**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Информационные техеологии |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | **Информационные системы и программирование** |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | Очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по лабораторной работе №4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | «Переменные окружения». | | |
|  |  | (наименование темы) | | |
|  |  | Меж процессное взаимодействии | | |
| **по дисциплине** | | |  | **Системное программирование** |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Обучающийся** |  | Старчик Артемий Филппович |  |  |  |
|  |  | (ФИО) |  |  | (подпись) |
| **Группа** |  | ДКИП-206Прог |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Сибирев Иван Валерьевич |  |  |
|  |  | (ФИО) |  | (подпись) |

Лабораторная работа № 4

Каналы передачи данных

1. Цель работы: получить навыки организации обмена информацией между процессами средствами неименованных или именованных каналов в программах на языке C в операционных системах семейства Linux.

2. Краткие теоретические сведения: см. параграф 6.2, стр. 87.

Примеры кода: файлы pipe\_parent.c и pipe\_child.c для программных

каналов, либо server\_main.c и client\_main.c для каналов FIFO в разделе

«Примеры программ из пособия», подраздел «Pipes» сайта по дисциплине [6].

3. Методические указания.

3.1. Проект может быть реализован в виде консольного приложения в среде ОС Ubuntu, Fedora, CentOS или других средствами компилятора gcc версии не ниже 4.

3.2. Проект является модификацией лабораторной работы № 3,

в которой обмен данными между родительской и дочерними программами производится через программные каналы.

3.3. Проект может быть реализован в виде многозадачного приложения при использовании программных каналов или клиент-серверного приложения при использовании каналов FIFO. Второй способ

соответствует заданию повышенной сложности.

3.4. Проект должен предусматривать обработку исключительных

ситуаций (отсутствие или неверное количество входных параметров,

ошибки открытия входного и/или выходного файла, ошибки чтения и

записи).

3.5. Проект должен предусматривать обмен данными между родительской и дочерними программами или между клиентом и сервером

только через каналы (программные или FIFO соответственно).

4. Порядок выполнения работы:

4.1. В соответствии с личными предпочтениями выбрать средство

реализации межзадачных коммуникаций.

4.2. Модифицировать и отладить в соответствии с выбранным

средством коммуникации программу из лабораторной работы № 3, реализующую порожденный (дочерний) процесс. При необходимости

модифицировать ее в отдельную программу-сервер. 221

4.3. Модифицировать и отладить в соответствии с выбранным

средством коммуникации программу из лабораторной работы № 3, реализующую родительский процесс. При необходимости модифицировать ее в отдельную программу-клиент.

4.4. Результатом работы программы является совокупность выходных текстовых файлов, соответствующих совокупности (2 и более)

входных файлов, содержащих текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций

или –1 в случае ошибки.

4.5. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.

4.6. Результатом выполнения лабораторной работы считается демонстрация работы программы и обработки исключительных ситуаций преподавателю.

5. Варианты заданий: см. лабораторную работу № 1 (стр. 214).

6. Содержание отчета.

6.1. Цель работы.

6.2. Вариант задания.

6.3. Листинг программы.

7. Контрольные вопросы: см. вопросы для самопроверки к разделу 6.2 (стр. 97)

**6.1. Цель работы**

Получение навыков организации обмена информацией между процессами с помощью программных каналов (pipe) в программах на языке C в операционных системах Linux. Реализация многозадачного приложения для подсчёта слов в тексте

**6.2. Вариант задания**

Вариант задания: Подсчет слов в файлах.  
Цель: создать программу, которая:

* Обрабатывает несколько входных файлов параллельно в отдельных процессах.
* Использует программные каналы для передачи результатов между процессами.
* Формирует выходные файлы с количеством слов в каждом из входных файлов.

**6.3. Листинг программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#define MAX\_THREADS 10

#define MAX\_FILENAME\_LEN 256

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

if (argc < 2) {

fprintf(stderr, "Usage: %s <input\_file1> <input\_file2> ... <input\_fileN>\n", argv[0]);

return -1;

}

int num\_files = argc - 1;

if (num\_files > MAX\_THREADS) {

fprintf(stderr, "Too many files. Maximum allowed: %d\n", MAX\_THREADS);

return -1;

}

int pipes[MAX\_THREADS][2]; // Массив каналов для каждого файла

pid\_t pids[MAX\_THREADS]; // Массив PID дочерних процессов

// Создание каналов и процессов

for (int i = 0; i < num\_files; i++) {

// Создаем канал

if (pipe(pipes[i]) == -1) {

perror("Pipe creation failed");

return -1;

}

// Создаем дочерний процесс

pids[i] = fork();

if (pids[i] == -1) {

perror("Fork failed");

close(pipes[i][0]);

close(pipes[i][1]);

return -1;

}

if (pids[i] == 0) { // Дочерний процесс

// Закрываем ненужный end канала

close(pipes[i][0]);

// Обработка файла

char \*input\_file = argv[i + 1];

FILE \*input = fopen(input\_file, "r");

if (!input) {

perror("Error opening input file");

close(pipes[i][1]);

exit(EXIT\_FAILURE);

}

int count = 0;

int in\_word = 0;

int c;

while ((c = fgetc(input)) != EOF) {

if (c == ' ' || c == '\t' || c == '\n') {

in\_word = 0;

} else {

if (!in\_word) {

count++;

in\_word = 1;

}

}

}

fclose(input);

// Передаем результат в канал

write(pipes[i][1], &count, sizeof(count));

close(pipes[i][1]);

exit(EXIT\_SUCCESS);

