PROJEKTNA NALOGA

Poročilo

Operacijski sistemi

Adrian Cvetko

RIT – UN, RV2

2024

Kazalo vsebine

Uvod

Za temo svoje projektne naloge sem se odločil, da bom delal “system monitor” v ukazni vrstici oz. neko vrsto nadgradnje orodja “top”.
Vizualno je moja implementacija dokaj podobna, ampak vključuje tudi barve ter stilizirano pisavo (**krepko**, *italic…*).
Za programski jezik sem se odločil, da bom uporabljal python. S svojo široko zbirko raznih knjižnic sem imel veliko izbiro med orodji, s katerimi sem si lahko pomagal za oblikovanje izgleda orodja ter tudi pridobitev sistemskih podatkov, ki sem jih potreboval. Najbolj sem se zanašal na knjižnico “curses” in “psutil”. Uporabil sem tudi nitenje, da določeni deli aplikacije delujejo neodvisno od drugih.

Funkcije, ki sem jih implementiral:

- uporabniku prijaznejši vmesnik
- interaktivno uporabo
- ustavljanje procesov po imenu ukaza ali poti
- izpis odprtih datotek/oprimkov procesa (število le teh)
- pregled nad zasedenim pomnilnikom procesa
- pregled nad odprtimi mrežnimi kanali komunikacije (Unix vtiči, TCP-UDP/IP vtiči)
- izpis procesov s podano delovno potjo
- izpis procesov za podanega uporabnika
- sortiranje procesov glede na procent uporabe procesorja/pomnilnika
- nekaj dodatkov nad splošnim pregledom sistema v primerjavi z orodjem “top”
- filtriranje procesov glede na uporabnika

Implementacija pa na žalost ni popolna, saj mi je zmanjkalo časa, da bi jo optimiziral, zato deluje počasneje v primerjavi z orodjem “top”. Osveževanje sicer deluje glede na enak interval, ampak implementacija ni tako instantno odzivna kot recimo orodje “htop”.

V splošnem pregledu sistema se pojavi tudi temperatura grafične kartice in hitrost njenih ventilatorjev. Trenutno ta implementacija podpira samo Nvidia grafične kartice, ker je ta prisotna v mojem računalniku in ker mi je zmanjkalo časa, da bi implementiral sistem, ki bi prepoznal izdelovalca grafične kartice v sistemu.

Za pridobitev informacij o sistemu, ki so procesi, zasedenost pomnilnika, poraba procesorja itd. sem uporabljal knjižnico “psutil”. To sem si privoščil zato, ker sem implementacijo delal v Pythonu. Četudi /proc omogoča maksimalno učinkovitost in minimalne odvisnosti programa od zunanjih knjižnic, zahteva dosti bolj napredno ročno obdelavo podatkov, medtem ko knjižnica “psutil” prinaša večji nivo abstrakcije v zameno za malo slabšo učinkovitost.

Pomožne funkcije

V programski kodi je opisan sklop funkcij, ki omogočajo spremljanje ključnih informacij o stanju sistema, kot so čas delovanja, temperatura procesorja in grafične kartice, status baterije, obremenitev procesorja, pomnilnika ter upravljanje procesov.

```
def format_uptime():
    uptime_seconds = time.time() - psutil.boot_time()
    uptime_struct = time.gmtime(uptime_seconds)
    return f"{uptime_struct.tm_yday - 1} days, {uptime_struct.tm_hour:02}:{uptime_struct.tm_min:02}"
```

Funkcija `format_uptime` izračuna, koliko časa je sistem že aktiven od zadnjega zagona. Najprej izračuna razliko med trenutnim časom in časom zagona sistema, ki ga pridobi s knjižnico `psutil`. To razliko nato pretvori v strukturirani časovni format z uporabo funkcije `time.gmtime`. Na podlagi tega izračuna število dni, ur in minut, kar vrne kot niz, na primer "1 days, 12:34".

```
def get_cpu_temperature():
    try:
        temps = psutil.sensors_temperatures()
        if "coretemp" in temps:
            return f"{temps['coretemp'][0].current:.1f}°C"
    except (AttributeError, KeyError):
        pass
    return "N/A"
```

