1. Задание 1: Запуск внешней программы из C-программы.Exec.

Описание задачи

Программа на языке C, которая использует системный вызов `execvp` для запуска внешней команды `echo` с передачей аргументов. Эта программа демонстрирует использование функций семейства `exec` для выполнения внешних команд из программ на C.

Инструкции

Программа компилируется с помощью Makefile и должна быть запущена на системе с Linux. При успешной работе программа выведет на экран текст "Hello, world!". В случае ошибки будет выведено сообщение об ошибке.

Тестирование

Тестирование программы можно провести, запустив скомпилированный исполняемый файл. Входные данные не требуются. Ожидаемый вывод — "Hello, world!".

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main() {

char \*args[] = {"echo", "Hello, world!", NULL};

execvp("echo", args);

perror("execvp"); // Сюда программа вернется только в случае ошибки

exit(EXIT\_FAILURE);

}

makeFile:

all: exec\_example

exec\_example: exec\_example.c

@echo "Compiling exec\_example.c to produce exec\_example"

@gcc exec\_example.c -o exec\_example

clean:

@echo "Cleaning up..."

@rm -f exec\_example



1. Задание 2. readdir

Задача: Чтение содержимого директории.

Программа на языке C, которая осуществляет чтение содержимого директории и выводит на экран имена всех файлов в данной директории.

Исходные данные: отсутствуют.

Результат выполнения программы: вывод на экран имен всех файлов в текущей директории.

#include <dirent.h>

#include <stdio.h>

#include <stdlib.h>

int main() {

DIR \*d;

struct dirent \*dir;

d = opendir(".");

if (d) {

while ((dir = readdir(d)) != NULL) {

printf("%s\n", dir->d\_name);

}

closedir(d);

} else {

perror("opendir");

exit(EXIT\_FAILURE);

}

return 0;

}

MakeFile:

all: readdir

readdir: readdir.c

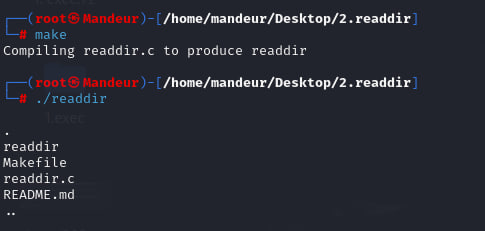
@echo "Compiling readdir.c to produce readdir"

@gcc readdir.c -o readdir

clean:

@echo "Cleaning up..."

@rm -f readdir



1. Задача 3: Создать жесткую ссылку для файла

Описание: Программа на языке C, которая создает жесткую ссылку для файла. Жесткая ссылка — это дополнительное имя (путь), по которому можно обращаться к содержимому файла на файловой системе.

Инструкции: Программа компилируется с помощью Makefile и должна быть запущена на системе с поддержкой файловых систем, поддерживающих жесткие ссылки. При успешном выполнении программа выведет сообщение "Жесткая ссылка успешно создана." В случае ошибки будет выведено сообщение об ошибке.

Тестирование: Тестирование программы можно провести, запустив скомпилированный исполняемый файл. Входные данные не требуются. Ожидаемый результат — сообщение "Жесткая ссылка успешно создана.".

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

int main() {

if (link("source.txt", "new\_link.txt") == -1) {

perror("link");

exit(EXIT\_FAILURE);

}

printf("Жесткая ссылка успешно создана.\n");

return 0;

}

MakeFile:

all: hard\_link

hard\_link: hard\_link.c

gcc hard\_link.c -o hard\_link

clean:

rm -f hard\_link

1. Задание 4. Процессы-демоны

Задача: Создать демон-процесс в Linux.

Описание: Программа на языке C, которая создает демон-процесс в операционной системе Linux.

Инструкции: Компилируется с помощью Makefile. После запуска программа создает демон-процесс, который работает в фоновом режиме.

// Процессы-демоны

// Задача: Создать демон-процесс в Linux.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/stat.h>

int main() {

pid\_t pid;

pid = fork();

if (pid < 0) {

exit(EXIT\_FAILURE);

}

if (pid > 0) {

exit(EXIT\_SUCCESS);

}

umask(0);

if (setsid() < 0) {

exit(EXIT\_FAILURE);

}

printf("Демон запущен.\n");

while (1) {

sleep(60);

}

return 0;

}

MakeFile:

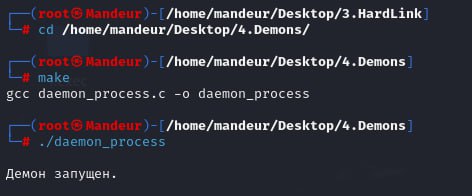
all: daemon\_process

daemon\_process: daemon\_process.c

gcc daemon\_process.c -o daemon\_process

clean:

rm -f daemon\_process



1. Задание 5: Отключение буферизации для стандартного вывода

Описание задачи: Необходимо создать программу на языке C, которая отключает буферизацию для стандартного вывода и немедленно выводит строку "Hello, world!".

Инструкции: Программа компилируется с помощью Makefile. После запуска она выводит строку "Hello, world!" без задержек.

