Отчёт по лабораторной работе №10

Петлин Артём Дмитриевич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Приобретение навыков написания программ для работы с файлами.

2 Задание

1. Напишите программу работающую по следующему алгоритму:
    - Вывод приглашения “Как Вас зовут?”
    - ввести с клавиатуры свои фамилию и имя
    - создать файл с именем name.txt
    - записать в файл сообщение “Меня зовут”
    - дописать в файл строку введенную с клавиатуры
    - закрыть файл

Создать исполняемый файл и проверить его работу. Проверить наличие файла и его содержимое с помощью команд ls и cat.

3 Теоретическое введение

3.1 Права доступа к файлам

ОС GNU/Linux является многопользовательской операционной системой. И для обеспечения защиты данных одного пользователя от действий других пользователей существуют специальные механизмы разграничения доступа к файлам. Кроме ограничения доступа, данный механизм позволяет разрешить другим пользователям доступ данным для совместной работы.
Права доступа определяют набор действий (чтение, запись, выполнение), разрешённых для выполнения пользователям системы над файлами. Для каждого файла пользователь может входить в одну из трех групп: владелец, член группы владельца, все остальные. Для каждой из этих групп может быть установлен свой набор прав доступа. Владельцем файла является его создатель. Для предоставления прав доступа другому пользователю или другой группе командой

```
chown [ключи] <новый_пользователь>[:новая_группа] <файл>
```

или

```
chgrp [ключи] < новая_группа > <файл>
```

Набор прав доступа задается тройками битов и состоит из прав на чтение, запись и исполнение файла. В символьном представлении он имеет вид строк

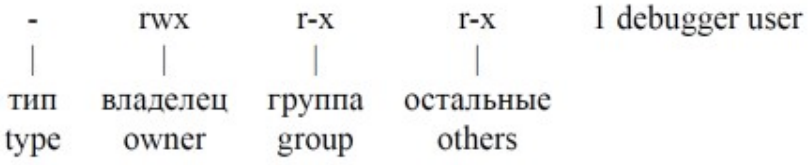
rwx, где вместо любого символа может стоять дефис. Всего возможно 8 комбинаций, приведенных в таблице 10.1. Буква означает наличие права (установлен в единицу второй бит триады r — чтение, первый бит w — запись, нулевой бит x — исполнение), а дефис означает отсутствие права (нулевое значение соответствующего бита). Также права доступа могут быть представлены как восьмеричное число. Так, права доступа rw- (чтение и запись, без исполнения) понимаются как три двоичные цифры 110 или как восьмеричная цифра 6.
Полная строка прав доступа в символьном представлении имеет вид:

```
<права_владельца> <права_группы> <права_остальных>
```

Так, например, права rwx r-x –x выглядят как двоичное число 111 101 001, или восьмеричное 751.
Свойства (атрибуты) файлов и каталогов можно вывести на терминал с помощью команды ls с ключом -l. Так например, чтобы узнать права доступа к файлу README можно узнать с помощью следующей команды:

```
$ls -l /home/debugger/README
-rwxr-xr-- 1 debugger users 0 Feb 14 19:08 /home/debugger/README
```

В первой колонке показаны текущие права доступа, далее указан владелец файла и группа:



Тип файла определяется первой позицией, это может быть: каталог — d, обычный файл — дефис (-) или символьная ссылка на другой файл — l. Следующие 3 набора по 3 символа определяют конкретные права для конкретных групп: r — разрешено чтение файла, w — разрешена запись в файл; x — разрешено исполнение файл и дефис (-) — право не дано.
Для изменения прав доступа служит команда chmod, которая понимает как

символьное, так и числовое указание прав. Для того чтобы назначить файлу /home/debugger/README права rw-r, то есть разрешить владельцу чтение и запись, группе только чтение, остальным пользователям — ничего:

```
$chmod 640 README # 110 100 000 == 640 == rw-r-----
$ls -l README
-rw-r 1 debugger users 0 Feb 14 19:08 /home/debugger/README
```

В символьном представлении есть возможность явно указывать какой группе какие права необходимо добавить, отнять или присвоить. Например, чтобы добавить право на исполнение файла README группе и всем остальным:

```
$chmod go+x README
$ls -l README
-rw-r-x--x 1 debugger users 0 Feb 14 19:08 /home/debugger/README
```

Формат символьного режима:

```
chmod <категория><действие><набор_прав><файл>
```

3.2 Работа с файлами средствами Nasm

В операционной системе Linux существуют различные методы управления файлами, например, такие как создание и открытие файла, только для чтения или для чтения и записи, добавления в существующий файл, закрытия и удаления файла, предоставление прав доступа.
Обработка файлов в операционной системе Linux осуществляется за счет использования определенных системных вызовов. Для корректной работы и доступа к файлу при его открытии или создании, файлу присваивается уникальный номер (16-битное целое число) – дескриптор файла.
Общий алгоритм работы с системными вызовами в Nasm можно представить в следующем виде:

1. Поместить номер системного вызова в регистр EAX;
2. Поместить аргументы системного вызова в регистрах EBX, ECX и EDX;
3. Вызов прерывания (int 80h);
4. Результат обычно возвращается в регистр EAX.

