МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ  
КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Поволжский Государственный Университет Телекоммуникаций и Информатики

Отчет по практической работе № 2

по дисциплине «Операционные системы и оболочки»

Выполнили:  
студент группы ИВТ-34У  
Горлов А.А.

Самара, 2024

Задание

Все варианты должны иметь графический интерфейс.

Разработать приложение, которое позволяет чтение и запись данных из адресного

пространства процесса.

Пользователь самостоятельно указывается адресное

пространство в программе.

Для поиска нужного адреса рекомендуется использовать бесплатную программу scanmem или gameconqueror

Четные номера вариантов работают с блокнотом, нечетные с любой

компьютерной игрой.

Дополнительно позволить запуск из приложения собственного

терминала (аналог команд Linux). Добавить по 10 команд для работы с

памятью ОС, дать им пояснение.

Вариант 7. Значение размера используемой оперативной памяти

Файл main

import sys

from PyQt5.QtWidgets import QApplication, QTabWidget

from memory\_app import MemoryApp

from terminal\_app import CustomTerminal

class CombinedApp(QTabWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.memory\_tab = MemoryApp()

        self.terminal\_tab = CustomTerminal()

        self.addTab(self.memory\_tab, "Чтение/Запись памяти")

        self.addTab(self.terminal\_tab, "Терминал")

        self.setWindowTitle('Memory & Terminal App')

def main():

    app = QApplication(sys.argv)

    combined\_app = CombinedApp()

    combined\_app.show()

    sys.exit(app.exec\_())

if \_\_name\_\_ == '\_\_main\_\_':

    main()

файл терминала

import psutil

import os

import struct

from PyQt5.QtWidgets import QWidget, QVBoxLayout, QLineEdit, QTextEdit, QLabel

def read\_from\_memory(pid, address, size=4):

    try:

        with open(f"/proc/{pid}/mem", 'rb', 0) as mem\_file:

            mem\_file.seek(address)

            return mem\_file.read(size)

    except Exception as e:

        return str(e)

def write\_to\_memory(pid, address, value):

    try:

        with open(f"/proc/{pid}/mem", 'wb', 0) as mem\_file:

            mem\_file.seek(address)

            mem\_file.write(value)

        return "Запись успешно завершена."

    except Exception as e:

        return str(e)

class CustomTerminal(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.attached\_process = None  # Процесс, к которому мы "присоединяемся"

        self.initUI()

    def initUI(self):

        layout = QVBoxLayout()

        self.command\_input = QLineEdit(self)

        self.command\_input.setPlaceholderText("Введите команду")

        self.command\_input.returnPressed.connect(self.process\_command)

        layout.addWidget(QLabel("Терминал:"))

        layout.addWidget(self.command\_input)

        self.output = QTextEdit(self)

        self.output.setReadOnly(True)

        layout.addWidget(QLabel("Вывод:"))

        layout.addWidget(self.output)

        self.setLayout(layout)

    def append\_output(self, text):

        self.output.append(text)

    def process\_command(self):

        command = self.command\_input.text().strip()

        if command:

            self.append\_output(f"Выполнение команды: {command}")

            self.command\_input.clear()

            args = command.split()

            cmd = args[0].lower()

            if cmd == "meminfo":

                self.meminfo()

            elif cmd == "readmem" and len(args) == 2:

                self.readmem(args[1])

            elif cmd == "writemem" and len(args) == 3:

                self.writemem(args[1], args[2])

            elif cmd == "processlist":

                self.processlist()

            elif cmd == "killprocess" and len(args) == 2:

                self.killprocess(args[1])

            elif cmd == "memusage":

                self.memusage()

            elif cmd == "findprocess" and len(args) == 2:

                self.findprocess(args[1])

            elif cmd == "attach" and len(args) == 2:

                self.attach(args[1])

            elif cmd == "detach":

                self.detach()

            elif cmd == "help":

                self.show\_help()

            else:

                self.append\_output("Ошибка: неизвестная команда или неверное количество аргументов.")

