LAPORAN SISTEM OPERASI

FINAL PROJECT 3

Monitoring Resource System (Linux Task Manager Advanced)

Dosen Pengampu:

Ferdi Chahyadi, S.Kom., M.Cs

Disusun Oleh:

Riki Andika Saputra	2401020134
Muhamad Isra Dwi Firmansya	2401020143
Bunga Salsabila Pebriyani	2401020150
Devina Rindumasha	2401020153

PRODI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN

UNIVERSITAS MARITIM RAJA ALI HAJI

2025/2026

ABSTRAK

Monitoring sumber daya sistem merupakan salah satu aspek penting dalam sistem operasi, khususnya pada sistem operasi Linux yang bersifat multitasking dan multiuser. Banyaknya proses yang berjalan secara bersamaan dapat mempengaruhi kinerja CPU dan penggunaan memori (RAM). Oleh karena itu, diperlukan sebuah aplikasi yang mampu menampilkan kondisi penggunaan sumber daya sistem secara real-time.

Final Project ini bertujuan untuk merancang dan mengimplementasikan aplikasi *Linux Task Monitor* berbasis terminal menggunakan bahasa pemrograman Python dan library **psutil**. Aplikasi ini digunakan untuk memantau penggunaan CPU dan RAM secara real-time pada sistem operasi Linux. Proses pengembangan dilakukan melalui beberapa tahapan, mulai dari pembuatan file program, penulisan source code, hingga pengujian sistem menggunakan metode *stress testing*.

Hasil pengujian menunjukkan bahwa aplikasi Linux Task Monitor mampu menampilkan perubahan penggunaan CPU dan RAM dengan baik, baik pada kondisi normal maupun saat sistem berada dalam kondisi beban tinggi. Project ini diharapkan dapat memberikan pemahaman praktis mengenai konsep manajemen proses, manajemen memori, serta monitoring resource pada sistem operasi Linux.

Kata kunci: Sistem Operasi, Task Monitor, Monitoring Resource, CPU, RAM, Linux, Python, psutil

BAB I

PENDAHULUAN

1.1 Latar Belakang

Sistem operasi berperan penting dalam mengelola seluruh sumber daya komputer, seperti CPU, memori, dan proses. Pada sistem operasi Linux, berbagai proses dapat berjalan secara bersamaan sehingga penggunaan sumber daya sistem perlu dipantau secara berkala. Tanpa adanya monitoring yang baik, pengguna akan kesulitan mengetahui kondisi performa sistem.

Monitoring CPU dan RAM menjadi hal yang penting untuk mengetahui beban kerja sistem, mendeteksi terjadinya penggunaan sumber daya yang berlebihan, serta membantu dalam proses analisis kinerja sistem. Oleh karena itu, diperlukan sebuah aplikasi sederhana yang mampu menampilkan informasi penggunaan CPU dan RAM secara real-time.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah dalam project ini adalah:

1. Bagaimana cara memonitor penggunaan CPU dan RAM pada sistem operasi Linux secara *real-time*?
2. Bagaimana cara mengimplementasikan aplikasi task monitor menggunakan bahasa pemrograman Python?
3. Bagaimana kestabilan aplikasi monitoring saat sistem berada pada kondisi beban tinggi?

1.3 Tujuan

Tujuan dari pelaksanaan Final Project ini adalah:

- Merancang dan mengimplementasikan aplikasi Linux Task Monitor berbasis terminal.
- Menampilkan informasi penggunaan CPU dan RAM secara real-time.
- Melakukan pengujian sistem menggunakan metode stress testing.

BAB II

LANDASAN TEORI

2.1 Sistem Operasi Linux

Linux merupakan sistem operasi berbasis open-source yang dikembangkan dengan konsep multitasking dan multiuser. Linux banyak digunakan pada server, desktop, serta lingkungan pengembangan karena stabilitas dan keamanannya. Dalam Linux, pengelolaan sumber daya seperti CPU dan memori dilakukan oleh kernel, sehingga monitoring terhadap sumber daya tersebut menjadi sangat penting untuk menjaga performa sistem.