3.2.1 Открытие и создание файла

Для создания и открытия файла служит системный вызов sys_creat, который использует следующие аргументы: права доступа к файлу в регистре ECX, имя файла в EBX и номер системного вызова sys_creat (8) в EAX.

```
mov ecx, 0777o ; установка прав доступа
mov ebx, filename ; имя создаваемого файла
mov eax, 8 ; номер системного вызова `sys_creat`
int 80h ; вызов ядра
```

Для открытия существующего файла служит системный вызов sys_open, который использует следующие аргументы: права доступа к файлу в регистре EDX, режим доступа к файлу в регистр ECX, имя файла в EBX и номер системного вызова sys_open (5) в EAX. Среди режимов доступа к файлам чаще всего используются:

```
(0) – O_RDONLY (открыть файл в режиме только для чтения);
(1) – O_WRONLY – (открыть файл в режиме только записи);
(2) – O_RDWR – (открыть файл в режиме чтения и записи).
```

С другими режимами доступа можно ознакомиться в https://man7.org/. Системный вызов возвращает файловый дескриптор открытого файла в регистр EAX. В случае ошибки, код ошибки также будет находиться в регистре EAX.

```
mov ecx, 0 ; режим доступа (0 - только чтение)
mov ebx, filename ; имя открываемого файла
mov eax, 5 ; номер системного вызова `sys_open`
int 80h ; вызов ядра
```

3.2.2 Запись в файл

Для записи в файл служит системный вызов sys_write, который использует следующие аргументы: количество байтов для записи в регистре EDX, строку содержимого для записи ECX, файловый дескриптор в EBX и номер системного вызова sys_write (4) в EAX.
Системный вызов возвращает фактическое количество записанных байтов в регистр EAX. В случае ошибки, код ошибки также будет находиться в регистре EAX.
Прежде чем записывать в файл, его необходимо создать или открыть, что позволит получить дескриптор файла.

```
mov ecx, 0777o ; Создание файла.
mov ebx, filename ; в случае успешного создания файла,
mov eax, 8 ; в регистр eax запишется дескриптор файла
int 80h
mov edx, 12 ; количество байтов для записи
mov ecx, msg ; адрес строки для записи в файл
mov ebx, eax ; дескриптор файла
mov eax, 4 ; номер системного вызова `sys_write`
int 80h ; вызов ядра
```

3.2.3 Чтение файла

Для чтения данных из файла служит системный вызов sys_read, который использует следующие аргументы: количество байтов для чтения в регистре EDX, адрес в памяти для записи прочитанных данных в ECX, файловый дескриптор в EBX и номер системного вызова sys_read (3) в EAX. Как и для записи, прежде чем читать из файла, его необходимо открыть, что позволит получить дескриптор файла.

```
mov ecx, 0 ; Открытие файла.
```

```
mov ebx, filename ; в случае успешного открытия файла,
mov eax, 5 ; в регистр EAX запишется дескриптор файла
int 80h
mov edx, 12 ; количество байтов для чтения
mov ecx, fileCont ; адрес в памяти для записи прочитанных данных
mov ebx, eax ; дескриптор файла
mov eax, 3 ; номер системного вызова `sys_read`
int 80h ; вызов ядра
```

3.2.4 Закрытие файла

Для правильного закрытия файла служит системный вызов sys_close, который использует один аргумент – дескриптор файла в регистре EBX. После вызова ядра происходит удаление дескриптора файла, а в случае ошибки, системный вызов возвращает код ошибки в регистр EAX.

```
mov ecx, 0 ; Открытие файла.
mov ebx, filename ; в случае успешного открытия файла,
mov eax, 5 ; в регистр EAX запишется дескриптор файла
int 80h
mov ebx, eax ; дескриптор файла
mov eax, 6 ; номер системного вызова `sys_close`
int 80h ; вызов ядра
```

3.2.5 Изменение содержимого файла

Для изменения содержимого файла служит системный вызов sys_lseek, который использует следующие аргументы: исходная позиция для смещения EDX, значение смещения в байтах в ECX, файловый дескриптор в EBX и номер системного вызова sys_lseek (19) в EAX. Значение смещения можно задавать в байтах. Значения обозначающие исходную позиции могут быть следующими:

(0) – SEEK_SET (начало файла);

(1) – SEEK_CUR (текущая позиция);

(2) – SEEK_END (конец файла).

