Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное

учреждение высшего образования

Московский технический университет связи и информатики

Кафедра «Системного программирования»

Лабораторная работа №5

по дисциплине «Операционные системы»

«Управление памятью. Работа с файлами и каталогами ОС GNU/LINUX»

Выполнила: студентка группы БСТ2104

Первухина А. А.

Проверила: Алексанян Диана Ашотовна

Москва 2023

**Содержание**

[**Задание 1. Управление памятью** 3](#_Toc151662668)

[**Задание 1.1. Получение карты виртуальной памяти процесса** 3](#_Toc151662669)

[**Задание 1.2. Настройка файла подкачки** 5](#_Toc151662670)

[**Задание 1.3. Тестирование памяти** 8](#_Toc151662671)

[**Задание 1.4. Дополнительное задание** 9](#_Toc151662672)

[**Задание 2. Работа с файлами и каталогами ОС GNU/LINUX** 10](#_Toc151662673)

[**Задание 2.1. Разработка программы построения карты ВАП процесса** 10](#_Toc151662674)

[**Задание 2.2. Разработка программы просмотра содержимого файла** 13](#_Toc151662675)

[**Вывод** 15](#_Toc151662676)

# **Задание 1. Управление памятью**

**Цели работы:**

* изучить основные методы управления памятью в Linux;
* построить карту виртуальной памяти процесса;
* приобрести практические навыки работы с инструментами мониторинга использования памяти, настройки параметров виртуальной памяти, тестирования работоспособности оперативной памяти.

# **Задание 1.1. Получение карты виртуальной памяти процесса**

С помощью **pmap** выведите карту виртуальной памяти любого работающего процесса, например, экземпляра терминала. Изучите структуру карты, сделайте вывод, какая информация может быть получена в результате работы **pmap**.

Результат выполнения представлен на рисунке 1:

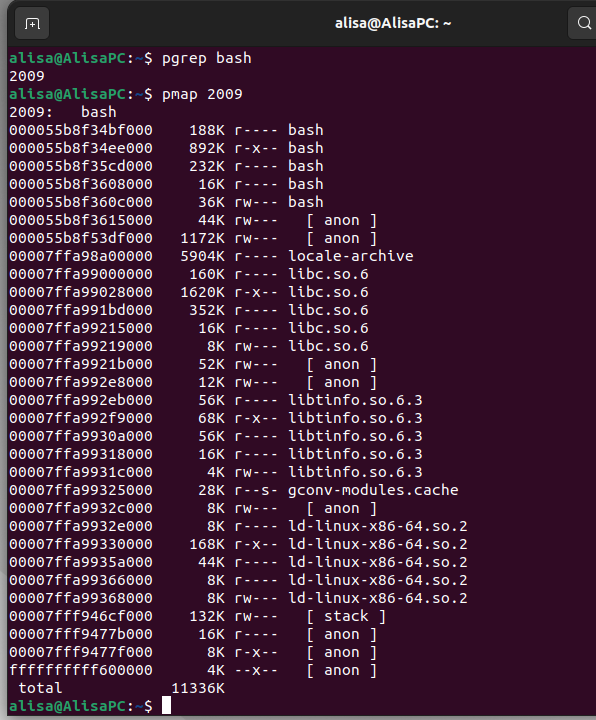


Рисунок 1– Карта виртуальной памяти процесса терминала

Информация, полученная в результате работы **pmap**:

1. адрес начала сегмента виртуальной памяти;
2. размер сегмента виртуальной памяти;
3. права доступа к сегменту виртуальной памяти;
4. имя файла или библиотеки сегмента виртуальной памяти.

# **Задание 1.2. Настройка файла подкачки**

1. Проверьте текущее использование памяти с помощью команды **free**.
2. Создайте файл подкачки размером 1 ГБ.
3. Установите требуемые разрешения и отформатируйте файл подкачки с применением **mkswap**.
4. Подключите созданный файл подкачки с применением команды **swapon**.
5. Убедитесь, что размер swap-файла изменился.
6. Добавьте настройку для автоматического подключения swap-файла при перезагрузке виртуальной машины (необходимо внести изменения в файл /etc/fstab), перезагрузите машину и убедитесь в корректности работы.
7. Выполните команды **free -h -t** и **swapon -s**.
8. Деактивируйте созданный файл подкачки и удалите его.