Funkcija `get_cpu_temperature` je namenjena pridobivanju trenutne temperature procesorja. Za to uporablja knjižnico `psutil`, natančneje metodo `sensors_temperatures`, ki vrne podatke o vseh zaznanih temperaturnih senzorjih. Če med temi podatki obstaja ključ `coretemp`, kar je običajno ime za senzorje procesorja, funkcija vrne temperaturo prvega jedra v obliki niza, na primer "50.5°C". V primeru, da podatkov ni mogoče pridobiti, funkcija vrne "N/A". Ta funkcionalnost je uporabna pri spremljanju toplotnega stanja sistema in preprečevanju pregrevanja.

```python
def get_gpu_temperature_and_fan():
    try:
        pynvml.nvmlInit()
        handle = pynvml.nvmlDeviceGetHandleByIndex(0)
        gpu_temp = pynvml.nvmlDeviceGetTemperature(handle, pynvml.NVML_TEMPERATURE_GPU)
        fan_speed = pynvml.nvmlDeviceGetFanSpeed(handle)
        pynvml.nvmlShutdown()
        return f"{gpu_temp}°C, Fan: {fan_speed}%"
    except pynvml.NVMLError:
        return "N/A"
```

S funkcijo `get_gpu_temperature_and_fan` je mogoče pridobiti temperaturo grafične kartice in hitrost njenega ventilatorja. Ta funkcija uporablja knjižnico `pynvml`, ki omogoča komunikacijo z NVIDIA grafičnimi karticami. Funkcija inicializira NVML, pridobi ročaj za prvo napravo in nato prebere temperaturo ter hitrost ventilatorja. Če je vse uspešno, vrne podatke, kot so "70°C, Fan: 40%". Če pride do napake, na primer če GPU ni podprt, vrne "N/A". Spremljanje teh informacij je potencialno uporabno za spremljanje grafično intenzivnih nalog, kot so igre in strojno učenje.

```python
def get_battery_status():
    battery = psutil.sensors_battery()
    if battery:
        percent = battery.percent
        charging = "Charging" if battery.power_plugged else "Discharging"
        return f"{percent:.1f}% ({charging})"
    else:
        return "N/A"
```

Funkcija `get_battery_status` omogoča vpogled v trenutno stanje baterije sistema. Z uporabo metode `psutil.sensors_battery` pridobi podatke o odstotku napolnjenosti baterije in preveri, ali se baterija polni. Vrne podatke v obliki niza, kot so "85.0% (Charging)" ali "42.3% (Discharging)". Če sistem ne uporablja baterije (na primer namizni računalnik), funkcija vrne "N/A". Ta funkcionalnost je uporabna samo za prenosne računalnike in mobilne naprave.

```python
def get_top_header(stdscr):
    with threadLock:
        height, width = stdscr.getmaxyx()

    current_time = datetime.now().strftime("%H:%M:%S")
    uptime = format_uptime()
    users = len(psutil.users())
    load_avg = os.getloadavg()
    battery_status = get_battery_status()

    all_procs = list(psutil.process_iter(['status']))
    total_tasks = len(all_procs)
    running_tasks = sum(1 for p in all_procs if p.info['status'] == psutil.STATUS_RUNNING)
    sleeping_tasks = sum(1 for p in all_procs if p.info['status'] == psutil.STATUS_SLEEPING)
    stopped_tasks = sum(1 for p in all_procs if p.info['status'] == psutil.STATUS_STOPPED)
    zombie_tasks = sum(1 for p in all_procs if p.info['status'] == psutil.STATUS_ZOMBIE)

    cpu_times = psutil.cpu_times_percent(interval=1)
    cpu_summary = (f"{cpu_times.user:.1f} us, {cpu_times.system:.1f} sy, {cpu_times.nice:.1f} ni, "
                   f"{cpu_times.idle:.1f} id, {cpu_times.iowait:.1f} wa, {cpu_times.irq:.1f} hi, "
                   f"{cpu_times.softirq:.1f} si, {cpu_times.steal:.1f} st")

    mem = psutil.virtual_memory()
    swap = psutil.swap_memory()
```

```python
    mem = psutil.virtual_memory()
    swap = psutil.swap_memory()
    mem_info = (f"MiB Mem : {mem.total / 1024**2:.1f} total, {mem.available / 1024**2:.1f} free, "
                f"{mem.used / 1024**2:.1f} used, {mem.buffers / 1024**2:.1f} buff/cache")
    swap_info = (f"MiB Swap: {swap.total / 1024**2:.1f} total, {swap.free / 1024**2:.1f} free, "
                 f"{swap.used / 1024**2:.1f} used")

    cpu_temp = get_cpu_temperature()
    gpu_info = get_gpu_temperature_and_fan()

    header = [f"top(Adrian verzija) - {current_time} up {uptime}, {users} users, load average: {load_avg[0]:.2f}, {load_avg[1]:.2f}
              f"Tasks: {total_tasks} total, {running_tasks} running, {sleeping_tasks} sleeping, {stopped_tasks} stopped, {zombie_ta
              f"%CPU(s): {cpu_summary}",
              f"CPU Temp: {cpu_temp} | GPU Temp & Fan: {gpu_info} | Battery: {battery_status}",
              f"{mem_info}\n{swap_info}"]

    header2 = ""
    for x in header:
        if len(x) > width:
            x = x[4:width]

        header2 += x + "\n"

    return header2
```