    def meminfo(self):

        mem = psutil.virtual\_memory()

        self.append\_output(f"Общая память: {mem.total}, Свободная память: {mem.available}, Используется: {mem.used}")

    def readmem(self, address):

        try:

            if not self.attached\_process:

                self.append\_output("Ошибка: необходимо присоединиться к процессу.")

                return

            addr = int(address, 16)

            pid = self.attached\_process.pid

            data = read\_from\_memory(pid, addr)

            if isinstance(data, bytes):

                value = struct.unpack('I', data)[0]  # Читаем 4 байта как целое число

                self.append\_output(f"Чтение данных по адресу {address}: значение {value}")

            else:

                self.append\_output(f"Ошибка при чтении: {data}")

        except ValueError:

            self.append\_output(f"Ошибка: неверный адрес {address}")

    def writemem(self, address, value):

        try:

            if not self.attached\_process:

                self.append\_output("Ошибка: необходимо присоединиться к процессу.")

                return

            addr = int(address, 16)

            pid = self.attached\_process.pid

            value\_to\_write = struct.pack('I', int(value))  # Записываем 4-байтовое целое число

            result = write\_to\_memory(pid, addr, value\_to\_write)

            self.append\_output(f"Запись значения {value} по адресу {address}: {result}")

        except ValueError:

            self.append\_output(f"Ошибка: неверный адрес {address}")

    def processlist(self):

        processes = psutil.process\_iter(['pid', 'name'])

        for proc in processes:

            self.append\_output(f"PID: {proc.info['pid']}, Имя: {proc.info['name']}")

    def killprocess(self, pid):

        try:

            p = psutil.Process(int(pid))

            p.terminate()

            self.append\_output(f"Процесс {pid} завершен.")

        except Exception as e:

            self.append\_output(f"Ошибка при завершении процесса {pid}: {str(e)}")

    def memusage(self):

        try:

            process = psutil.Process(os.getpid())

            mem\_info = process.memory\_info()

            self.append\_output(f"Использование памяти приложением: {mem\_info.rss} байт")

        except Exception as e:

            self.append\_output(f"Ошибка при получении использования памяти: {str(e)}")

    def findprocess(self, name):

        found = False

        for proc in psutil.process\_iter(['pid', 'name']):

            if proc.info['name'] == name:

                self.append\_output(f"Найден процесс: PID {proc.info['pid']}, Имя {proc.info['name']}")

                found = True

        if not found:

            self.append\_output(f"Процесс с именем {name} не найден.")

    def attach(self, pid):

        try:

            os.system(f"sudo gdb -p {pid}")  # Используем gdb для присоединения к процессу

            self.attached\_process = psutil.Process(int(pid))

            self.append\_output(f"Присоединение к процессу {pid}")

        except Exception as e:

            self.append\_output(f"Ошибка при присоединении к процессу: {str(e)}")

    def detach(self):

        if self.attached\_process:

            try:

                os.system(f"sudo gdb -p {self.attached\_process.pid} -batch -ex detach")  # Отключение через gdb

                self.append\_output(f"Отключение от процесса {self.attached\_process.pid}")

                self.attached\_process = None

            except Exception as e:

                self.append\_output(f"Ошибка при отключении: {str(e)}")

        else:

            self.append\_output("Нет присоединенного процесса для отключения.")

    def show\_help(self):

        """Показ справки по командам"""

        help\_text = """

Доступные команды:

  meminfo               - Показать информацию о памяти

  readmem <address>     - Чтение данных по указанному адресу

  writemem <address> <value> - Запись значения по указанному адресу

  processlist           - Показать список процессов

  killprocess <pid>     - Завершить процесс по ID

  memusage              - Показать использование памяти текущим приложением

  findprocess <name>    - Найти процесс по имени

  attach <pid>          - Присоединиться к процессу

  detach                - Отключиться от процесса

  help                  - Показать эту справку

"""

        self.append\_output(help\_text)

