LAPORAN PROGRESS 4(MINGGU 14)

Impelementasi kill process dan pengujian stress

PROJECT 3

1. Tujuan

Tujuan dari Progress 4 adalah melanjutkan progress 3 Task Monitor dengan melakukan pembuatan file program serta melakukan uji stress pada sistem. Pengujian ini bertujuan untuk melihat kestabilan program dalam memonitor penggunaan CPU dan RAM kerika sistem berada pada kondisi beban tinggi.

2. Langkah-langkah Pelaksanaan

2.1 Membuat File Program

Pada tahap ini dilakukan pembuatan file program Task Monitor menggunakan nano pada terminal Linux.

```
riki@DESKTOP-VC4979A:~/SO_PROJECT$ nano process_manager1.py
```

2.2 Mengisi Program Task Monitor

Setelah file program berhasil dibuat, dilakukan penulisan source code ke dalam file process_manager.py. program berisi implementasi monitoring CPU dan RAM secara real-time menggunakan psutil.

```
  GNU nano 7.2                                                process_manager1.
import psutil
import os
import signal
import time

def show_processes():
    print("\n=== DAFTAR PROSES (TOP CPU) ===")

    # Priming CPU counter
    for p in psutil.process_iter():
        try:
            p.cpu_percent(interval=None)
            cpu = p.cpu_percent(interval=None)
        except psutil.NoSuchProcess:
            pass

    time.sleep(0.1)

    processes = []
    for p in psutil.process_iter(['pid', 'name', 'cpu_percent']):
        try:
            processes.append(p.info)
        except psutil.NoSuchProcess:
            pass

    processes = sorted(processes, key=lambda x: x['cpu_percent'], reverse=True)[:10]

    for p in processes:
        print(f"PID: {p['pid']} | {p['name']} | CPU: {p['cpu_percent']}%")
```

```python
def kill_process():
    try:
        pid = int(input("\nMasukkan PID: "))
        print("1. SIGTERM (Normal)")
        print("2. SIGKILL (Paksa)")
        choice = input("Pilih signal [1/2]: ")

        if choice == "1":
            os.kill(pid, signal.SIGTERM)
            print("SIGTERM dikirim.")
        elif choice == "2":
            os.kill(pid, signal.SIGKILL)
            print("SIGKILL dikirim.")
        else:
            print("Pilihan tidak valid.")

    except PermissionError:
        print("Permission denied (butuh sudo).")
    except ProcessLookupError:
        print("PID tidak ditemukan.")
    except ValueError:
        print("Input PID tidak valid.")
```


```python
def kill_process():
    try:
        pid = int(input("\nMasukkan PID: "))
        print("1. SIGTERM (Normal)")
        print("2. SIGKILL (Paksa)")
        choice = input("Pilih signal [1/2]: ")

        if choice == "1":
            os.kill(pid, signal.SIGTERM)
            print("SIGTERM dikirim.")
        elif choice == "2":
            os.kill(pid, signal.SIGKILL)
            print("SIGKILL dikirim.")
        else:
            print("Pilihan tidak valid.")

    except PermissionError:
        print("Permission denied (butuh sudo).")
    except ProcessLookupError:
        print("PID tidak ditemukan.")
    except ValueError:
        print("Input PID tidak valid.")
```


```python
while True:
    print("\n=== PROCESS MANAGER ===")
    print("1. Lihat Proses")
    print("2. Kill Proses")
    print("3. Keluar")

    menu = input("Pilih menu: ")

    if menu == "1":
        show_processes()
    elif menu == "2":
        kill_process()
    elif menu == "3":
        break
    else:
        print("Menu tidak valid.")
```

