

Отчет по лабораторной работе №11

Операционные системы

Фадин В.В.

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Задача 1	6
2.2	Задача 2	8
2.3	Задача 3	10
3	Выводы	12
4	Ответы на онтрольные вопросы	13

Список иллюстраций

2.1	semaphore.sh	6
2.2	Доработка скрипта semaphore.sh	7
2.3	Пример запуска трех процессов	8
2.4	Реализация man_command.sh	9
2.5	Вызов man_command.sh	10
2.6	Реализация random.sh	11
2.7	Вызов random.sh	11

Список таблиц

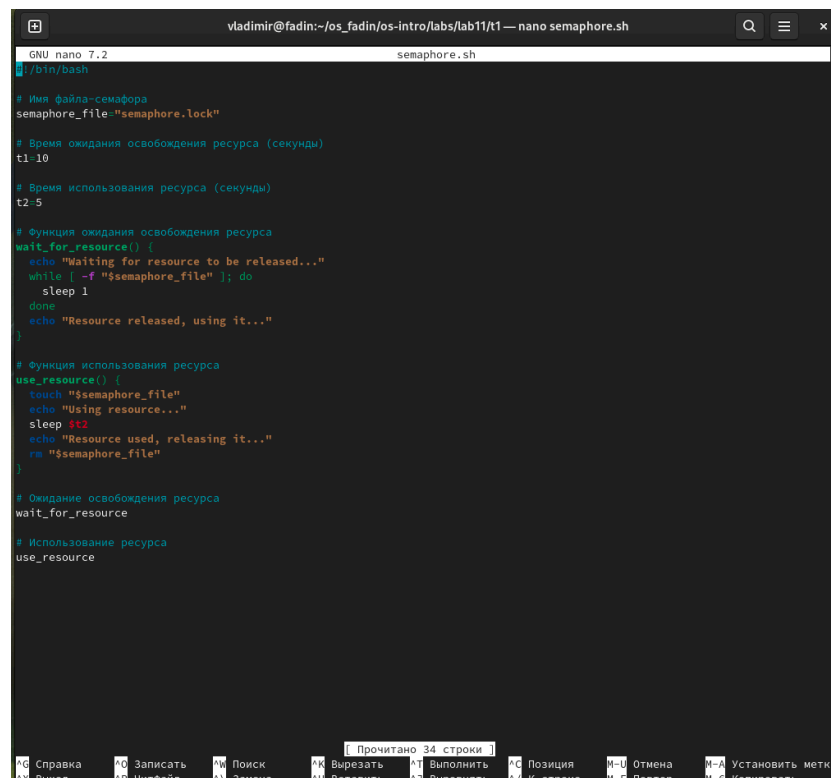
1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Выполнение лабораторной работы

2.1 Задача 1

Вот пример командного файла, реализующего упрощенный механизм семафоров:



```
GNU nano 7.2 semaphore.sh
/bin/bash

# Имя файла-семафора
semaphore_file="semaphore.lock"

# Время ожидания освобождения ресурса (секунды)
t1=10

# Время использования ресурса (секунды)
t2=5

# Функция ожидания освобождения ресурса
wait_for_resource() {
    echo "Waiting for resource to be released..."
    while [ -f "$semaphore_file" ]; do
        sleep 1
    done
    echo "Resource released, using it..."
}

# Функция использования ресурса
use_resource() {
    touch "$semaphore_file"
    echo "Using resource..."
    sleep $t2
    echo "Resource used, releasing it..."
    rm "$semaphore_file"
}

# Ожидание освобождения ресурса
wait_for_resource

# Использование ресурса
use_resource
```

Рис. 2.1: semaphore.sh

Командный файл создает файл-семафор semaphore.lock для синхронизации доступа к ресурсу. Функция wait_for_resource ожидает освобождения ресурса,

