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 № 1

по дисциплине «Операционные системы и оболочки»

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

Самара, 2024

Задание

Все варианты должны иметь графический интерфейс. Для каждого из

вариантов необходимо реализовать возможность задавать приоритет

каждого процесса и/или порожденных потоков, вручную пользователем.

Процессы и потоки должны быть созданы явно.

Дополнительно позволить запуск из приложения собственного

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

процессами и потоками ОС, дать им пояснение. Например, команда

терминала Linux – find, которая осуществляет поиск в файловой системе,

файлах и папках. Cоздать аналог команды с именем find. Реализовать все

команды на языке программирования.

Вариант 7. Изменение заряда батареи телефона. Телефон имеет

уровень заряда 100%. Один процесс содержит два потока, которые

случайным образом уменьшают уровень заряда на 0,1%. Второй процесс

содержит один поток исполнения и случайным образом увеличивает

уровень заряда на 0,1%.

Main.py

import tkinter as tk

from tkinter import ttk

from battery import charge\_battery\_process, discharge\_battery\_process

from command\_line import CommandLineApp

from multiprocessing import Value

from multiprocessing import Process

class BatteryApp:

    def \_\_init\_\_(self, parent):

        self.parent = parent

        self.battery\_level = Value('d', 100.0)

        self.battery\_label = ttk.Label(parent, text=f"Battery level: {self.battery\_level.value:.1f}%")

        self.battery\_label.pack(pady=10)

        self.start\_discharge\_button = ttk.Button(parent, text="Start Discharging", command=self.start\_discharge\_process)

        self.start\_discharge\_button.pack(pady=5)

        self.start\_charge\_button = ttk.Button(parent, text="Start Charging", command=self.start\_charge\_process)

        self.start\_charge\_button.pack(pady=5)

        self.priority\_label = ttk.Label(parent, text="Set Process Priority (-20 to 19):")

        self.priority\_label.pack(pady=5)

        self.priority\_entry = ttk.Entry(parent)

        self.priority\_entry.pack(pady=5)

        self.priority\_entry.insert(0, "0")

        self.update\_battery\_label()

    def update\_battery\_label(self):

        self.battery\_label.config(text=f"Battery level: {self.battery\_level.value:.1f}%")

        self.parent.after(1000, self.update\_battery\_label)

    def start\_discharge\_process(self):

        priority = int(self.priority\_entry.get())

        discharge\_process = Process(target=discharge\_battery\_process, args=(self.battery\_level, priority))

        discharge\_process.start()

    def start\_charge\_process(self):

        priority = int(self.priority\_entry.get())

        charge\_process = Process(target=charge\_battery\_process, args=(self.battery\_level, priority))

        charge\_process.start()

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

    root = tk.Tk()

    root.title("Battery Simulator")

    notebook = ttk.Notebook(root)

    battery\_frame = ttk.Frame(notebook)

    command\_frame = ttk.Frame(notebook)

    notebook.add(battery\_frame, text="Battery Simulator")

    notebook.add(command\_frame, text="Command Line")

    notebook.pack(expand=True, fill="both")

    app = BatteryApp(battery\_frame)

    cli = CommandLineApp(command\_frame)

    root.mainloop()

command\_line.py

import os

import psutil

import tkinter as tk

from tkinter import ttk

from threading import Thread, Event

from multiprocessing import Process

import glob

processes = {}

threads = {}

thread\_stop\_flags = {}

class CommandLineApp:

    def \_\_init\_\_(self, parent):

        self.parent = parent

        self.command\_label = ttk.Label(parent, text="Enter command:")

        self.command\_label.pack(pady=5)

        self.command\_entry = ttk.Entry(parent)

        self.command\_entry.pack(pady=5)

        self.execute\_button = ttk.Button(parent, text="Execute", command=self.execute\_command)

        self.execute\_button.pack(pady=5)

        self.output\_text = tk.Text(parent, height=10, width=50)

        self.output\_text.pack(pady=5)

    def execute\_command(self):

        command = self.command\_entry.get()

        output = self.run\_command(command)

        self.output\_text.insert(tk.END, output + "\n")

    def run\_command(self, command):

        global processes, threads, thread\_stop\_flags

        command\_parts = command.split()

        cmd = command\_parts[0]

        if cmd == "ps":