Funkcija `get_top_header` združuje vse prej omenjene funkcionalnosti v enoten prikaz za terminalno aplikacijo. Vsebuje več korakov:

1. Pridobi dimenzije terminala, trenutni čas, čas delovanja, število uporabnikov in obremenitev sistema.
2. Prešteje vse procese in razvrsti njihove statuse (tečejo, spijo, zombiji itd.).
3. Zajame podatke o uporabi procesorja, pomnilnika in swap prostora.
4. Preveri temperature CPU in GPU ter hitrost ventilatorja.
5. Sestavi vse te podatke v večvrstični niz, ki ga prilagodi širini terminala.

Rezultat je natančen pregled stanja sistema, ki je hkrati informativen in pregleden.

```python
def kill_process_by_name(name_or_path):
    with threadLock:
        for proc in psutil.process_iter(['name']):
            try:
                process_name = proc.info['name']
                process_path = proc.exe()

                if name_or_path.lower() in process_name.lower() or name_or_path.lower() in process_path.lower():
                    proc.terminate()
            except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
                pass
```

Funkcija `kill_process_by_name` omogoča, da uporabnik s podanim imenom ali potjo zaključi določene procese. S pomočjo metode `psutil.process_iter` iterira prek vseh procesov in preveri njihovo ime ter pot. Če ime ali pot ustreza iskanemu nizu, proces zaključi z metodo `proc.terminate`. Funkcija je robustna in ignorira morebitne napake, kot so manjkajoči procesi ali omejitve dostopa. Ta funkcionalnost omogoča lažje upravljanje sistemskih virov in odstranjevanje nezaželenih ali blokiranih procesov.

Glavno delovanje programa

Glavno delovanje se izvaja v dveh večjih funkcijah, in sicer “footer” in “menu”. Kot je možno sklepati iz imen, prva funkcija skrbi za spodnjo “nogo” programa, “menu” pa za prikaz vseh informacij, ki jih je možno pridobiti iz zgornjih funkcij. Obe tečeta vsaka v svoji niti:

```python
def main(stdscr):
    thread_footer = threading.Thread(target=footer, args=(stdscr,))
    thread_menu = threading.Thread(target=menu, args=(stdscr,))

    thread_footer.start()
    thread_menu.start()

    thread_footer.join()
    thread_menu.join()

if __name__ == "__main__":
    curses.wrapper(main)
```

Funkcija `footer` skrbi za dinamično upravljanje in prikaz spodnjega dela uporabniškega vmesnika v terminalni aplikaciji, ki deluje z uporabo knjižnice `curses`. Gre za interaktiven element, ki omogoča uporabniku izvajanje različnih dejanj, kot so sortiranje procesov, filtriranje, preklapljanje med različnimi načini prikaza in zaključevanje aplikacije. Funkcija vključuje več delov, od inicializacije spremenljivk in nastavitev, do obdelave dogodkov in prikaza informacij.

