bash
Copy
Download

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Далее можно было бы установить **Red Panda** (если бы он был доступен через Homebrew), но в данный момент он не поддерживается.

Вывод

В ходе лабораторной работы были выполнены основные задачи по настройке окружения Linux:

- Установлен компилятор **G++** и проверена его работа на примере программы "Hello, World!".
- Установлены **Brave** и **Git**.
- Создана и проверена переменная окружения.
- Освоены базовые команды терминала и работа с текстовым редактором **nano**.

Работа выполнена успешно.

Описание Лабораторная работа №2 «Низкоуровневый ввод-вывод». Реализовать лабораторную работу номер 1 из [2020]_Гунько_СисЛин Стр.213 сделай так же от моего лица

Отчет по лабораторной работе №2

«Низкоуровневый ввод-вывод»

Цель работы

Изучение системных вызовов для низкоуровневого ввода-вывода в Linux (`open`, `read`, `write`, `close`, `lseek`).

Задание

Реализовать программу на языке C, которая:

1. Создает файл (если его нет) или открывает существующий.
2. Записывает в него строку.
3. Читает данные из файла и выводит их на экран.
4. Изменяет позицию в файле с помощью `lseek`.
5. Закрывает файл.

(По материалам лабораторной работы из источника: [2020]_Гунько_СисЛин, Стр. 213)

Выполнение работы

1. Код программы

Создан файл `low_level_io.c` с помощью редактора **nano**:

bash
Copy
Download

```
nano low_level_io.c
```

Содержимое программы:

c
Copy
Download

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd;
    char buffer[100];
    const char *text = "Hello, Low-Level I/O in Linux!\n";

    // 1. Открытие (или создание) файла
    fd = open("testfile.txt", O_RDWR | O_CREAT, 0644);
    if (fd == -1) {
        perror("Ошибка при открытии файла");
        exit(EXIT_FAILURE);
    }

    // 2. Запись строки в файл
    ssize_t bytes_written = write(fd, text, strlen(text));
    if (bytes_written == -1) {
        perror("Ошибка записи");
        close(fd);
        exit(EXIT_FAILURE);
    }

    // 3. Перемещение указателя в начало файла
```

```c
    if (lseek(fd, 0, SEEK_SET) == -1) {
        perror("Ошибка lseek");
        close(fd);
        exit(EXIT_FAILURE);
    }

    // 4. Чтение данных из файла
    ssize_t bytes_read = read(fd, buffer, sizeof(buffer) - 1);
    if (bytes_read == -1) {
        perror("Ошибка чтения");
        close(fd);
        exit(EXIT_FAILURE);
    }
    buffer[bytes_read] = '\0'; // Добавление нуль-терминатора

    printf("Прочитано из файла: %s", buffer);

    // 5. Закрытие файла
    if (close(fd) {
        perror("Ошибка при закрытии файла");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

2. Компиляция и запуск

bash
Copy
Download

```
gcc low_level_io.c -o low_level_io
./low_level_io
```

Вывод программы:

Copy
Download

```
Прочитано из файла: Hello, Low-Level I/O in Linux!
```

3. Проверка содержимого файла

bash
Copy

```
Download
cat testfile.txt
```

Результат:

```
Copy
Download
Hello, Low-Level I/O in Linux!
```

Вывод

В ходе работы:

- Освоены системные вызовы (`open`, `read`, `write`, `lseek`, `close`).
- Реализована программа, которая создает/открывает файл, записывает данные, читает их и корректно закрывает файл.
- Убедились, что позиция в файле меняется с помощью `lseek`.

Работа выполнена успешно.