Проверьте доступное количество свободной памяти.

Результаты выполнения представлены на рисунках 2-4:

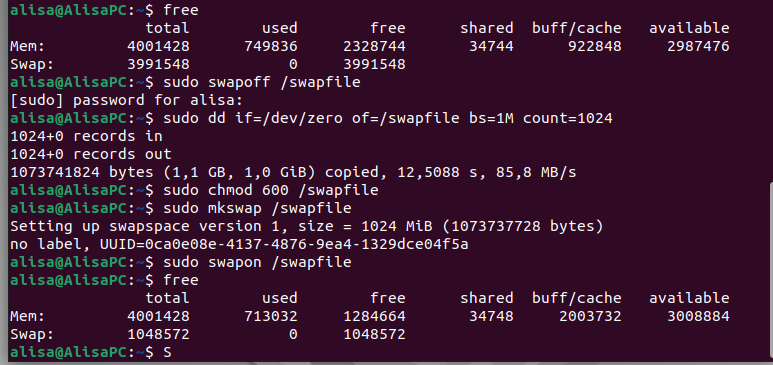


Рисунок 2 – Проверка использования памяти, создание, форматирование и подключение файла подкачки

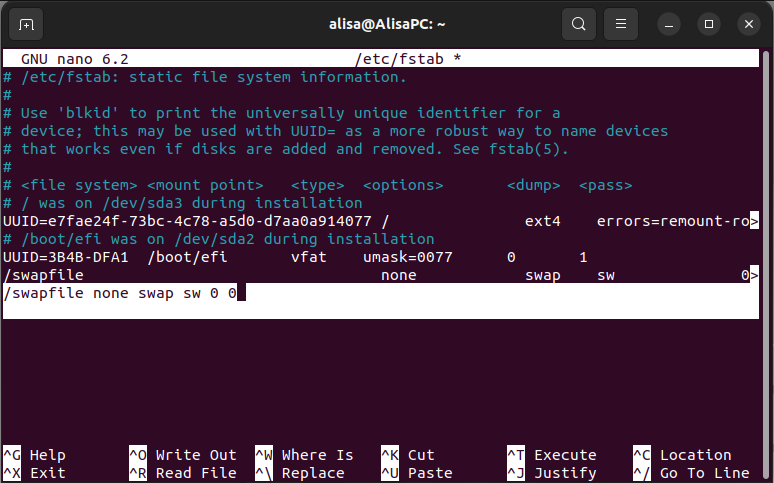


Рисунок 3 – Настройка автоматического подключения swap-файла

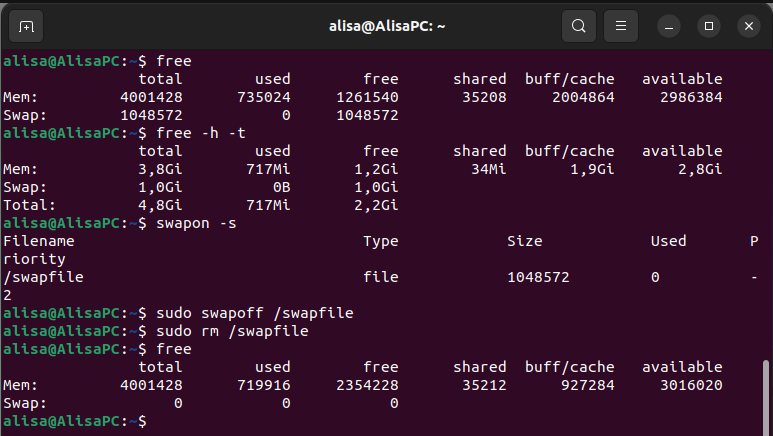


Рисунок 4 – Проверка использования памяти, деактивация и удаление созданного файла подкачки

# **Задание 1.3. Тестирование памяти**

С помощью утилиты memtester проведите тестирование 1 ГБ свободной памяти (если данный объём недоступен, выберите меньшее количество свободной памяти), выполнив не менее двух проходов.