Na začetku funkcije je poskrbljeno za inicializacijo terminalskega vmesnika z uporabo knjižnice `curses`. Funkcija skrije kurzor, omogoči asinhrono obdelavo uporabniških vnosov in določi časovno omejitev čakanja na vnos. Poleg tega konfigurira barve in barvne pare, ki se uporabljajo za vizualno označevanje določenih elementov v vmesniku.

```
def footer(stdscr):
    global sortValue
    global running
    global filter, filterMode
    global unixSockets, tcpSockets, udpSockets, monitor
    with threadLock:
        curses.curs_set(0)
        stdscr.nodelay(1)
        stdscr.timeout(50)
        curses.init_color(8, 0, 0, 0)
        curses.init_pair(1, 8, curses.COLOR_MAGENTA)
```

Sledi inicializacija ključnih spremenljivk, ki spremljajo stanje aplikacije. Med njimi so `searchMode`, ki določa, ali je uporabnik v načinu iskanja, ter `search_query`, kjer se beleži trenutna iskalna poizvedba. Poleg tega je definirana spremenljivka `killMode`, ki omogoča upravljanje funkcionalnosti za zaključevanje procesov. Na tem mestu je prav tako ustvarjen seznam nizov `footer`, ki vsebuje opise funkcionalnih tipk, kot so sortiranje procesov, filtriranje in izhod iz aplikacije.

```
searchMode = False
search_query = ""
killMode = False


footer = ["F1-sort cpu%", "F2-sort mem%", "F3-filter", "F4-kill",
```

Jedro funkcije predstavlja glavna zanka, ki deluje, dokler aplikacija ostane aktivna. V tej zanki se najprej počisti spodnja vrstica zaslona, da se prepreči prekrivanje besedila. Nato funkcija dinamično prikaže vsebino glede na trenutno stanje aplikacije. Če je uporabnik v načinu iskanja, se v spodnji vrstici prikaže poziv za vnos iskalne poizvedbe, ki je poudarjen z uporabo barvnih parov. V nasprotnem primeru funkcija prikaže gumbe funkcionalnosti, pri čemer vsak gumb prilagodi širini zaslona, da zagotovi pravilno postavitev.

```
while running == True:
    try:
        with threadLock:
            height, width = stdscr.getmaxyx()
            for y in range(height - 1, height):
                stdscr.move(y, 0)
                stdscr.clrtoeol()


        counter = 0

        with threadLock:
            if searchMode == True:
                stdscr.addstr(height - 1, 0, f"Input: {search_query}".ljust(width-1), curses.color_pair(1))
            else:
                for line in footer:
                    if counter + len(line) + 1 > width:
                        break

                    stdscr.addstr(height - 1, counter, line, curses.A_REVERSE)
                    counter += len(line)

                    if counter < width:
                        stdscr.addstr(height - 1, counter, " ")
                        counter += 1


            key = stdscr.getch()
```

Ena ključnih nalog funkcije je obdelava uporabniških vnosov, kjer vsak pritisk tipke sproži določeno dejanje. Tipka F10 zapre aplikacijo, F1 in F2 omogočata sortiranje procesov po uporabi CPU-ja ali pomnilnika, medtem ko F3 in F4 omogočata preklapljanje med načinoma filtriranja in zaključevanja procesov. Dodatne funkcijske tipke, kot so F5 do F8, omogočajo preklapljanje med različnimi prikazi, vključno z Unix vtiči, TCP vtiči, UDP vtiči in splošnim seznamom procesov.