файл приложения

import os

import psutil

from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QLineEdit, QTextEdit, QLabel

class MemoryApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.initUI()

    def initUI(self):

        layout = QVBoxLayout()

        self.address\_input = QLineEdit(self)

        self.address\_input.setPlaceholderText("Введите адрес памяти (hex)")

        layout.addWidget(QLabel("Адрес памяти:"))

        layout.addWidget(self.address\_input)

        self.value\_input = QLineEdit(self)

        self.value\_input.setPlaceholderText("Введите значение для записи")

        layout.addWidget(QLabel("Значение:"))

        layout.addWidget(self.value\_input)

        self.pid\_input = QLineEdit(self)

        self.pid\_input.setPlaceholderText("Введите PID процесса")

        layout.addWidget(QLabel("PID процесса:"))

        layout.addWidget(self.pid\_input)

        self.output = QTextEdit(self)

        self.output.setReadOnly(True)

        layout.addWidget(QLabel("Вывод:"))

        layout.addWidget(self.output)

        read\_btn = QPushButton("Читать", self)

        read\_btn.clicked.connect(self.read\_memory)

        layout.addWidget(read\_btn)

        write\_btn = QPushButton("Записать", self)

        write\_btn.clicked.connect(self.write\_memory)

        layout.addWidget(write\_btn)

        process\_info\_btn = QPushButton("Информация о процессах", self)

        process\_info\_btn.clicked.connect(self.show\_process\_info)

        layout.addWidget(process\_info\_btn)

        self.setLayout(layout)

    def read\_memory(self):

        address = self.address\_input.text()

        pid = self.pid\_input.text()

        if not address or not pid:

            self.output.append("Ошибка: адрес или PID не введены.")

            return

        try:

            addr = int(address, 16)

            pid = int(pid)

            mem\_file = f"/proc/{pid}/mem"

            with open(mem\_file, 'rb') as mem:

                mem.seek(addr)

                value = mem.read(4)  # Чтение 4 байт

                value\_hex = value.hex()

                self.output.append(f"Чтение данных по адресу {address} (PID {pid}): значение {value\_hex}")

        except Exception as e:

            self.output.append(f"Ошибка при чтении данных: {str(e)}")

    def write\_memory(self):

        address = self.address\_input.text()

        value = self.value\_input.text()

        pid = self.pid\_input.text()

        if not address or not value or not pid:

            self.output.append("Ошибка: адрес, значение или PID не введены.")

            return

        try:

            addr = int(address, 16)

            pid = int(pid)

            value\_bytes = bytes.fromhex(value)

            mem\_file = f"/proc/{pid}/mem"

            with open(mem\_file, 'r+b') as mem:

                mem.seek(addr)

                mem.write(value\_bytes)

                self.output.append(f"Запись значения {value} по адресу {address} (PID {pid}) выполнена успешно.")

        except Exception as e:

            self.output.append(f"Ошибка при записи данных: {str(e)}")

    def show\_process\_info(self):

        try:

            processes = psutil.process\_iter(['pid', 'name', 'memory\_info'])

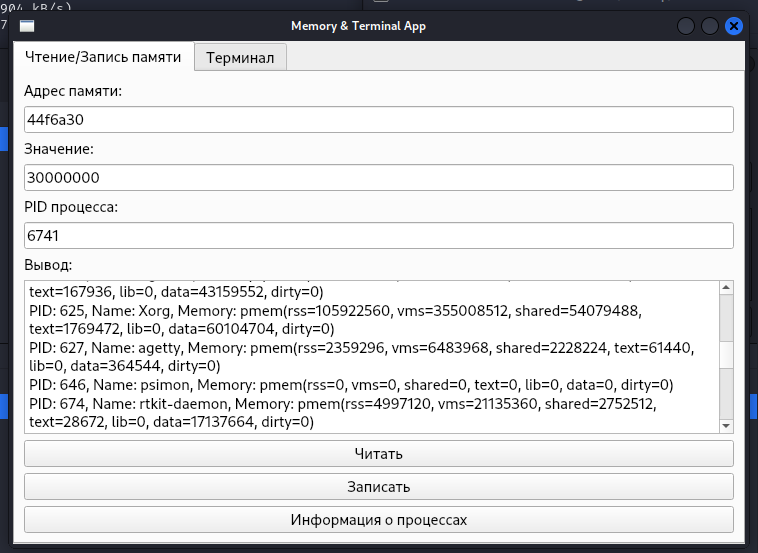
            for proc in processes:

                self.output.append(f"PID: {proc.info['pid']}, Name: {proc.info['name']}, Memory: {proc.info['memory\_info']}")