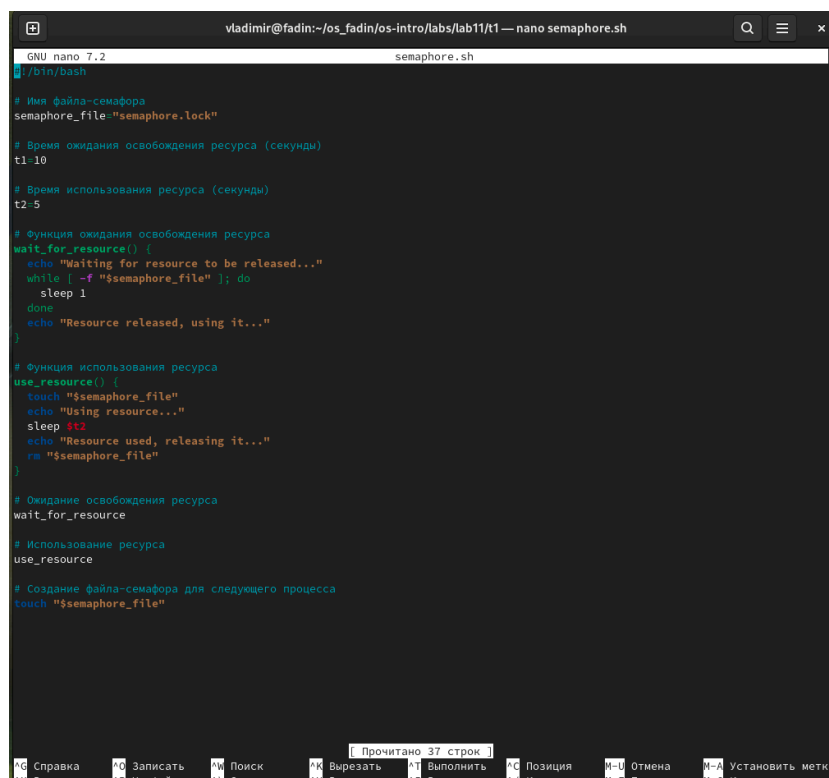
проверяя наличие файла-семафора, и выводит сообщение о ожидании. Функция `use_resource` использует ресурс, выводит сообщение о его использовании, и после использования ресурса удаляет файл-семафор.

Чтобы запустить командный файл в привилегированном режиме, мы можем использовать следующую команду:

```
sudo ./semaphore.sh
```

Здесь мы запускаем командный файл с привилегиями суперпользователя (`sudo`).

Чтобы доработать программу для взаимодействия трех и более процессов, мы можем использовать следующий подход:



```
GNU nano 7.2 semaphore.sh
#!/bin/bash

# Имя файла-семафора
semaphore_file="semaphore.lock"

# Время ожидания освобождения ресурса (секунды)
t1=10

# Время использования ресурса (секунды)
t2=5

# Функция ожидания освобождения ресурса
wait_for_resource() {
    echo "Waiting for resource to be released..."
    while [ -f "$semaphore_file" ]; do
        sleep 1
    done
    echo "Resource released, using it..."
}

# Функция использования ресурса
use_resource() {
    touch "$semaphore_file"
    echo "Using resource..."
    sleep $t2
    echo "Resource used, releasing it..."
    rm "$semaphore_file"
}

# Ожидание освобождения ресурса
wait_for_resource

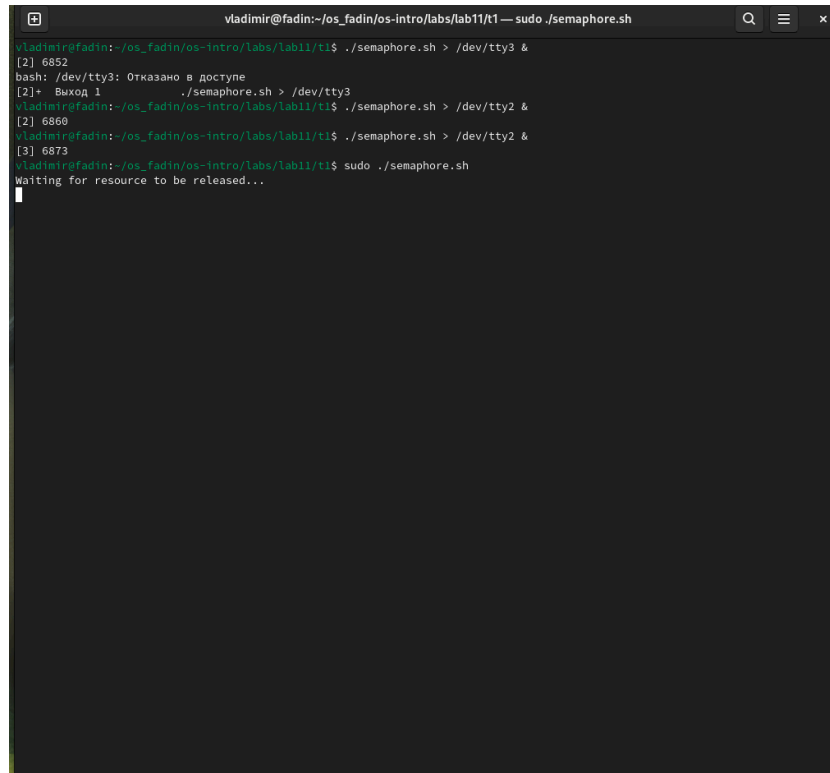
# Использование ресурса
use_resource

# Создание файла-семафора для следующего процесса
touch "$semaphore_file"
```