2.3 Melakukan Uji Stress Sistem

Setelah program selesai dibuat,dilakukan uji stress pada sistem menggunakan perintah stress atau stress-ng melalui terminal. Uji stress difokuskan pembebanan CPU dan RAM untuk mengamati perubahan penggunaan sumber daya yang dimonitor oleh program.

```
riki@DESKTOP-VC4979A:~/SO_PROJECT$ yes > /dev/null
```

```
=== PROCESS MANAGER ===
1. Lihat Proses
2. Kill Proses
3. Keluar
Pilih menu: 1

=== DAFTAR PROSES (TOP CPU) ===
PID: 1957 | yes | CPU: 91.7%
PID: 1906 | python3 | CPU: 9.2%
PID: 1 | systemd | CPU: 0.0%
PID: 2 | init-systemd(Ub | CPU: 0.0%
PID: 6 | init | CPU: 0.0%
PID: 48 | systemd-journald | CPU: 0.0%
PID: 102 | systemd-udevd | CPU: 0.0%
PID: 113 | systemd-resolved | CPU: 0.0%
PID: 114 | systemd-timesyncd | CPU: 0.0%
PID: 176 | cron | CPU: 0.0%

=== PROCESS MANAGER ===
1. Lihat Proses
2. Kill Proses
3. Keluar
Pilih menu: 2

Masukkan PID: 1957
1. SIGTERM (Normal)
2. SIGKILL (Paksa)
Pilih signal [1/2]: 1
SIGTERM dikirim.

=== PROCESS MANAGER ===
1. Lihat Proses
2. Kill Proses
3. Keluar
Pilih menu: 1

=== DAFTAR PROSES (TOP CPU) ===
PID: 1906 | python3 | CPU: 9.0%
PID: 1 | systemd | CPU: 0.0%
PID: 2 | init-systemd(Ub | CPU: 0.0%
PID: 6 | init | CPU: 0.0%
PID: 48 | systemd-journald | CPU: 0.0%
PID: 102 | systemd-udevd | CPU: 0.0%
PID: 113 | systemd-resolved | CPU: 0.0%
PID: 114 | systemd-timesyncd | CPU: 0.0%
PID: 176 | cron | CPU: 0.0%
PID: 177 | dbus-daemon | CPU: 0.0%
```

```
=== PROCESS MANAGER ===
1. Lihat Proses
2. Kill Proses
3. Keluar
Pilih menu: 1

=== DAFTAR PROSES (TOP CPU) ===
PID: 1903 | yes | CPU: 90.7%
PID: 1 | systemd | CPU: 0.0%
PID: 2 | init-systemd(Ub | CPU: 0.0%
PID: 6 | init | CPU: 0.0%
PID: 48 | systemd-journald | CPU: 0.0%
PID: 102 | systemd-udevd | CPU: 0.0%
PID: 113 | systemd-resolved | CPU: 0.0%
PID: 114 | systemd-timesyncd | CPU: 0.0%
PID: 176 | cron | CPU: 0.0%
PID: 177 | dbus-daemon | CPU: 0.0%

=== PROCESS MANAGER ===
1. Lihat Proses
2. Kill Proses
3. Keluar
Pilih menu: 2

Masukkan PID: 1903
1. SIGTERM (Normal)
2. SIGKILL (Paksa)
Pilih signal [1/2]: 2
SIGKILL dikirim.

=== PROCESS MANAGER ===
1. Lihat Proses
2. Kill Proses
3. Keluar
Pilih menu: 1

=== DAFTAR PROSES (TOP CPU) ===
PID: 1 | systemd | CPU: 0.0%
PID: 2 | init-systemd(Ub | CPU: 0.0%
PID: 6 | init | CPU: 0.0%
PID: 48 | systemd-journald | CPU: 0.0%
PID: 102 | systemd-udevd | CPU: 0.0%
PID: 113 | systemd-resolved | CPU: 0.0%
PID: 114 | systemd-timesyncd | CPU: 0.0%
PID: 176 | cron | CPU: 0.0%
PID: 177 | dbus-daemon | CPU: 0.0%
PID: 193 | systemd-logind | CPU: 0.0%
```

3. **Hasil yang Diperoleh**

   Berdasarkan pelaksanan Progress 4, diperoleh hasil sebagai berikut.

   1. File program berhasil dibuat menggunakan nano
   2. Source code Task Monitor berhasil dituliskan dan disimpan tanpa error.
   3. Program mampu memonitor penggunaan CPU dan RAM sesuai dengan implementasi sebelumnya.
   4. Saat uji stress dijalankan, penggunaan COU dan RAM mengalami pengingkatan sesuai beban yang diberikan.
   5. Program Task Monitor tetap berjalan stabil selama proses uji stress berlangsung