Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
|  |
| Кафедра «Информатика» |
| кафедра |

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ**

|  |
| --- |
| Лабораторная работа №4. Управление файлами в ОС GNU/Linux |
| Тема |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  |  |  | А. С. Кузнецов |
|  | |  |  |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ19-17/1б 031939175 | | |  |  |  | А. Д. Непомнящий |
|  | номер группы, зачетной книжки | | |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2021

1. Цель работы

Цель состоит в изучении принципов управления файлами в ОС GNU/Linux.

1. Задачи

Выполнение работы сводится к следующим задачам.

1. Ознакомление с теоретическим материалом по управлению областями виртуальной памяти в ОС GNU/Linux.
2. Разработка программы в соответствии с полученным заданием, в которой должен использоваться механизм управления кучами либо стеком.
3. Написание настоящего отчета защита его с исходными текстами и исполняемым модулем программы. Исходные тексты программ должны содержать комментарии в стиле системы doxygen.

Требуется разработать программу, позволяющую считывать, модифицировать существующие и добавлять новые структуры (записи) фиксированной длины из/во входные и выходные файлы. При реализации должны использоваться только средства низкоуровневого ввода-вывода. Все операции выполняются только над содержимым файловых объектов, а не над содержимым информационных структур, хранящихся во внутренней памяти. Использование высокоуровневых средств является ошибочным. Аналогично, выполнение операций во внутренней памяти с сохранением и загрузкой результатов в файлы является ошибочным.

Вариант 17. Структура данных: архипелаг; количество островов; количество обитаемых островов. Создать два запроса, позволяющих определить, имеются ли архипелаги, состоящие только из необитаемых островов, и получить список архипелагов с указанием количества островов в них.

1. Описание использованных при выполнении задания функций Linux API управления областями виртуальной памяти
   1. int open (const char \* file, int oflag, …);

В качестве аргументов принимаются строка с именем файла и флаги, определяющие способ открытия. С помощью данной функции можно создавать новый файл; для этого предназначен третий аргумент функции – права доступа к файлу. Когда задан флаг O\_CREAT, должен присутствовать третий аргумент, определяющий права доступа к создаваемому файлу.

* 1. int close (int fd);

По окончании работы с файлом его необходимо закрыть вызовом функции close. После того как файл закрыт, очевидно, что обращаться к нему нельзя. Закрытие файла вызывает разную реакцию ОС, в зависимости от типа файла. Linux ограничивает количество файлов, которые могут быть открыты процессом в определенный момент времени. Дескрипторы занимают ресурсы ядра ОС; поэтому желательно файлы закрывать вовремя, чтобы они удалялись из системных.

* 1. ssize\_t read (int fd, void\* Buf, size\_t BytesToRead);

Чтение данных из файла осуществляется функцией read. Очевидно, что файл должен быть открыт к моменту выполнения данной операции. Природа данных, которые считываются из файла посредством функции read, ей неинтересна. Она работает с байтовыми последовательностями.

* 1. off\_t lseek (int fd, off\_t bytesToMove, int whence);

Для выполнения еще одной полезной файловой операции, а именно перемещения внутри файла, принято использовать функцию lseek. Функция возвращает новую позицию файлового указателя (или (–1)), в случае ошибки. Если аргумент whence равен SEEK\_SET, то второй аргумент интерпретируется как смещение от начала файла. Если whence равен SEEK\_CUR, то смещение производится от текущей позиции в файле. Если whence равен SEEK\_END, то аргумент bytesToMove интерпретируется как смещение от конца файла. Функция неприменима к некоторым типам файлов, например сокетам.

* 1. int fstat (int fd, struct stat\* stat\_info);

В случае успеха эта функция возвращает 0 и заполняет структуру с данными о файле, в противном случае возвращается (–1).

* 1. int rename (char\* old\_path, char\* new\_path);

За переименование файлов ответственность несет функция rename.