Pomemben del funkcionalnosti predstavlja obravnava iskanja in zaključevanja procesov. V načinu iskanja uporabnik vnese iskalno poizvedbo, ki se sproti posodablja ob vsakem pritisku tipke. Funkcija podpira brisanje znakov in potrjevanje vnosa z vnosom tipke za potrditev (ENTER). Če je med iskanjem aktiven način zaključevanja procesov, funkcija uporabi iskalno poizvedbo za identifikacijo procesov, ki jih je treba zaključiti. V nasprotnem primeru se iskalna poizvedba shrani kot filter za prikaz procesov, kar uporabniku omogoča lažje iskanje specifičnih aplikacij ali storitev.

```
        if key == curses.KEY_F10:
            running = False

        if key == curses.KEY_F2:
            sortValue = 'memory_percent'

        if key == curses.KEY_F1:
            sortValue = 'cpu_percent'

        if key == curses.KEY_F3:
            with threadLock:
                if filterMode == True:
                    filterMode = not filterMode
                    footer[2] = "F3 - filter"
                else:
                    searchMode = not searchMode
                    search_query = ""
                    footer[2] = "F3 - cancel"


        if key == curses.KEY_F4:
            searchMode = not searchMode
            killMode = True
            search_query = ""
```

```
        if key == curses.KEY_F5:
            with threadLock:
                stdscr.clear()
                unixSockets = True
                tcpSockets = False
                udpSockets = False
                monitor = False

        if key == curses.KEY_F6:
            with threadLock:
                stdscr.clear()
                unixSockets = False
                tcpSockets = True
                udpSockets = False
                monitor = False

        if key == curses.KEY_F7:
            with threadLock:
                stdscr.clear()
                unixSockets = False
                tcpSockets = False
                udpSockets = True
                monitor = False
```

```
    if searchMode == True:
        if key == curses.KEY_BACKSPACE:
            search_query = search_query[:-1]
        elif key == 10:  # Enter

            if killMode == True:
                kill_process_by_name(search_query)
                killMode = False
            else:
                with threadLock:
                    filter = search_query
                    filterMode = True


            searchMode = not searchMode
            search_query = ""
        elif 32 <= key <= 126:  # za crke pa stevilke
            search_query += chr(key)

except:
    continue
```

Funkcija `menu` je osrednji del terminalske aplikacije, ki omogoča dinamično izbiranje in prikaz različnih podatkov, vključno z informacijami o procesih, omrežnih povezavah in vtičih. Z uporabo knjižnice `curses` skrbi za vizualno predstavitev podatkov, obdelavo uporabniških vnosov ter prilagajanje vsebine zaslona glede na uporabnikove izbire. Funkcija je sestavljena iz več delov, od inicializacije terminala, določanja stanja aplikacije, do obdelave podatkov in prikaza različnih kategorij informacij.

Na začetku funkcije se izvajajo pripravljalne operacije za uporabo knjižnice `curses`. V tem delu se skrije kurzor in omogoči asinhrono obdelavo uporabniških vnosov, kar zagotavlja, da aplikacija ostaja odzivna tudi brez neposrednega vnosa. Funkcija prav tako inicializira barvne pare, ki se uporabljajo za poudarjanje določenih elementov na zaslonu. Ta uvodni del definira tudi osnovne parametre zaslona, kot sta višina in širina terminalskega okna, kar omogoča prilagodljiv prikaz podatkov glede na velikost okna.

```
def menu(stdscr):
    global sortValue
    global filter, filterMode
    global unixSockets, tcpSockets, udpSockets, monitor
    caseNum = 0
    with threadLock:
        curses.curs_set(0)
        stdscr.nodelay(1)
        #stdscr.timeout(1000)
        curses.start_color()
        curses.init_color(8, 0, 0, 0)
        curses.init_pair(1, 8, curses.COLOR_MAGENTA)
        curses.init_pair(2, curses.COLOR_BLACK, curses.COLOR_GREEN)
        height, width = stdscr.getmaxyx()


    while running == True:

        with threadLock:
            if unixSockets == True:
                caseNum = 1
            elif tcpSockets == True:
                caseNum = 2
            elif udpSockets == True:
                caseNum = 3
            elif monitor == True:
                caseNum = 0
```

Osrednji del funkcije je glavna zanka, ki se izvaja, dokler aplikacija deluje. Znotraj te zanke funkcija preverja globalne spremenljivke, ki določajo trenutno stanje aplikacije, in na podlagi njih izbere ustrezen prikaz podatkov. Ti pogoji vključujejo različne načine, kot so prikaz procesov, Unix vtičev, TCP povezav in UDP povezav. Funkcija uporablja ukaz `match` za preklapljanje med različnimi načini prikaza, pri čemer vsak primer obdela specifično vrsto informacij.