В случае ошибки, системный вызов возвращает код ошибки в регистр EAX.

```
mov ecx, 1 ; Открытие файла (1 - для записи).
mov ebx, filename
mov eax, 5
int 80h
mov edx, 2 ; значение смещения -- конец файла
mov ecx, 0 ; смещение на 0 байт
mov ebx, eax ; дескриптор файла
mov eax, 19 ; номер системного вызова `sys_lseek`
int 80h ; вызов ядра
mov edx, 9 ; Запись в конец файла
mov ecx, msg ; строки из переменной `msg`
mov eax, 4
int 80h
```

3.2.6 Удаление файла

Удаление файла осуществляется системным вызовом sys_unlink, который использует один аргумент – имя файла в регистре EBX.

```
mov ebx, filename ; имя файла
mov eax, 10 ; номер системного вызова `sys_unlink`
int 80h ; вызов ядра
```

В качестве примера приведем программу, которая открывает существующий файл, записывает в него сообщение и закрывает файл.

Результат работы программы:

```
user@dk4n31:~$ nasm -f elf -g -l main.lst main.asm
user@dk4n31:~$ ld -m elf_i386 -o main main.o
user@dk4n31:~$ ./main
Введите строку для записи в файл: Hello world!
user@dk4n31:~$ ls -l
-rwxrwxrwx 1 user user 20 Jul 2 13:06 readme.txt
-rwxrwxrwx 1 user user 11152 Jul 2 13:05 main
-rwxrwxrwx 1 user user 1785 Jul 2 13:03 main.asm
-rwxrwxrwx 1 user user 22656 Jul 2 13:05 main.lst
-rwxrwxrwx 1 user user 4592 Jul 2 13:05 main.o
user@dk4n31:~$ cat readme.txt
Hello world!
user@dk4n31:~$
```

4 Выполнение лабораторной работы

```
petlin@fedora:~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab10
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ touch lab10-1.asm readme-1.txt readme-2.txt
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ls
lab10-1.asm  presentation  readme-1.txt  readme-2.txt  report
```

Переходим в каталог для программам лабораторной работы № 10 и создаём файлы lab10-1.asm, readme-1.txt и readme-2.txt:

```
lab10-1.asm        [----] 10 L:[  1+36  37/ 37] *(1172/1172b) <EOF>
%include 'in_out.asm'
SECTION .data
 filename db 'readme-1.txt', 0h ; Имя файла
 msg db 'Введите строку для записи в файл: ', 0h ; Сообщение
SECTION .bss
 contents resb 255 ; переменная для вводимой строки
SECTION .text
global _start
_start:
 ; --- Печать сообщения `msg`
 mov eax,msg
 call sprint
 ; ---- Запись введеной с клавиатуры строки в `contents`
 mov ecx, contents
 mov edx, 255
 call sread
 ; --- Открытие существующего файла (`sys_open`)
 mov ecx, 2 ; открываем для записи (2)
 mov ebx, filename
 mov eax, 5
 int 80h
 ; --- Запись дескриптора файла в `esi`
 mov esi, eax
 ; --- Расчет длины введенной строки
 mov eax, contents ; в `eax` запишется количество
 call slen ; введенных байтов
 ; --- Записываем в файл `contents` (`sys_write`)
 mov edx, eax
 mov ecx, contents
 mov ebx, esi
 mov eax, 4
 int 80h
 ; --- Закрываем файл (`sys_close`)
 mov ebx, esi
 mov eax, 6
 int 80h
 call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ nasm -f elf lab10-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-1
Введите строку для записи в файл: this is a string
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ cat readme-1.txt
this is a string
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$
```