            return self.ps()

        elif cmd == "kill":

            return self.kill(command\_parts)

        elif cmd == "top":

            return self.top()

        elif cmd == "thread-start":

            return self.thread\_start(command\_parts)

        elif cmd == "thread-stop":

            return self.thread\_stop(command\_parts)

        elif cmd == "proc-start":

            return self.proc\_start(command\_parts)

        elif cmd == "proc-stop":

            return self.proc\_stop(command\_parts)

        elif cmd == "proc-status":

            return self.proc\_status(command\_parts)

        elif cmd == "proc-priority":

            return self.proc\_priority(command\_parts)

        elif cmd == "find":

            return self.find(command\_parts)

        else:

            return f"Unknown command: {cmd}"

    def ps(self):

        return "\n".join([f"PID: {p.pid}, Name: {p.name()}" for p in psutil.process\_iter()])

    def kill(self, args):

        if len(args) < 2:

            return "Usage: kill <pid>"

        pid = int(args[1])

        try:

            psutil.Process(pid).terminate()

            return f"Process {pid} terminated."

        except psutil.NoSuchProcess:

            return f"No such process with PID {pid}."

    def top(self):

        processes = [(p.pid, p.name(), p.cpu\_percent(), p.memory\_percent()) for p in psutil.process\_iter()]

        processes.sort(key=lambda x: x[2], reverse=True)

        output = "PID    Name    CPU%    Mem%\n"

        for p in processes[:10]:

            output += f"{p[0]}    {p[1]}    {p[2]:.1f}    {p[3]:.1f}\n"

        return output

    def thread\_start(self, args):

        if len(args) < 2:

            return "Usage: thread-start <name>"

        name = args[1]

        stop\_flag = Event()

        thread = Thread(target=self.dummy\_thread, args=(name, stop\_flag))

        thread.start()

        threads[name] = thread

        thread\_stop\_flags[name] = stop\_flag

        return f"Thread {name} started."

    def thread\_stop(self, args):

        if len(args) < 2:

            return "Usage: thread-stop <name>"

        name = args[1]

        if name in threads:

            thread\_stop\_flags[name].set()

            threads[name].join()

            del threads[name]

            del thread\_stop\_flags[name]

            return f"Thread {name} stopped."

        return f"No thread with name {name}."

    def dummy\_thread(self, name, stop\_flag):

        while not stop\_flag.is\_set():

            print(f"Thread {name} is running...")

            time.sleep(1)

    def proc\_start(self, args):

        if len(args) < 2:

            return "Usage: proc-start <name>"

        name = args[1]

        process = Process(target=self.dummy\_process, args=(name,))

        process.start()

        processes[process.pid] = process

        return f"Process {name} started with PID {process.pid}."

    def proc\_stop(self, args):

        if len(args) < 2:

            return "Usage: proc-stop <pid>"

        pid = int(args[1])

        if pid in processes:

            processes[pid].terminate()

            return f"Process {pid} stopped."

        return f"No process with PID {pid}."

    def dummy\_process(self, name):

        while True:

            print(f"Process {name} is running...")

            time.sleep(1)

    def proc\_status(self, args):

        if len(args) < 2:

            return "Usage: proc-status <pid>"

        pid = int(args[1])

        if pid in processes:

            return f"Process {pid} is running."

        return f"No process with PID {pid}."

    def proc\_priority(self, args):

        if len(args) < 3:

            return "Usage: proc-priority <pid> <priority>"

        pid = int(args[1])

        priority = int(args[2])

        if pid in processes:

            os.nice(priority)

            return f"Priority of process {pid} set to {priority}."

        return f"No process with PID {pid}."

    def find(self, args):

        if len(args) < 2:

            return "Usage: find <file\_name>"

        file\_name = args[1]

        files = glob.glob(f"\*\*/{file\_name}", recursive=True)

        return "\n".join(files) if files else f"No files found with name {file\_name}."