* 1. int remove (char\* path);

За удаление файлов отвечает несколько функций, одна из них – remove.

1. Примеры использования этих функций в представленном программном коде

Листинг 1 – Код функции открытия файла для последующей работы с ним

/\*! \brief Prepares new file for saving island groups info

\*

\* \details Creates file with name filename, opens it putting descriptor

\* into \*fd by pointer, puts meta into this file.

\*

\* \param fd Pointer to file descriptor

\* \param filename Mame of save file

\* \param inputSize Maximal size of island groups' name

\*

\* \return 0 on success, -1 otherwise.

\*/

int PrepareNewFile(int\* fd, char\* filename, int inputSize)

{

mode\_t mode = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH;

int fdReturn;

fdReturn = open(filename, O\_RDWR | O\_CREAT | O\_TRUNC, mode);

if (fdReturn < 0)

{

return -1;

}

\*fd = fdReturn;

WriteMeta(\*fd, inputSize);

return 0;

}

Листинг 2 – Код функции, получающей размер файла

/\*! \brief Returns size of file

\*

\* \param fd File descriptor

\*

\* \return Size of file

\*/

int GetFileSize(int fd)

{

struct stat info;

int fstatSuccess = fstat(fd, &info);

if (fstatSuccess)

{

perror("Getting file info went wrong.");

}

return (int) info.st\_size;

}

Листинг 3 – Код функции, находящей в файле архипелаг с необходимым именем

/\*! \brief Moves the pointer in the file to group by name

\*

\* \param fd File descriptor

\* \param groupName Name of the group to seek

\* \param inputSize Maximal size of island groups' name

\*

\* \return 0 if group was found and seek set, -1 otherwise

\*/

int SeekToGroupByName(int fd, char\* groupName, int inputSize)

{

lseek(fd, sizeof(int32\_t), SEEK\_SET);

char currentGroupName[MAX\_INPUT\_SIZE];

strcpy(currentGroupName, groupName);

while (lseek(fd, 0, SEEK\_CUR) < GetFileSize(fd))

{

ReadInfo(fd, &currentGroupName, sizeof(char) \*

inputSize);

if (strcmp(groupName, currentGroupName) == 0)

{

lseek(fd, (int) -sizeof(char) \* inputSize, SEEK\_CUR);

return 0;

}

lseek(fd, sizeof(int32\_t) \* 2, SEEK\_CUR);

}

return -1;

}

Листинг 4 – Фрагмент кода с переименованием файла

int renameReturn = rename(TEMP\_FILE\_NAME, filename);

if (renameReturn != 0)

{

perror("Rename error");

Окончание листинга 4

}

Листинг 5 – Код функции, удаляющей файл

int DeleteFile(char\* filename)

{

int removeReturn = remove(filename);

if (removeReturn != 0)

{

perror("File delete error");

return -1;

}

return 0;

}

1. Содержимое файла Makefile

На следующем листинге приведено содержимое файла Makefile.

Листинг 6 – Код в файле Makefile

CC = gcc

CFLAGS = -std=gnu99

SOURCES = main.c Input.c FileIO.c

all:

$(CC) $(CFLAGS) $(SOURCES) -o start.o

1. Тестовые примеры работы программ

Далее на рисунках приведены тестовые примеры работы программы.