Вводим в файл lab10-1.asm текст программы из листинга 10.1 (Программа записи в файл сообщения). Создаём исполняемый файл и проверяем его работу. Программа работает корректно.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ chmod -x lab10-1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-1
bash: ./lab10-1: Permission denied
```

С помощью команды chmod изменяем права доступа к исполняемому файлу lab10-1, запретив его выполнение. Пытаемся выполнить файл, получаем

“permission denied”, потому что мы поставили запрет на выполнение программы.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ chmod -x lab10-1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ chmod +x lab10-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-1.asm
./lab10-1.asm: line 1: fg: no job control
./lab10-1.asm: line 2: SECTION: command not found
./lab10-1.asm: line 3: filename: command not found
./lab10-1.asm: line 3: Имя: command not found
./lab10-1.asm: line 4: msg: command not found
./lab10-1.asm: line 4: Сообщение: command not found
./lab10-1.asm: line 5: SECTION: command not found
./lab10-1.asm: line 6: contents: command not found
./lab10-1.asm: line 6: переменная: command not found
./lab10-1.asm: line 7: SECTION: command not found
./lab10-1.asm: line 8: global: command not found
./lab10-1.asm: line 9: _start:: command not found
./lab10-1.asm: line 10: syntax error near unexpected token `;'
./lab10-1.asm: line 10: ` ; --- Печать сообщения `msg`'
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$
```

С помощью команды chmod изменяем права доступа к файлу lab10-1.asm с исходным текстом программы, добавив права на исполнение. Пытаемся выполнить файл, это не удается, потому что такие файлы нужно компилировать в машинный код, а потом выполнять.

Вариант 6

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ chmod u=w,g=rx,o=w readme-1.txt
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ chmod 317 readme-2.txt
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ls -l
total 28
-rw-r--r--. 1 petlin petlin 3942 Nov  9 16:41 in_out.asm
-rw-r--r--. 1 petlin petlin 9164 Dec 14 21:19 lab10-1
-rwxr-xr-x. 1 petlin petlin 1172 Dec 14 21:17 lab10-1.asm
-rw-r--r--. 1 petlin petlin 1472 Dec 14 21:19 lab10-1.o
drwxr-xr-x. 1 petlin petlin  100 Nov 16 20:15 presentation
--w-r-x-w-. 1 petlin petlin   17 Dec 14 21:38 readme-1.txt
--wx--xrwx. 1 petlin petlin    0 Dec 14 21:09 readme-2.txt
drwxr-xr-x. 1 petlin petlin   62 Dec 14 21:08 report
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$
```

Используем команду chmod для предоставления нудных прав и проверяем сделанное командой ls -l.

5 Задание для самостоятельной работы

Создадим новый файл lab10-2.asm и напишем в нем программу работающую по следующему алгоритму:

- Вывод приглашения “Как Вас зовут?”
- ввести с клавиатуры свои фамилию и имя
- создать файл с именем name.txt
- записать в файл сообщение “Меня зовут”
- дописать в файл строку введенную с клавиатуры
- закрыть файл

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ touch lab10-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ls
in_out.asm  lab10-1  lab10-1.asm  lab10-1.o  lab10-2.asm  presentation  readme-1.txt  readme-2.txt  report
```

```
lab10-2.asm        [-M--]  8 L:[  1+29  30/ 52] *(474 / 745b) 0010 0x00A
%include 'in_out.asm'
SECTION .data
 filename db 'name.txt', 0
 msg db 'Как вас зовут? ', 0h
 msg2 db 'Меня зовут ', 0h
SECTION .bss
 contents: resb 80
SECTION .text
global _start
_start:
 mov eax,msg
 call sprintLF
 mov ecx, contents
 mov edx, 80
 call sread
 mov ecx, 0777o
 mov ebx, filename
 mov eax, 8
 int 80h
 mov esi, eax
 mov eax, msg2
 call slen
 mov edx, eax
 mov ecx, msg2
 mov ebx, esi
 mov eax, 4
 int 80h
 mov ebx, esi
 mov eax, 6
 int 80h
 mov ecx, 1
 mov ebx, filename
 mov eax, 5
 int 80h
```

```
mov esi, eax
mov edx, 2
mov ecx, 0
mov ebx, eax
mov eax, 19
int 80h
mov eax, contents
call slen
mov edx, eax
mov ecx, contents
mov ebx, esi
mov eax, 4
int 80h
mov ebx, esi
mov eax, 6
int 80h
call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ls
in_out.asm  lab10-1  lab10-1.asm  lab10-1.o  lab10-2  lab10-2.asm  lab10-2.o  presentation  readme-1.txt  readme-2.txt  report
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ nasm -f elf lab10-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ld -m elf_i386 -o lab10-2 lab10-2.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-2
Как вас зовут?
Петлин Артём
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ ls
in_out.asm  lab10-1  lab10-1.asm  lab10-1.o  lab10-2  lab10-2.asm  lab10-2.o  name.txt  presentation  readme-1.txt  readme-2.txt  report
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab10$ cat name.txt
Меня зовут Петлин Артём
```

Программа работает корректно

6 Выводы

Мы приобрели навыки написания программ для работы с файлами.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).