Рис. 2.2: Доработка скрипта `semaphore.sh`

В этом примере мы создаем файл-семафор после использования ресурса, чтобы следующий процесс мог ожидать его освобождения. Таким образом, мы мо-

жем запустить несколько процессов, которые будут ожидать освобождения ресурса и использовать его по очереди.



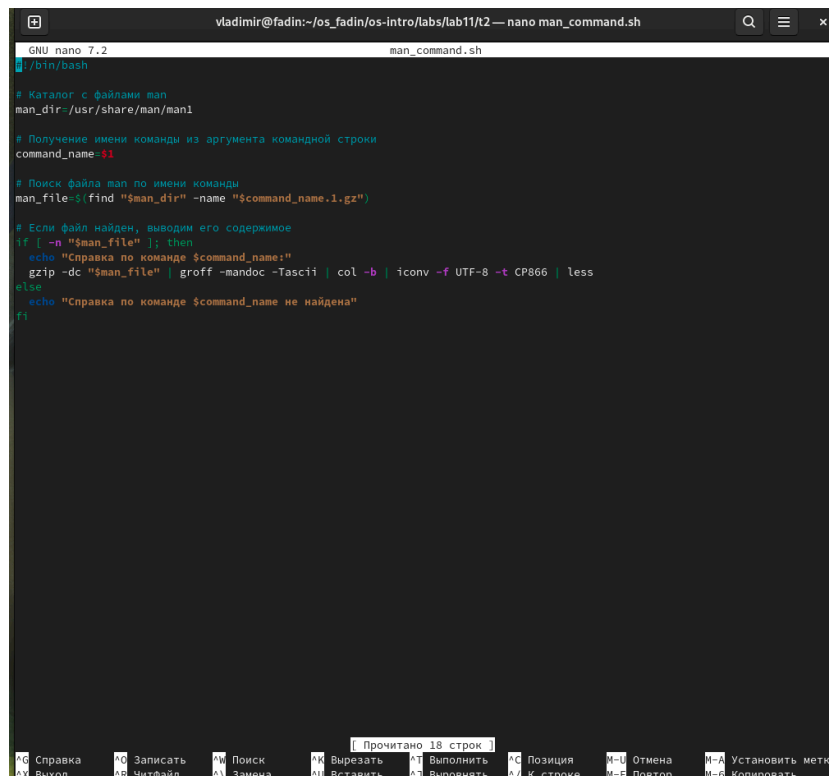
```
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t1 — sudo ./semaphore.sh
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t1$ ./semaphore.sh > /dev/tty3 &
[2] 6852
bash: /dev/tty3: Отказано в доступе
[2]+  Выход 1 ./semaphore.sh > /dev/tty3
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t1$ ./semaphore.sh > /dev/tty2 &
[2] 6860
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t1$ ./semaphore.sh > /dev/tty2 &
[3] 6873
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t1$ sudo ./semaphore.sh
Waiting for resource to be released...
```

Рис. 2.3: Пример запуска трех процессов

Здесь мы запускаем три процесса, каждый из которых будет ожидать освобождения ресурса и использовать его по очереди.