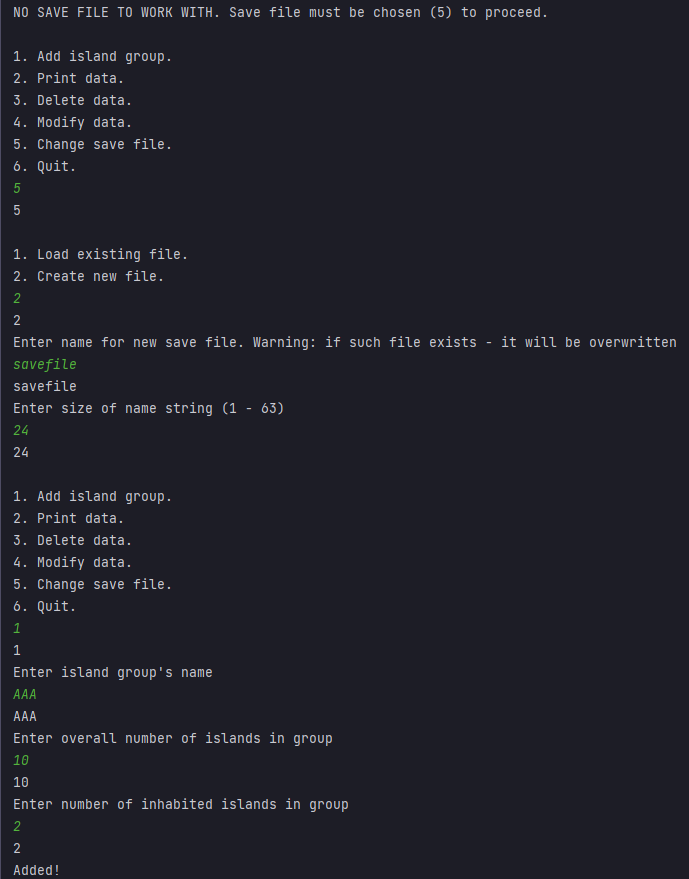


Рисунок 1 – Создание файла и добавление архипелага

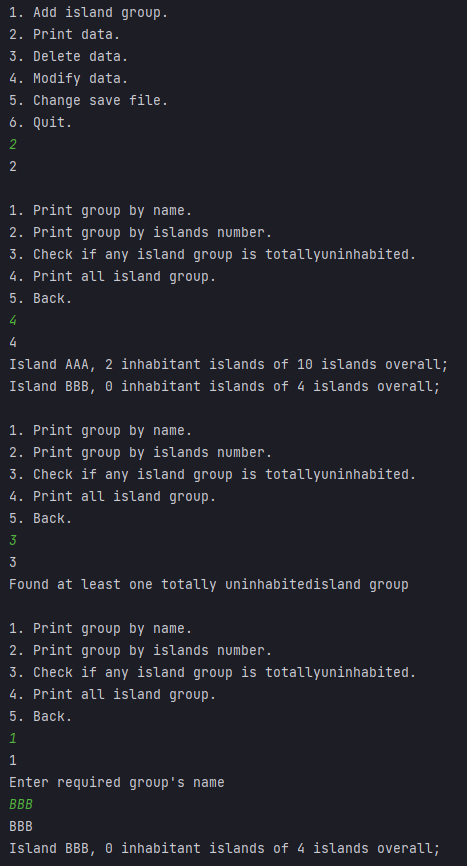


Рисунок 2 – Вывод информации

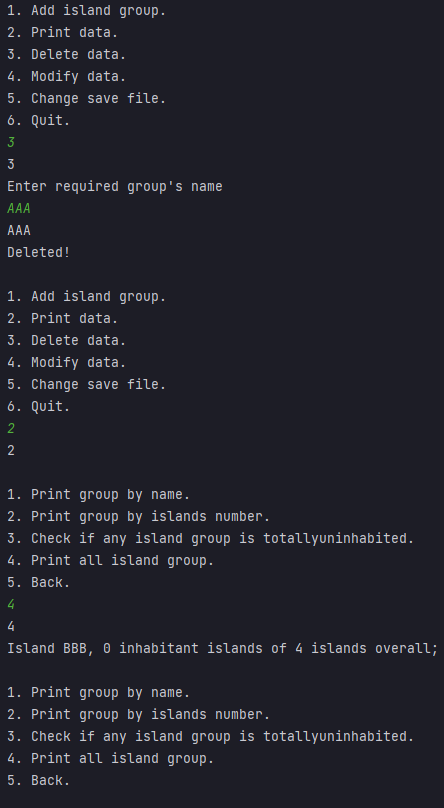


Рисунок 3 – Удаление архипелага

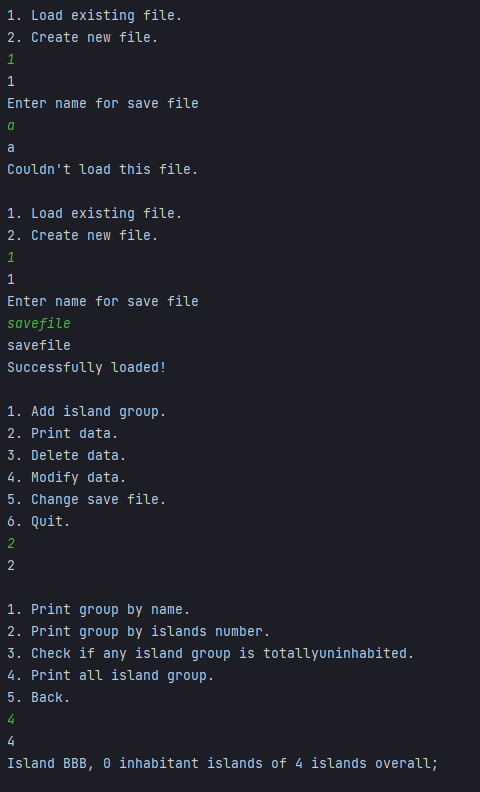