Результат выполнения представлен на рисунке 5:

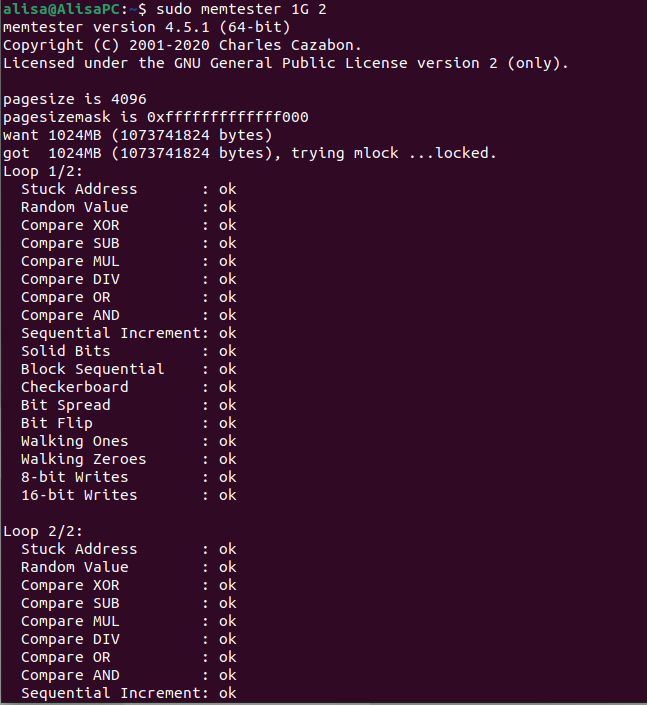


Рисунок 5 – Тестирование 1 ГБ свободной памяти

# **Задание 1.4. Дополнительное задание**

Получите текущее значение параметра **swappiness**. Измените его. На что влияет значение этого параметра?

Результат выполнения представлен на рисунке 6:

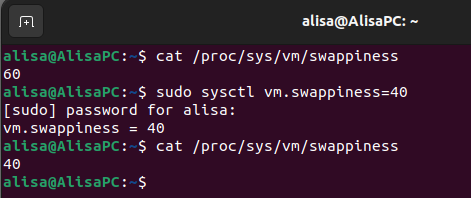


Рисунок 6 – Получение и изменение параметра **swappiness**

**Swappiness** определяет, насколько активно ядро Linux будет использовать область подкачки. Значение этого параметра находится в диапазоне от 0 до 100, где 0 означает минимальное использование подкачки, а 100 максимальное.

* Значение 0. Компьютер будет стараться избегать использования области подкачки, сохраняя данные в оперативной памяти столько, насколько это возможно.
* Значение 100. Компьютер будет активно использовать область подкачки, чтобы освободить оперативную память, когда она начинает заполняться.

Изменение параметра **swappiness** может применяться для оптимального использования ресурсов в зависимости от конкретных потребностей.

# **Задание 2. Работа с файлами и каталогами ОС GNU/LINUX**

**Цели работы:**

* построить карту виртуальной памяти процесса;
* изучить основные системные вызовы и функции в ОС GNU/LINUX для работы с файлами и каталогами.

# **Задание 2.1. Разработка программы построения карты ВАП процесса**

Разработать программу, строящую карту виртуальной памяти для процесса, идентификатор которого передаётся в качестве аргумента при запуске программы. Карту памяти необходимо отобразить на экране или сохранить в текстовом файле (имя файла передаётся в качестве аргумента при запуске, файл разместите в одной директории с программой).

Предусмотрите обработку ситуации, когда файл с указанным именем уже существует в директории и предложите пользователю выбор: заменить файл на новый или переименовать результирующий файл. Реализуйте контроль ошибок открытия/закрытия/чтения файла.

Результаты выполнения представлены на рисунках 7-8:

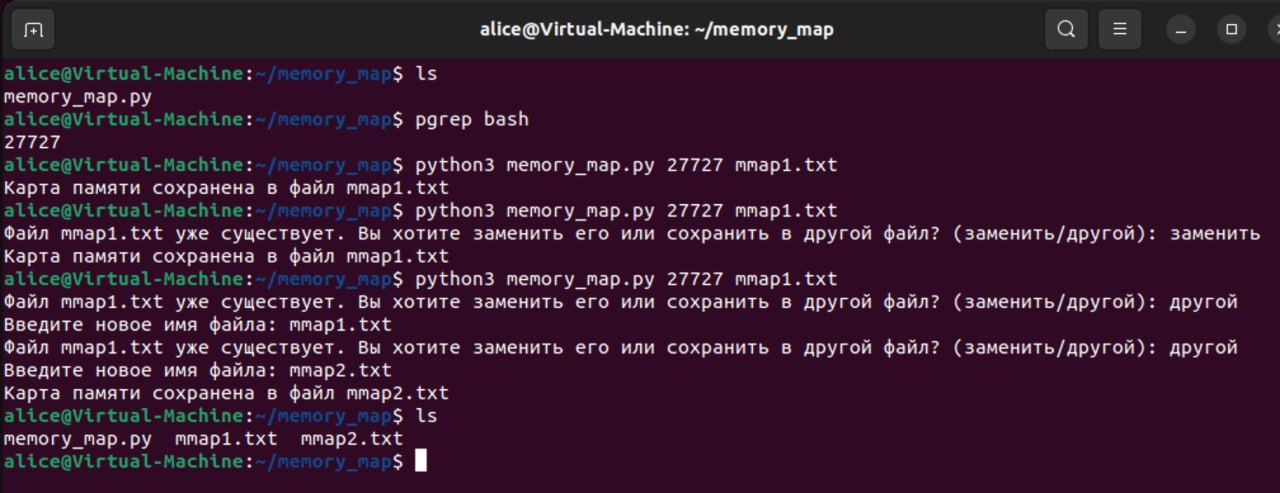


Рисунок 7 – Создание и запуск программы «memory\_map.py»