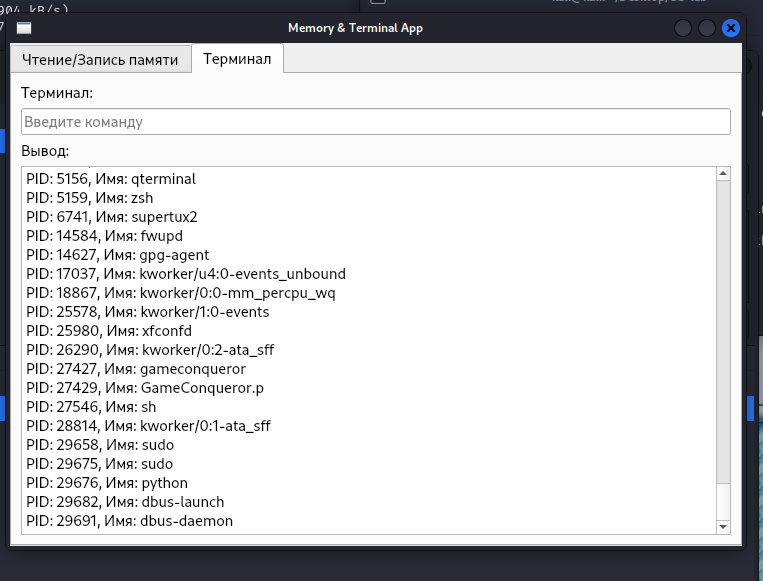
        except Exception as e:

            self.output.append(f"Ошибка при получении информации о процессах: {str(e)}")

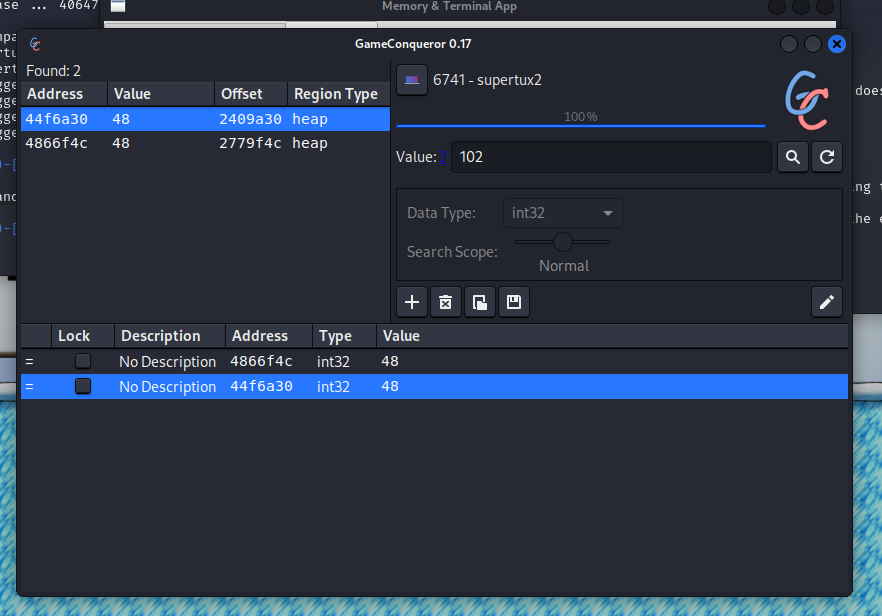
окно приложения

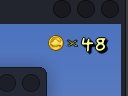


Окно терминала



Поиск ячейки адреса монеток в процессе игры





Запись нового числа в 16 ричном виде