2.2 Задача 2

Командный файл, реализующий команду `map`:



```
GNU nano 7.2 man_command.sh
/bin/bash

# Каталог с файлами man
man_dir=/usr/share/man/man1

# Получение имени команды из аргумента командной строки
command_name=$1

# Поиск файла man по имени команды
man_file=$(find "$man_dir" -name "$command_name.1.gz")

# Если файл найден, выводим его содержимое
if [ -n "$man_file" ]; then
    echo "Справка по команде $command_name:"
    gzip -dc "$man_file" | groff -mandoc -Tascii | col -b | iconv -f UTF-8 -t CP866 | less
else
    echo "Справка по команде $command_name не найдена"
fi
```

Рис. 2.4: Реализация man_command.sh

Командный файл получает в виде аргумента командной строки название команды и в виде результата выдает справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге man1.

Пример использования:

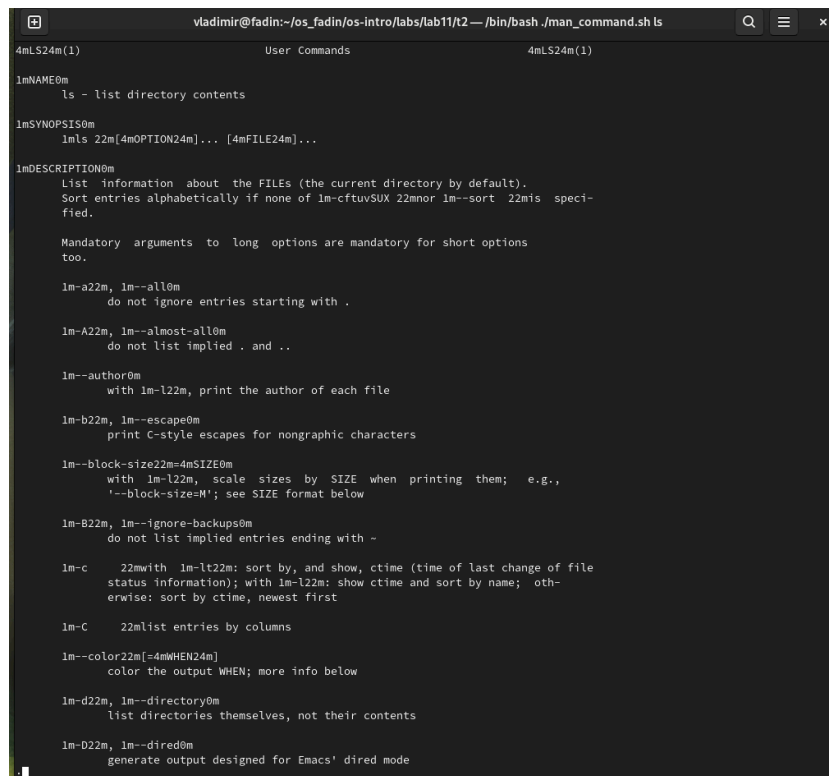
A terminal window titled 'vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t2 — /bin/bash ./man_command.sh ls'. The window shows the output of the 'man ls' command. The output is displayed in a less-like format with headers like '1mLS24m(1)', 'User Commands', and '4mLS24m(1)'. The main content includes the command 'ls - list directory contents', a synopsis '1mSYNOPSIS0m', and a detailed description '1mDESCRIPTION0m'. The description explains that 'ls' lists information about files in the current directory by default, sorts entries alphabetically, and lists mandatory arguments for long options. It also lists various options like '-a', '-A', '-l', '-lR', '-s', '-t', '-c', '-C', '-p', '-d', and '-D' with their functions. The terminal has a dark background and a light-colored text color.