Prvi primer v strukturi `match` je namenjen prikazu procesov. V tem načinu funkcija pridobi podatke o aktivnih procesih z uporabo knjižnice `psutil`. Podatki vključujejo ključne informacije, kot so ID procesa, uporabniško ime, poraba CPU-ja in pomnilnika, prioriteta ter ime procesa. Funkcija nato te podatke uredi po izbranem kriteriju, kot sta poraba CPU-ja ali pomnilnika, kar omogoča uporabniku pregled procesov po pomembnosti.

```
    with threadLock:
        num_cores = psutil.cpu_count()

    start_line = 7
    max_processes = height - start_line - 1

    with threadLock:
        for y in range(height, height-2):
            stdscr.move(y, 0)
            stdscr.clrtoeol()

    header = "    PID   USER      PR  NI     RES    CPU%   MEM%   FD   NAME                    PATH"
    processes = []
    for proc in psutil.process_iter(['pid', 'username', 'cpu_percent',
                                     'memory_percent', 'name', 'nice',
                                     'memory_info', 'cpu_times']):
        proc.cpu_percent()

    for proc in psutil.process_iter(['pid', 'username', 'cpu_percent',
                                     'memory_percent', 'name', 'nice',
                                     'memory_info', 'cpu_times']):
        try:
            pr_value = 20 + proc.info['nice']
            processes.append({
                'pid': proc.info['pid'],  #pid procesa
                'pr': pr_value,  #prioriteta procesa
                'ni': proc.info['nice'],  #nice value
                'username': proc.info['username'],  #ime uporabnika
                'res': proc.info['memory_info'].rss // 1000**2,  #fizicana poraba glavnega pomnilnika
                'cpu_percent': proc.info['cpu_percent'] / num_cores,  #uporaba procesorja
                'memory_percent': proc.info['memory_percent'],  #zasedenost v pomnilniku
                'time': proc.info['cpu_times'].user + proc.info['cpu_times'].system,  #cpu time
                'name': proc.info['name'],  #ime procesa
                'fd_count': proc.num_fds(),  # file descriptors count
                'path': proc.exe()  # path
            })
        except (psutil.NoSuchProcess, psutil.AccessDenied):
            continue
```

Poleg tega funkcija podpira filtriranje procesov glede na uporabniško ime, kar omogoča osredotočenje na določene procese. Podatki so nato oblikovani v vrstico, ki vključuje omejitve dolžine za nekatere elemente, kot je ime procesa, da se zagotovi pravilen prikaz na terminalu. Vrstice so izrisane na zaslonu z uporabo različnih barv in slogov, ki poudarjajo specifične lastnosti, na primer procese, ki jih izvaja uporabnik `root`

```
        stdscr.addstr(6, 0, header.ljust(width), curses.color_pair(1) | curses.A_BOLD)
        counter = 0

        if filterMode == True:
            for i in range(start_line, height-1):
                stdscr.addstr(i, 0, ' ' * width)

        for idx, proc in enumerate(processes[:max_processes]):
            try:
                if filterMode:
                    if proc['username'] == filter:

                        if start_line + idx >= height - 1:
                            break

                        max_name_length = 18
                        truncated_name = proc['name'][:max_name_length] + ('...' if len(proc['name']) > max_name_length else '')

                        line = f"{proc['pid']:7}   {proc['username']:8} {proc['pr']:3} {proc['ni']:3} {proc['res']:6}M  {proc['cpu_percent'

                        if len(line) > width:
                            line = line[:width-3] + '...'

                        if counter == 0:
                            stdscr.addstr(start_line + counter, 0, line.ljust(width), curses.color_pair(2) | curses.A_ITALIC)
                        else:
                            if proc['username'] == "root":
                                stdscr.addstr(start_line + counter, 0, line.ljust(width), curses.A_BOLD | curses.A_ITALIC)
                            else:
                                stdscr.addstr(start_line + counter, 0, line.ljust(width))

                        counter += 1
                    else:
                        max_processes += 1
                else:
                    if start_line + idx >= height - 1:
                        break
```