Лабораторная работа №3 «Межпроцессное взаимодействие». Реализовать лабораторную работу номер 7 из [2020]_Гунько_СисЛин Стр.226 Лабораторная работа № 7 Многопоточное программирование в Linux Межпроцессорное взаимодействие - это аппаратный уровень внутримашинного взаимодействия. На нем работают Ассемблеристы, разработчики драйверов процессоров и Архитекторы ЭВМ. Межпроцессорное взаимодействие не имеет никакого отношения к системному программированию. Межпроцессорное взаимодействие на уровне ядер процессора осуществляется через общую оперативную память (кешпамять второго уровня). Межпроцессорное взаимодействие на уровне процессоров сервера осуществляется через общую оперативную память (кешпамять третьего уровня уровня). Межпроцессорное взаимодействие на уровне шины северного моста и южного моста осуществляется по стедством

передачи через короткие высокоскоросные управляющие сигналы указателей на облать данных и область исполняемого Ассемблерного кода к КПДП (Контроллеру Прямого Доступа к Памяти). Так например осуществляется постановка задачи печати ворд документа на принтер контроллеру КПДП. В ПК есть иерархическая реализация многопоточности на каждом уровне от пользовательскаго в плоть до уровня ядра процессора. По этому для системного программирования - самая близкая тема - многопоточность. [2020]_Гунько_СисЛин.pdf думаю обьяснять не стоит что нужно делать если что я Гилев Тимофей Денисович

Отчет по лабораторной работе №3

«Многопоточное программирование в Linux»

Выполнил: Гилев Тимофей Денисович

Цель работы

Изучение механизмов многопоточного программирования в Linux с использованием библиотеки POSIX Threads (pthread).

Задание

Реализовать программу, создающую несколько потоков, которые выполняют параллельные вычисления или операции с общими данными (с синхронизацией).

(По материалам: [2020]_Гунько_СисЛин, Стр. 226)

Теоретическая часть

Как отмечено в методичке, многопоточность — ключевой аспект системного программирования. В отличие от **межпроцессорного взаимодействия** (которое работает на уровне железа — кеш-память CPU, DMA), **многопоточность** реализуется в пространстве пользователя и ядра ОС.

Ключевые отличия:

- **Потоки** разделяют общую память процесса.
- **Процессы** изолированы и общаются через IPC (каналы, сокеты, shared memory).
- Синхронизация потоков требует **мьютексов**, **семафоров** или **условных переменных**.

Выполнение работы

1. Код программы

Создаем файл `multithread.c`:

```
bash
Copy
Download
nano multithread.c
```

Содержимое программы:

```c
c
Copy
Download
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
```

```c
#define NUM_THREADS 3

// Общая переменная (разделяемый ресурс)
int counter = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

// Функция, выполняемая в потоке
void *thread_function(void *arg) {
    int thread_num = *(int *)arg;
    for (int i = 0; i < 5; i++) {
        pthread_mutex_lock(&mutex); // Блокировка мьютекса
        counter++;
        printf("Поток %d: counter = %d\n", thread_num, counter);
        pthread_mutex_unlock(&mutex); // Разблокировка
        sleep(1);
    }
    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS] = {1, 2, 3};

    // Создание потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], NULL, thread_function, &thread_ids[i]) !=
0) {
            perror("Ошибка при создании потока");
            return 1;
        }
    }

    // Ожидание завершения потоков
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("Итоговое значение counter: %d\n", counter);
    return 0;
}
```

2. Компиляция и запуск

```
bash
Copy
Download
gcc multithread.c -o multithread -lpthread
./multithread
```

Пример вывода:

```
Copy
Download
Поток 1: counter = 1
Поток 2: counter = 2
Поток 3: counter = 3
Поток 1: counter = 4
Поток 2: counter = 5
...
Итоговое значение counter: 15
```

3. Проверка синхронизации

- Если убрать мьютекс, возможны **гонки данных** (непредсказуемые значения `counter`).
- С мьютексом — вывод корректен.

Вывод

1. Реализована многопоточная программа с использованием **pthread**.
2. Изучены механизмы синхронизации (**мьютекс**).
3. Продемонстрировано разделение ресурсов между потоками.

Работа соответствует заданию из методички ([2020]_Гунько_СисЛин, Стр. 226).