Рисунок 4 – Загрузка из файла с попыткой загрузить несуществующий файл

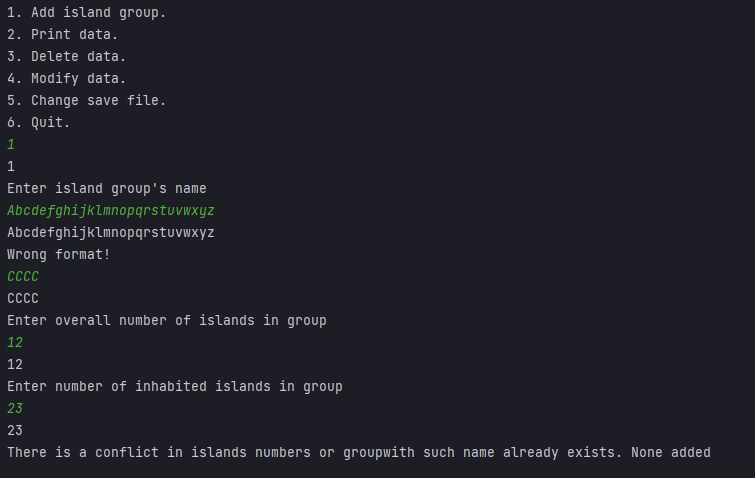


Рисунок 5 – Попытки неверного ввода

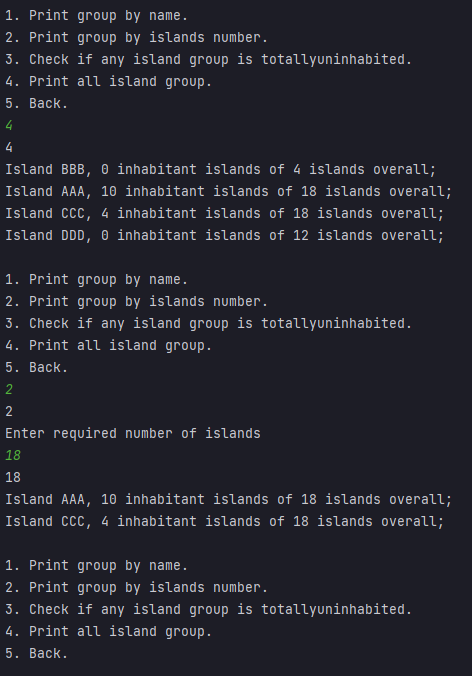


Рисунок 6 – Запрос всех архипелагов по заданному количеству островов