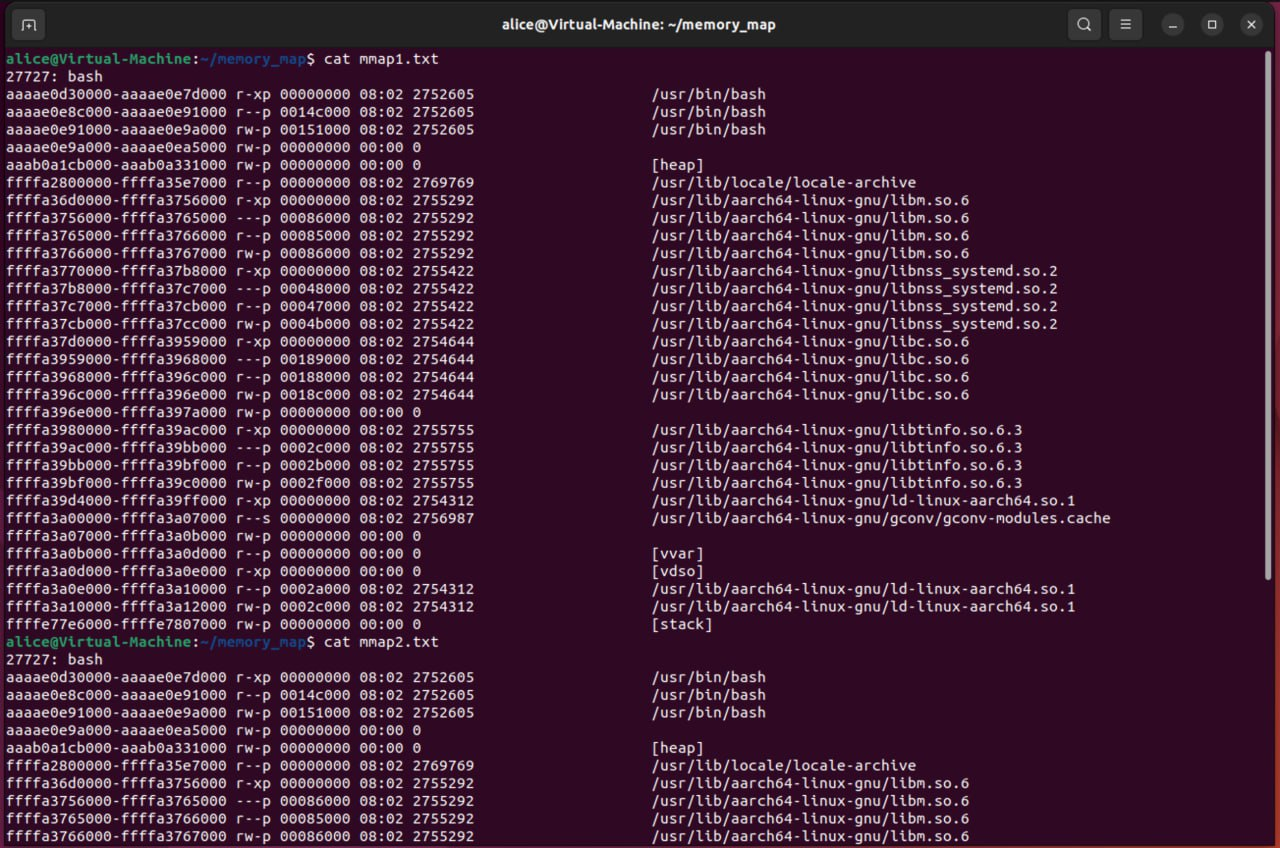


Рисунок 8 – Результаты работы программы «memory\_map.py»

Текст программы «memory\_map.py»

import os

import sys

def save\_memory\_map\_to\_file(process\_id, output\_file):

try:

# Путь к файлу maps процесса

maps\_file\_path = f"/proc/{process\_id}/maps"

# Чтение информации о карте памяти

with open(maps\_file\_path, 'r') as maps\_file:

maps\_data = maps\_file.read()

# Запись информации в файл

with open(output\_file, 'w') as file:

file.write(f"{process\_id}: {process\_name(process\_id)}\n")

file.write(maps\_data)

print(f"Карта памяти сохранена в файл {output\_file}")

except FileNotFoundError:

print(f"Процесс с идентификатором {process\_id} не найден.")

except Exception as e:

print(f"Произошла ошибка: {e}")

def process\_name(process\_id):

# Получение имени процесса из /proc/[pid]/comm

comm\_file\_path = f"/proc/{process\_id}/comm"

with open(comm\_file\_path, 'r') as comm\_file:

return comm\_file.read().strip()

def main():

if len(sys.argv) != 3:

print("Использование: python3 process\_memory\_map.py <process\_id> <output\_file>")

sys.exit(1)

process\_id = int(sys.argv[1])

output\_file = sys.argv[2]

while os.path.exists(output\_file):

user\_choice = input(f"Файл {output\_file} уже существует. Вы хотите заменить его или сохранить в другой файл? (заменить/другой): ").lower()

if user\_choice == 'заменить':

break

elif user\_choice == 'другой':

new\_name = input("Введите новое имя файла: ")

output\_file = new\_name

else:

print("Некорректный выбор. Пожалуйста, выберите 'заменить' или 'другой'.")

try:

save\_memory\_map\_to\_file(process\_id, output\_file)

except Exception as e:

print(f"Произошла ошибка: {e}")

sys.exit(1)

if \_\_name\_\_ == "\_\_main\_\_":

main()

# **Задание 2.2. Разработка программы просмотра содержимого файла**

Разработать программу вывода содержимого текстового файла на экран (в качестве аргумента при запуске программы передаётся имя файла, второй аргумент (**N**) устанавливает вывод по группам строк (по **N**-строк) или сплошным текстом (**N**=0)). Для вывода очередной группы строк необходимо ожидать нажатия пользователем любой клавиши. Для чтения или записи файла используйте только функции посимвольного ввода-вывода.

Предусмотрите контроль ошибок открытия/закрытия/чтения/записи файла.

Результаты выполнения представлены на рисунке 9:

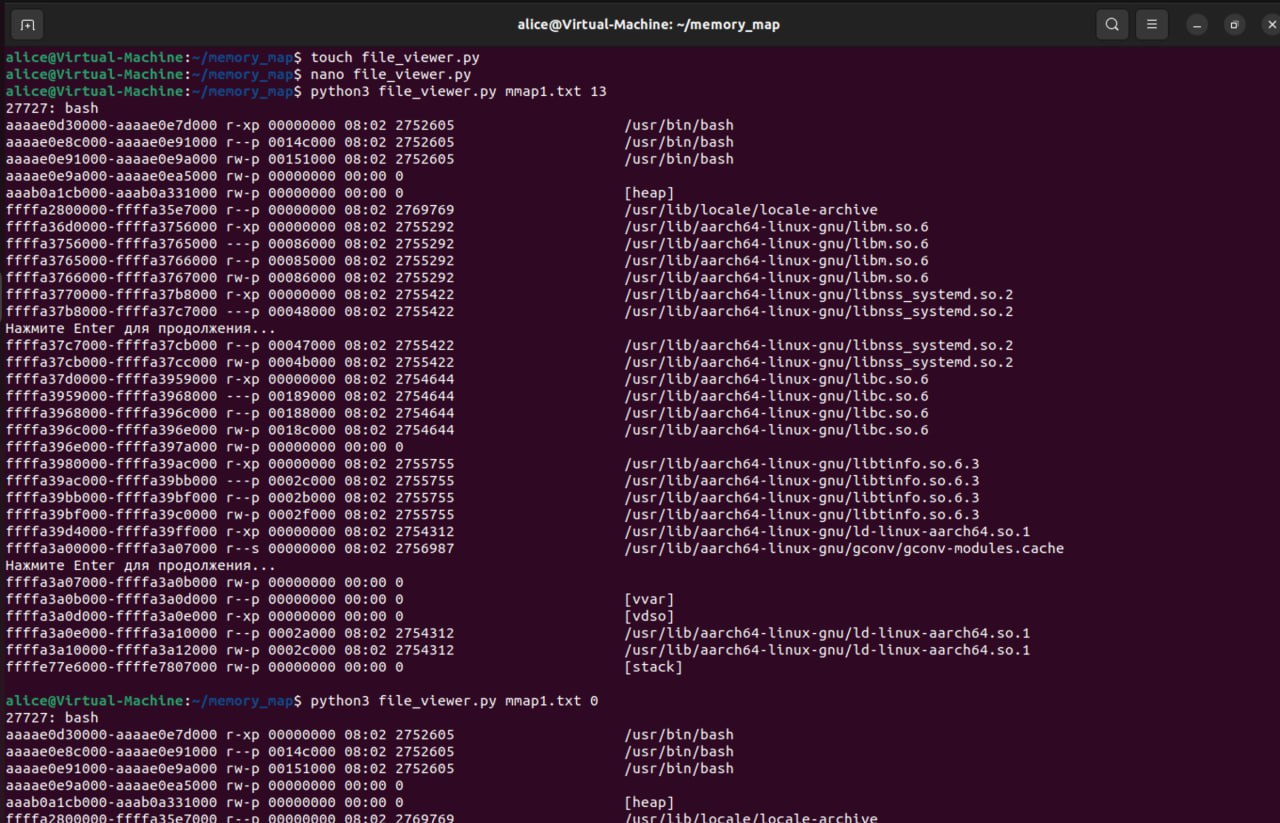


Рисунок 9 – Результаты работы программы «file\_viewer.py»

Текст программы «file\_viewer.py»

def read\_file(file\_name):

try:

# Попытка открыть файл для чтения

file = open(file\_name, 'r')

# Чтение содержимого файла

content = file.read()

# Закрытие файла

file.close()

return content

except IOError:

# Обработка ошибки при открытии файла

print("Ошибка открытия файла.")

return None

def display\_content(content, group\_size):

if content is not None:

if group\_size == 0:

# Если размер группы равен 0, выводим весь текст

print(content)

else:

# Разбиваем содержимое файла на строки

lines = content.split('\n')

# Итерируем по строкам с шагом group\_size

for i in range(0, len(lines), group\_size):

# Выбираем группу строк

group = lines[i:i + group\_size]

# Выводим группу строк на экран

print('\n'.join(group))

# Если не достигнут конец файла, ожидаем нажатия Enter

if i + group\_size < len(lines):

input("Нажмите Enter для продолжения...")

if \_\_name\_\_ == "\_\_main\_\_":

import sys

if len(sys.argv) != 3:

# Проверка наличия двух аргументов командной строки

print("Использование: python program\_name.py <file\_name> <group\_size>")

sys.exit(1)

file\_name = sys.argv[1]

try:

group\_size = int(sys.argv[2])

except ValueError:

# Обработка ошибки при некорректном вводе размера группы

print("Второй аргумент должен быть целым числом.")

sys.exit(1)

content = read\_file(file\_name)

if content is not None:

# Вывод содержимого файла на экран с учетом размера группы строк

display\_content(content, group\_size)

# **Вывод**

В результате проделанной лабораторной работы были изучены основные методы управления памятью, а также основные системные вызовы и функции для работы с файлами и каталогами, построены карты виртуальной памяти процессов, а также приобретены практические навыки работы с инструментами мониторинга использования памяти, настройки параметров виртуальной памяти и тестирования работоспособности оперативной памяти в ОС GNU/LINUX.