} else { // Родительский процесс

close(pipes[i][1]); // Закрываем write end

}

}

// Родительский процесс ждет завершения всех дочерних

for (int i = 0; i < num\_files; i++) {

wait(NULL);

}

// Собираем результаты из каналов и записываем в файлы

for (int i = 0; i < num\_files; i++) {

int count;

if (read(pipes[i][0], &count, sizeof(count)) != sizeof(count)) {

perror("Error reading from pipe");

close(pipes[i][0]);

continue;

}

char output\_file[MAX\_FILENAME\_LEN];

snprintf(output\_file, MAX\_FILENAME\_LEN, "output\_%d.txt", i + 1);

FILE \*output = fopen(output\_file, "w");

if (output) {

fprintf(output, "Word count: %d\n", count);

fclose(output);

} else {

perror("Error opening output file");

}

close(pipes[i][0]); // Закрываем read end

}

return 0;

}

,

**1. В чем состоит необходимость организации библиотек объектов?**

Библиотеки объектов упрощают повторное использование кода, уменьшают размер программы (из-за общего доступа к функциям), позволяют обновлять компоненты независимо друг от друга и ускоряют процесс компиляции.

**2. Дайте определение библиотеки объектных файлов.**

Библиотека объектных файлов — это набор объектных файлов (**.o**), объединённых в один файл (например, **.a**), который используется компилятором для связывания с программой.

**3. Дайте определение статической и динамической библиотеки.**

* Статическая библиотека (например, **.a**):  
  код библиотеки встраивается в программу при компиляции, что делает исполняемый файл автономным, но увеличивает его размер.
* Динамическая библиотека (например, **.so**):  
  код загружается в память при запуске программы, что экономит место и позволяет обновлять библиотеку без перекомпиляции программы.

**4. Опишите команду создания и редактирования статических библиотек.**

* Создание/редактирование :  
  Используется утилита **ar** (архиватор):

ar rcs libmylib.a file1.o file2.o # Создать/обновить статическую библиотеку

* Сортировка объектных файлов:

ranlib libmylib.a # Обновить индекс библиотеки (необязательно)

**5. Перечислите варианты подключения статических библиотек.**

1. Через флаг **-l** при компиляции:

1

gcc main.c -L/path/to/lib -lmylib # Подключить libmylib.a

1. Указание пути и имени библиотеки напрямую:

gcc main.c /path/to/libmylib.a

**6. Каковы особенности разделяемых библиотек?**

* Адресно-независимый код (PIC/Position-Independent Code).
* Общий для процессов: Одна копия библиотеки в памяти используется многими процессами.
* Динамическая линковка: Загружается при запуске программы.
* Версионность: Можно иметь несколько версий библиотеки.

**7. Опишите процесс создания разделяемых библиотек.**

1. Компилируйте**-fPIC** (д

gcc -c -fPIC file1.c -o file1.o

1. Соберите их в динамическую библиотеку:

gcc -shared -o libmylib.so file1.o file2.o

**8. Что такое адресно-независимый код (PIC)?**

Адрес-независимый код — это код, который может быть загружен в любую область памяти и корректно работать, поскольку все переходы и ссылки реализованы через таблицы перенаправления (например, GOT и PLT в ELF).

**9. Каковы особенности применения разделяемых библиотек?**

* Динамическая загрузка: Библиотека загружается при запуске программы.
* Совместное использование: Одна копия библиотеки используется многими процессами.
* Обновление без перекомпиляции: изменения в библиотеке не требуют перекомпиляции программы.
* Зависимости: Программа зависит от версии библиотеки и её наличия в системе.

**10. Опишите команды objdump, readelf, nm.**

* **objdump** :  
  Просмотр секций, дизассемблированного кода, символов и зависимостей.

objdump -d ./program # Просмотр дизассемблерного кода

* **readelf** :  
  Информация о формате ELF (секции, символы, зависимости).

readelf -a ./libmylib.so

* **nm** :  
  Список символов (функций и

nm libmylib.a

**11. Что такое динамически загружаемые библиотеки?**

Динамически загружаемые библиотеки (например, **.so** в Linux) — это библиотеки, которые могут быть загружены в память и использованы программой во время её выполнения (например, через **dlopen()**).

**12. Перечислите функции интерфейса dlopen.**

* **dlopen()**: Загружает библиотеку в память.
* **dlsym()**: Ищет символ (функцию/переменную) в загруженной библиотеке.
* **dlclose()**: Освобождает библиотеку из памяти.
* **dlerror()**: Возвращает описание последней ошибки.

**13. Каковы особенности применения функции dlopen()?**

* Загрузка библиотеки в runtime.
* Возможность задать режим загрузки (например, `RTLD\_L**RTLD\_LAZY** или **RTLD\_NOW**).
* Необходимость обработки ошибок через **dlerror()**.

**\*\*14. Опишите флаги, применяемые14. Опишите флаги, применяемые в функции dlopen().**

* **RTLD\_LAZY**: Откладывает связывание символов до их первого использования.
* **RTLD\_NOW**: Выполняет связывание всех символов сразу после загрузки.
* **RTLD\_GLOBAL**: Символы библиотеки доступны для других библиотек и модулей.
* **RTLD\_LOCAL** (по умолчанию): Символы видны только текущей программе и библиотеке.

**15. Каковы особенности применения функции dlsym()?**

* Ищет символ по имени (например, **void\* func = dlsym(handle, "my\_function");**).
* Возвращает **NULL** при ошибке.
* Требует корректного имени символа и загруженной библиотеки.
* Используется для динамического вызова функций.