2.2 Manajemen Proses

Manajemen proses adalah bagian dari sistem operasi yang bertanggung jawab dalam mengatur proses yang sedang berjalan. Proses merupakan program yang sedang dieksekusi dan membutuhkan sumber daya CPU untuk dapat berjalan. Sistem operasi mengatur penjadwalan proses agar penggunaan CPU dapat dibagi secara adil dan efisien. Monitoring penggunaan CPU membantu dalam mengetahui beban kerja sistem dan jumlah proses yang aktif.

2.3 Manajemen Memori

Manajemen memori berfungsi untuk mengatur penggunaan RAM agar dapat dimanfaatkan secara optimal oleh sistem dan aplikasi. Sistem operasi harus memastikan bahwa setiap proses mendapatkan alokasi memori yang cukup tanpa menyebabkan pemborosan. Monitoring memori digunakan untuk mengetahui kondisi memori total, memori terpakai, dan memori bebas sehingga dapat mencegah terjadinya kehabisan memori.

2.4 Monitoring Resource System

Monitoring resource sistem merupakan proses pengamatan penggunaan sumber daya sistem secara berkala. Monitoring CPU dan RAM memberikan informasi penting mengenai performa sistem. Dengan monitoring yang baik, pengguna dapat mengetahui kondisi sistem secara *real-time* dan mengambil tindakan yang diperlukan ketika terjadi lonjakan penggunaan sumber daya.

2.5 Bahasa Pemrograman Phyton

Python adalah bahasa pemrograman tingkat tinggi yang bersifat interpreted dan mudah dipahami. Python banyak digunakan dalam pengembangan aplikasi sistem karena memiliki sintaks yang sederhana dan dukungan library yang lengkap. Dalam project ini, Python digunakan sebagai bahasa utama untuk mengimplementasikan aplikasi Task Monitor.

2.6 Library Psutil

psutil merupakan library Python yang digunakan untuk mengambil informasi sistem seperti penggunaan CPU, memori, dan proses. Library ini menyediakan fungsi-fungsi yang memudahkan pengembang dalam melakukan monitoring resource sistem secara *real-time*. Penggunaan psutil memungkinkan aplikasi Task Monitor memperoleh data sistem secara akurat.

2.7 Stress Testing

Stress testing adalah metode pengujian sistem dengan memberikan beban tinggi untuk mengetahui batas kemampuan dan kestabilan sistem. Pengujian ini penting untuk memastikan bahwa aplikasi monitoring tetap berjalan dengan baik ketika sistem berada pada kondisi ekstrem.

BAB III

PERANCANGAN SISTEM

3.1 Gambaran Umum Sistem

Sistem Task Monitor dirancang sebagai aplikasi monitoring resource yang berjalan pada sistem operasi Linux. Aplikasi ini berjalan melalui terminal dan bertujuan untuk menampilkan informasi penggunaan CPU dan RAM secara *real-time*. Sistem dirancang sederhana agar mudah dipahami dan digunakan.

3.2 Arsitektur Sistem

Arsitektur sistem Linux Task Monitor dirancang dengan pendekatan sederhana dan efisien. Data penggunaan CPU dan RAM diambil secara langsung dari sistem operasi menggunakan library psutil dan ditampilkan ke terminal secara real-time. Data monitoring ditampilkan secara langsung ke terminal tanpa disimpan sebagai histori, sehingga aplikasi tetap ringan dan sederhana. Pendekatan ini dipilih agar aplikasi tidak membebani sistem serta mudah dipahami dalam konteks pembelajaran sistem operasi.

3.3 Diagram Alur Sistem

Alur kerja aplikasi Linux Task Monitor dimulai dengan proses inisialisasi program pada sistem operasi Linux. Setelah program dijalankan, sistem akan mengakses informasi penggunaan sumber daya komputer melalui library *psutil*. Data yang diambil meliputi persentase penggunaan CPU dan penggunaan memori (RAM) pada sistem.