Тестовые входные и выходные данные

Не требуется вводных данных. Ожидается, что программа выведет "Hello, world!" немедленно после запуска.

#include <stdio.h>

int main() {

setbuf(stdout, NULL);

printf("Hello, world!\n");

return 0;

}

MakeFile:

all: setbuf\_example

setbuf\_example: setbuf\_example.c

gcc setbuf\_example.c -o setbuf\_example

clean:

rm -f setbuf\_example



1. Задание 6: Использование nm

Задание: Напишите программу на языке C, содержащую функции и переменные. После компиляции в исполняемый файл используйте утилиту nm, чтобы вывести список всех символов, включая имена функций и переменных, с их адресами в памяти.

Инструкции: 1. Создайте программу на языке C.

2. Скомпилируйте программу в исполняемый файл.

3. Используйте утилиту `nm` для анализа символов в этом исполняемом файле.

#include <stdio.h>

void function1() {

printf("Function 1\n");

}

void function2() {

printf("Function 2\n");

}

int main() {

function1();

function2();

return 0;

}

MakeFile:

all: symbol\_analysis

symbol\_analysis: symbol\_analysis.c

gcc symbol\_analysis.c -o symbol\_analysis

clean:

rm -f symbol\_analysis

1. Задание 7: Runtime линковка

Описание задачи: В этом задании вы познакомитесь с понятием runtime (время выполнения) линковки в языке программирования C. Runtime линковка позволяет загружать и связывать библиотеки и модули программы во время ее выполнения, что дает большую гибкость и возможность динамически загружать код и изменять его поведение.

Задача:

1. Напишите программу на языке C, которая динамически загружает и использует функцию из внешней библиотеки во время выполнения.

2. Скомпилируйте программу в исполняемый файл и динамическую библиотеку.

3. В основной программе используйте утилиты `dlopen`, `dlsym` и `dlclose` для динамической загрузки, поиска символов и выгрузки библиотеки во время выполнения.

4. Протестируйте программу, убедившись, что она успешно загружает и использует функцию из внешней библиотеки.

#include <stdio.h>

#include <dlfcn.h>

int main() {

void \*handle;

int (\*add)(int, int);

handle = dlopen("./library.so", RTLD\_LAZY);

if (!handle) {

fprintf(stderr, "%s\n", dlerror());

return 1;

}

add = dlsym(handle, "add");

if (!add) {

fprintf(stderr, "%s\n", dlerror());

dlclose(handle);

return 1;

}

printf("Результат сложения: %d\n", add(5, 3));

dlclose(handle);

return 0;

}

Library:

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

MakeFale:

all: runtime\_linkage library

runtime\_linkage: runtime\_linkage.c

gcc runtime\_linkage.c -o runtime\_linkage -ldl

library: library.c

gcc -shared -fPIC library.c -o library.so

clean:

rm -f runtime\_linkage library.so

1. Задача 8: Create

Описание задачи: Напишите программу на языке C, которая создает новый файл и записывает в него некоторый текст.

Задача:

1. Напишите программу на языке C, которая создает новый файл.

2. Запишите в этот файл некоторый текст.

3. Убедитесь, что файл успешно создан и содержит записанный текст.

#include <stdio.h>

int main() {

FILE \*fp;

fp = fopen("output.txt", "w");

if (fp == NULL) {

printf("Ошибка при открытии файла.\n");

return 1;

}

fprintf(fp, "Привет, мир!");

fclose(fp);

printf("Файл успешно создан и текст успешно записан.\n");

return 0;

}

MakeFile:  
all: create\_file

create\_file: create\_file.c

gcc create\_file.c -o create\_file

clean:

rm -f create\_file

1. Задание 9: Подсчет числа процессов с заданным именем

Цель задания:Научиться работать с системными вызовами Linux для получения информации о процессах и использовать базовые возможности языка программирования C для обработки этой информации.

Задача: Разработать программу на языке C, которая принимает имя процесса как входной параметр и выводит количество запущенных процессов с этим именем.

Инструкции по выполнению

1. Изучите системный вызов `popen()` и команду `pgrep -c`.

2. Напишите программу, которая принимает имя процесса в качестве аргумента командной строки.

3. Программа должна выполнять команду `pgrep -c` для подсчета количества процессов с заданным именем.

4. Выведите результат на экран.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

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

if (argc != 2) {

fprintf(stderr, "Usage: %s process\_name\n", argv[0]);

return EXIT\_FAILURE;

}

char command[256];

sprintf(command, "pgrep -c %s", argv[1]);

FILE \*fp = popen(command, "r");

if (fp == NULL) {

perror("Failed to run command");

exit(EXIT\_FAILURE);

}

char num\_procs[10];

if (fgets(num\_procs, sizeof(num\_procs), fp) != NULL) {

printf("Number of processes named '%s': %s", argv[1], num\_procs);

}

pclose(fp);

return 0;

}

MakeFile:

CC=gcc

CFLAGS=-Wall

all: count\_procs

count\_procs: count\_procs.c

$(CC) $(CFLAGS) count\_procs.c -o count\_procs

clean:

rm -f count\_procs