battery.py  
import os

import time

import random

from multiprocessing import Process, Value

from threading import Thread, Event

def discharge\_battery\_thread(battery\_level, stop\_flag):

    while not stop\_flag.is\_set() and battery\_level.value > 0:

        time.sleep(random.uniform(0.1, 0.5))

        with battery\_level.get\_lock():

            battery\_level.value = max(0, battery\_level.value - 0.1)

        print(f"Discharging... Battery level: {battery\_level.value:.1f}%")

def discharge\_battery\_process(battery\_level, priority):

    os.nice(priority)  # Установка приоритета процесса

    stop\_flag = Event()

    thread1 = Thread(target=discharge\_battery\_thread, args=(battery\_level, stop\_flag))

    thread2 = Thread(target=discharge\_battery\_thread, args=(battery\_level, stop\_flag))

    thread1.start()

    thread2.start()

    thread1.join()

    thread2.join()

def charge\_battery\_thread(battery\_level, stop\_flag):

    while not stop\_flag.is\_set() and battery\_level.value < 100:

        time.sleep(random.uniform(0.1, 0.5))

        with battery\_level.get\_lock():

            battery\_level.value = min(100, battery\_level.value + 0.1)

        print(f"Charging... Battery level: {battery\_level.value:.1f}%")

def charge\_battery\_process(battery\_level, priority):

    os.nice(priority)  # Установка приоритета процесса

    stop\_flag = Event()

    thread = Thread(target=charge\_battery\_thread, args=(battery\_level, stop\_flag))

    thread.start()

    thread.join()

process\_utils.py  
import time

from threading import Thread, Event

from multiprocessing import Process

def start\_thread(name, stop\_flag, target\_func):

    thread = Thread(target=target\_func, args=(name, stop\_flag))

    thread.start()

    return thread

def stop\_thread(name, threads, thread\_stop\_flags):

    if name in threads:

        thread\_stop\_flags[name].set()

        threads[name].join()

        del threads[name]

        del thread\_stop\_flags[name]

def start\_process(name, target\_func):

    process = Process(target=target\_func, args=(name,))

    process.start()

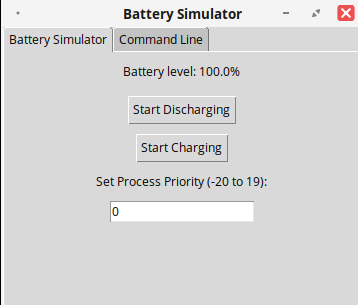
    return process

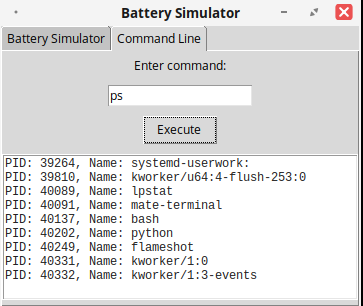
def stop\_process(pid, processes):

    if pid in processes:

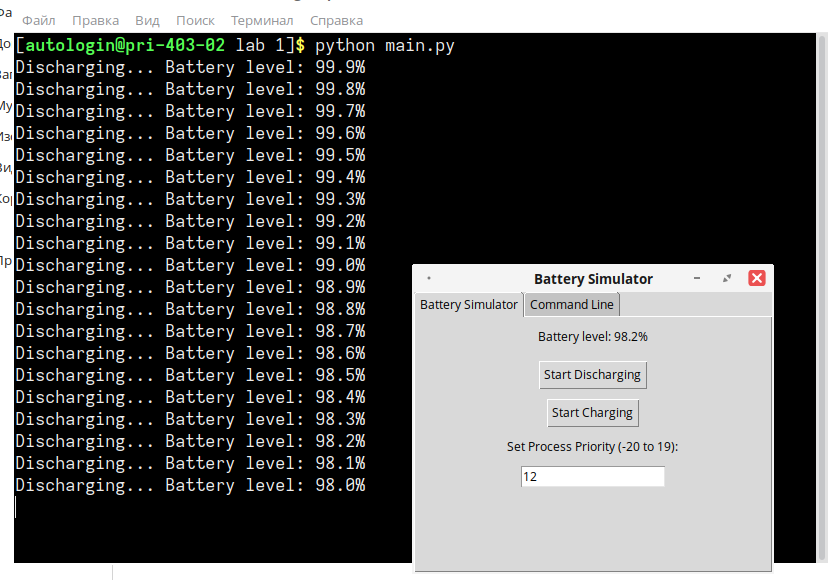
        processes[pid].terminate()

        del processes[pid]

окно программы  
  
окно терминала



Запуск разрядки



Запуск зарядки