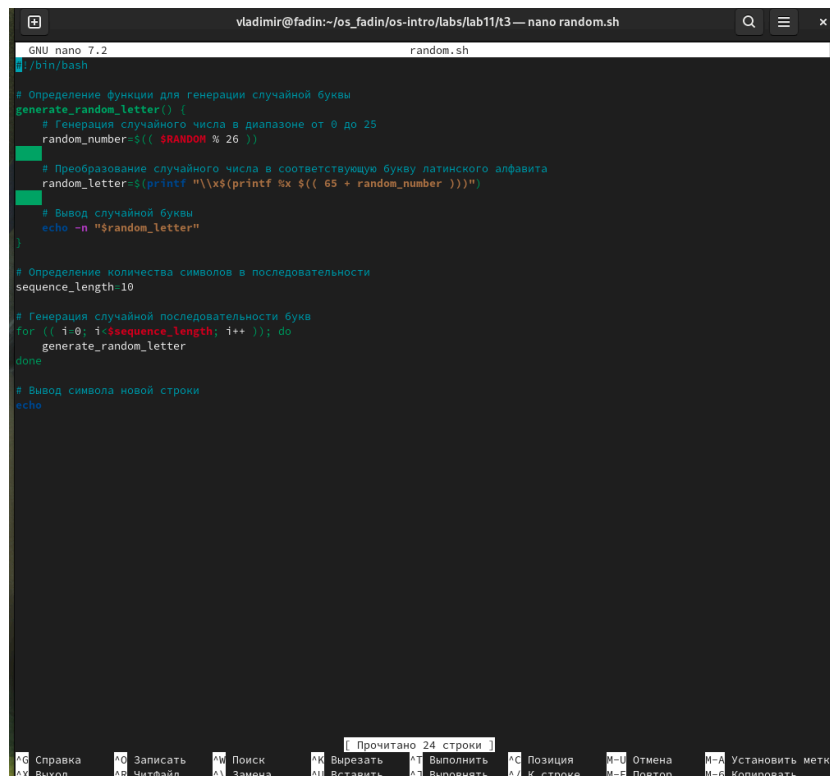
Рис. 2.5: Вызов man_command.sh

В этом примере командный файл будет искать файл `ls.1.gz` в каталоге `/usr/share/man/man1` и выводить его содержимое с помощью `less`, если файл найден. Если файл не найден, будет выдано сообщение об отсутствии справки.

Также видно, что в тексте присутствуют иероглифы. Решение устранения пока найти не удалось.

2.3 Задача 3

Реализация скрипта:



```
GNU nano 7.2 random.sh
/bin/bash

# Определение функции для генерации случайной буквы
generate_random_letter() {
    # Генерация случайного числа в диапазоне от 0 до 25
    random_number=$(( $RANDOM % 26 ))

    # Преобразование случайного числа в соответствующую букву латинского алфавита
    random_letter=$(printf "%c$(printf %x $(( 65 + random_number )))")

    # Вывод случайной буквы
    echo -n "$random_letter"
}

# Определение количества символов в последовательности
sequence_length=10

# Генерация случайной последовательности букв
for (( i=0; i<sequence_length; i++ )); do
    generate_random_letter
done

# Вывод символа новой строки
echo
```

Рис. 2.6: Реализация random.sh

Этот скрипт генерирует случайную последовательность букв латинского алфавита, используя переменную \$RANDOM для генерации случайного числа между 0 и 25, а затем используя команду printf для преобразования числа в букву латинского алфавита (A-Z). Функция generate_random_letter вызывается 10 раз для генерации последовательности из 10 случайных букв.



```
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t3$ ./random.sh
VGEACEHPYK
vladimir@fadin:~/os_fadin/os-intro/labs/lab11/t3$
```

Рис. 2.7: Вызов random.sh

3 Выводы

Мы изучили основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

4 Ответы на онтрольные вопросы

Вот ответы на контрольные вопросы:

1. Синтаксическая ошибка в строке `while [$1 != "exit"]` заключается в отсутствии пробелов между скобками и переменными/операторами. Это должно быть `while ["$1" != "exit"]`.
2. Чтобы объединить несколько строк в одну, вы можете использовать следующий синтаксис: `result="${str1}${str2}${str3}"`.
3. Утилита `seq` генерирует последовательность чисел. Его можно заменить циклом `for` или использовать синтаксис `{start..end}` в `bash`. Например, `for ((i=1; i<=10; i++)); do echo $i; done` или «`echo {1..10}`».
4. Результатом выражения `$((10/3))` является 3.
5. Основные различия между `zsh` и `bash`:
 - `zsh` имеет более продвинутые функции завершения и подстановки.
 - `zsh` имеет более мощный синтаксис для сценариев оболочки.
 - `zsh` имеет лучшую поддержку Unicode и интернационализации.
 - `zsh` имеет более настраиваемую подсказку.
6. Синтаксис `for ((a=1; a <= LIMIT; a++))` верен.
7. `Bash` часто сравнивают с другими языками сценариев, такими как `Perl`, `Python` и `Ruby`. Преимущество `Bash` заключается в тесной интеграции со средой командной строки `Unix/Linux`, что делает его мощным инструментом для задач системного администрирования и автоматизации. Однако ему может не хватать некоторых функций и гибкости языков программирования более общего назначения.