Selanjutnya, data penggunaan CPU dan RAM tersebut ditampilkan secara langsung ke terminal dalam bentuk informasi teks yang mudah dibaca oleh pengguna. Informasi yang ditampilkan mencerminkan kondisi sistem pada saat itu dan diperbarui secara berkala sesuai dengan interval waktu yang telah ditentukan di dalam program.

Proses pengambilan dan penampilan data ini berjalan secara terus-menerus selama program masih dijalankan. Aplikasi Linux Task Monitor tidak melakukan penyimpanan data penggunaan sistem ke dalam bentuk histori, sehingga setiap pembaruan data yang ditampilkan merupakan kondisi terbaru dari sistem. Pendekatan ini dipilih agar aplikasi tetap sederhana, ringan, dan tidak membebani kinerja sistem.

3.4 Design Program

Design program Task Monitor dibuat secara modular agar mudah dikembangkan. Program terdiri dari bagian import library, inisialisasi variabel, loop utama monitoring, serta fungsi untuk menampilkan output. Design ini memudahkan proses debugging dan pengembangan lanjutan.

3.5 Design Input dan Output

Input dari sistem ini berupa data penggunaan CPU dan RAM yang diperoleh langsung dari sistem operasi melalui library psutil. Output dari sistem berupa tampilan informasi penggunaan CPU dan RAM yang ditampilkan pada terminal dalam format teks yang terstruktur dan mudah dibaca.

BAB IV

IMPLEMENTASI SISTEM

4.1 Lingkungan Pengembangan

Implementasi aplikasi *Task Monitor Sistem Operasi* dilakukan pada lingkungan sistem operasi Linux yang dijalankan melalui Oracle VirtualBox. Penggunaan mesin virtual bertujuan untuk menyediakan lingkungan pengembangan yang terisolasi, stabil, dan sesuai dengan kebutuhan praktikum Sistem Operasi. Sistem operasi Linux dipilih karena memiliki dukungan penuh terhadap pemantauan proses dan sumber daya sistem.

Bahasa pemrograman yang digunakan dalam project ini adalah Python 3, yang telah dipastikan terinstal dan dapat dijalankan dengan baik pada sistem. Python dipilih karena kemudahan sintaks, fleksibilitas, serta ketersediaan library monitoring sistem yang lengkap.

4.2 Persiapan dan Instalasi Library Pendukung

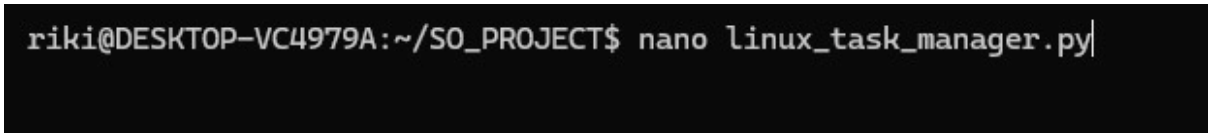
Sebelum program dikembangkan, dilakukan instalasi library pendukung yang diperlukan untuk monitoring sistem. Library utama yang digunakan adalah **psutil**, yang berfungsi untuk mengambil informasi penggunaan CPU dan RAM dari sistem operasi. Selain itu, dilakukan instalasi **pip3** sebagai package manager Python dan **python3-tk** untuk mendukung kebutuhan visualisasi apabila dikembangkan lebih lanjut.

Proses instalasi dilakukan melalui terminal Linux dan dipastikan seluruh library dapat digunakan tanpa error sebelum masuk ke tahap pembuatan program.

4.3 Pembuatan File Program Menggunakan Nano

Tahap selanjutnya adalah pembuatan file program utama menggunakan editor teks **nano**. File program dibuat dengan nama **linux_task_manager.py**. Penggunaan nano dipilih karena merupakan editor teks standar pada Linux dan mudah digunakan melalui terminal.

Pada tahap ini, struktur dasar file program disiapkan sebagai wadah untuk seluruh kode program Task Monitor. Pembuatan file ini merupakan langkah awal sebelum dilakukan penulisan source code.