Drugi primer se osredotoča na prikaz Unix vtičev. Funkcija pridobi seznam povezav vrste Unix z uporabo funkcije `psutil.net_connections` in za vsako povezavo pridobi informacije, kot so ID procesa, datotečni deskriptor in pot. Te informacije so nato oblikovane v vrstico in prikazane na zaslonu, pri čemer funkcija skrbi za prilagoditev dolžine vrstic glede na širino zaslona. Ta način omogoča uporabniku vpogled v specifične omrežne povezave, ki so povezane z Unix vtiči.

```
case 1:
    try:
        start_line = 1
        max_connections = height - start_line - 1

        with threadLock:
            for y in range(height, height-1):
                stdscr.move(y, 0)
                stdscr.clrtoeol()

        header = f"    PID    FD    PATH"
        stdscr.addstr(0, 0, header.ljust(width), curses.color_pair(1) | curses.A_BOLD)
        unix_sockets = psutil.net_connections(kind='unix')
        unix_info = [
            {"fd": conn.fd,
             "path": conn.laddr if conn.laddr else "N/A",
             "raddr": conn.raddr if conn.raddr else "N/A",
             "family": conn.family,
             "pid": conn.pid,

             } for conn in unix_sockets if conn.laddr
        ]
        with threadLock:
            for idx, proc in enumerate(unix_info[:max_connections]):
                try:
                        if start_line + idx >= height - 1:
                            break

                        line = f"{proc['pid']:7}    {proc['fd']:3}    {proc['path']:3}"

                        if len(line) > width:
                            line = line[:width-3] + '...'

                        stdscr.addstr(start_line + idx, 0, line.ljust(width))
                except (psutil.NoSuchProcess, KeyError):
                    continue
```

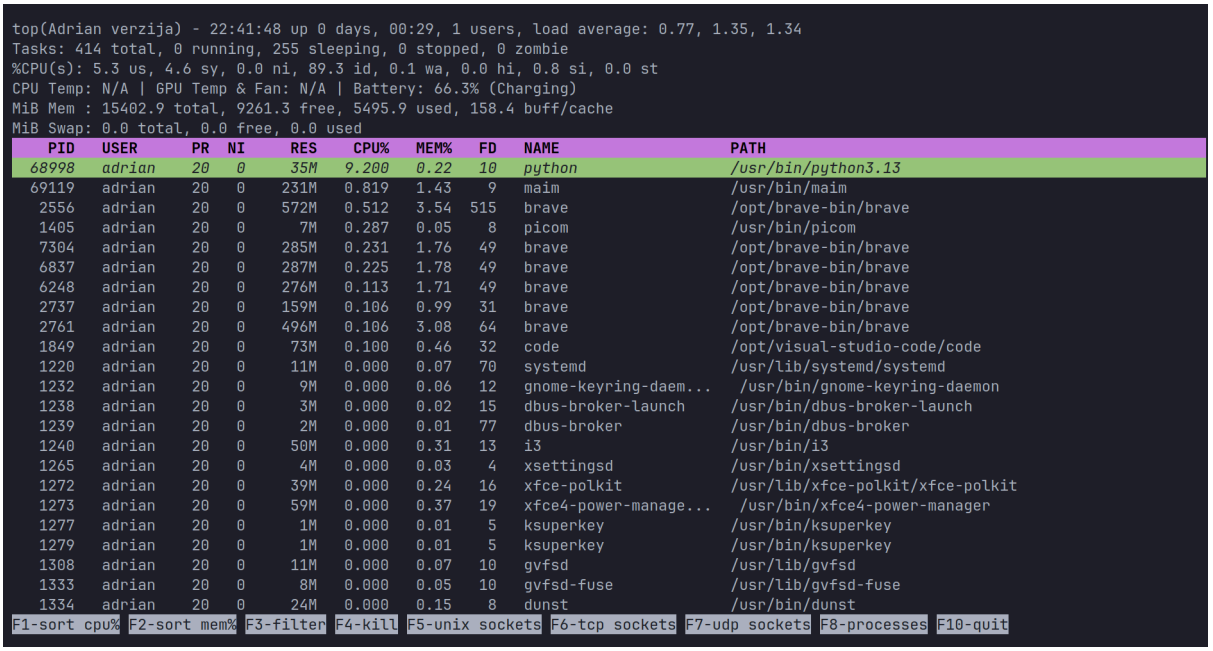
Tretji primer je namenjen prikazu TCP povezav. V tem načinu funkcija pridobi seznam vseh TCP povezav, vključno z informacijami, kot so lokalni in oddaljeni naslovi, stanje povezave ter ID procesa. Te podatke funkcija oblikuje v strukturirane vrstice, ki jih nato izriše na zaslon. Prikaz vključuje tudi prilagoditve dolžine, da se zagotovi pravilna poravnava in preglednost podatkov na terminalu. Na enak način četrti primer obravnava UDP povezave, pri čemer funkcija pridobi in prikaže ustrezne podatke za ta tip povezav.

```
case 2:
    try:
        start_line = 1
        max_connections = height - start_line - 1
        with threadLock:
            for y in range(height, height-1):
                stdscr.move(y, 0)
                stdscr.clrtoeol()

        header = f"    PID    TYPE    LADDR                          RADDR                    STATUS"
        stdscr.addstr(0, 0, header.ljust(width), curses.color_pair(1) | curses.A_BOLD)
        tcp_sockets = psutil.net_connections(kind='tcp')
        tcp_info = [
            {"type": "TCP",
             "laddr": conn.laddr,
             "raddr": conn.raddr,
             "status": conn.status,
             "pid": conn.pid,
             } for conn in tcp_sockets
        ]
        with threadLock:
            for idx, proc in enumerate(tcp_info[:max_connections]):
                try:
                        if start_line + idx >= height - 1:
                            break

                        line = (f"{proc['pid']:7}    {proc['type']:5}     "
                                f"{':'.join(map(str, proc['laddr'])) if proc['laddr'] != 'N/A' else 'N/A':23}      "
                                f"{':'.join(map(str, proc['raddr'])) if proc['raddr'] != 'N/A' else 'N/A':20}      "
                                f"{proc['status']:10}")

                        if len(line) > width:
                            line = line[:width-3] + '...'

                        stdscr.addstr(start_line + idx, 0, line.ljust(width))
                    except (psutil.NoSuchProcess, KeyError):
                        continue
```

```
case 3:
    try:
        start_line = 1
        max_connections = height - start_line - 1
        with threadLock:
            for y in range(height, height-1):
                stdscr.move(y, 0)
                stdscr.clrtoeol()

        header = f"    PID    TYPE      LADDR                          RADDR"
        stdscr.addstr(0, 0, header.ljust(width), curses.color_pair(1) | curses.A_BOLD)
        udp_sockets = psutil.net_connections(kind='udp')
        udp_info = [
            {"type": "UDP",
             "laddr": conn.laddr,
             "raddr": conn.raddr,
             "pid": conn.pid,
             } for conn in udp_sockets
        ]
        with threadLock:
            for idx, proc in enumerate(udp_info[:max_connections]):
                try:
                        if start_line + idx >= height - 1:
                            break
                        line = (f"{proc['pid']:7}    {proc['type']:5}     "
                                f"{':'.join(map(str, proc['laddr'])) if proc['laddr'] != 'N/A' else 'N/A':23}      "
                                f"{':'.join(map(str, proc['raddr'])) if proc['raddr'] != 'N/A' else 'N/A':2}")

                        if len(line) > width:
                            line = line[:width-3] + '...'

                        stdscr.addstr(start_line + idx, 0, line.ljust(width))
                    except (psutil.NoSuchProcess, KeyError):
                        continue
```

Vsak način prikaza vključuje skrb za čiščenje zaslona pred izrisom novih podatkov, da se prepreči prekrivanje ali zameglitev vsebine. Poleg tega funkcija upošteva višino in širino terminala, da prilagodi število prikazanih vrstic in prepreči preseganje meja zaslona. Zasnova omogoča tudi obvladovanje napak, saj funkcija ignorira napake, kot so dostopne omejitve ali neobstoječi procesi, in nadaljuje z obdelavo preostalih podatkov.

Zaključek



Glede na zgornjo sliko končnega izgleda implementacije nadgradnje orodja “top” se vidi, da sem se solidno približal končnemu cilju naloge.
Na žalost mi je zelo primanjkovalo časa zaradi tudi drugih študijskih obveznosti, zato implementacija ni niti približno optimizirana na nivoju, ki bi si ga želel, ampak deluje in tudi omogoča skoraj vse funkcionalnosti, ki so zahtevane v opisu predloga za projektno nalogo.
Ko bom imel čas, se bom definitivno vrnil k temu projektu in ga dokončal na način, da bo dejansko optimalno deloval.