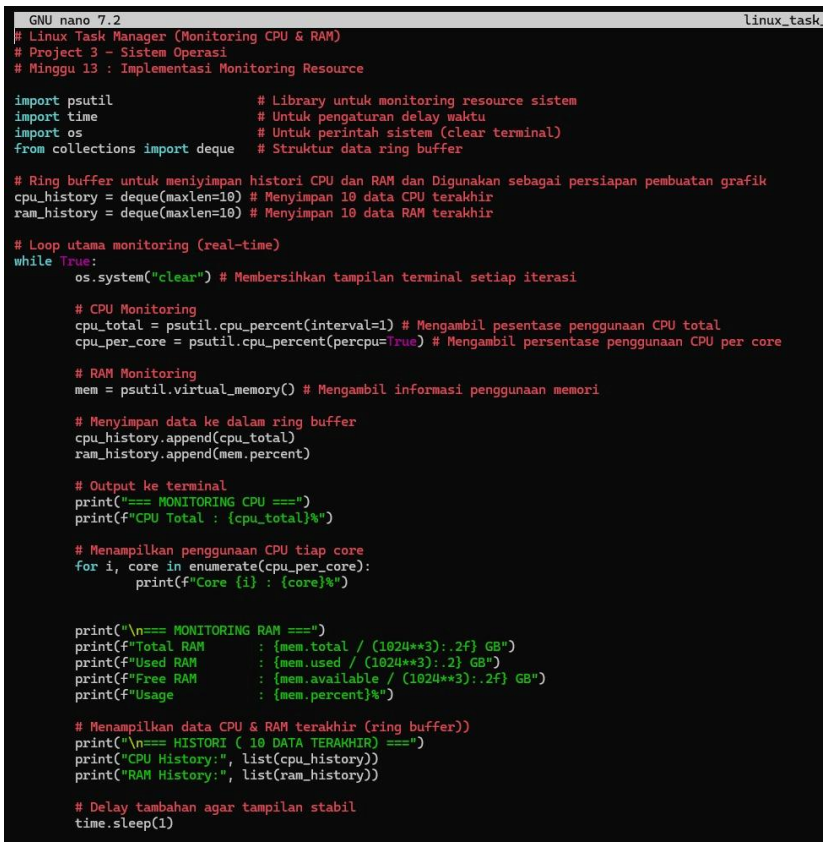
Gambar 4.1 Pembuatan file program linux_task_manager.py menggunakan nano

4.4 Penulisan dan Implementasi Source Code Program

Setelah file program berhasil dibuat, dilakukan penulisan source code ke dalam file **linux_task_manager.py**. Source code mencakup proses import library yang dibutuhkan, inisialisasi variabel, serta pembuatan loop utama untuk monitoring sistem.

Implementasi monitoring CPU dilakukan dengan mengambil data penggunaan CPU total dan CPU per-core menggunakan fungsi yang tersedia pada library psutil. Selain itu, monitoring RAM diimplementasikan untuk menampilkan informasi total memori, memori terpakai, memori bebas, serta persentase penggunaan memori.

Seluruh data yang diperoleh ditampilkan secara *real-time* pada terminal dengan interval pembaruan setiap satu detik, sehingga pengguna dapat memantau kondisi sistem secara langsung.



Gambar 4.2 Tampilan source code program Task Monitor

BAB V

PENGUJIAN DAN ANALISIS

5.1 Pengujian Pembuatan dan Penulisan Program

Pengujian awal dilakukan untuk memastikan bahwa file program **task_proses_manager.py** berhasil dibuat dan source code dapat dituliskan dengan benar menggunakan editor nano. Pada tahap ini, dicek apakah file dapat disimpan dan dibuka kembali tanpa mengalami error.

Hasil pengujian menunjukkan bahwa proses pembuatan file dan penulisan source code berjalan dengan baik dan siap untuk tahap pengujian berikutnya.

5.2 Pengujian Monitoring System

Setelah program selesai ditulis, dilakukan pengujian monitoring sistem pada kondisi normal. Program dijalankan melalui terminal Linux dan diamati output yang ditampilkan.

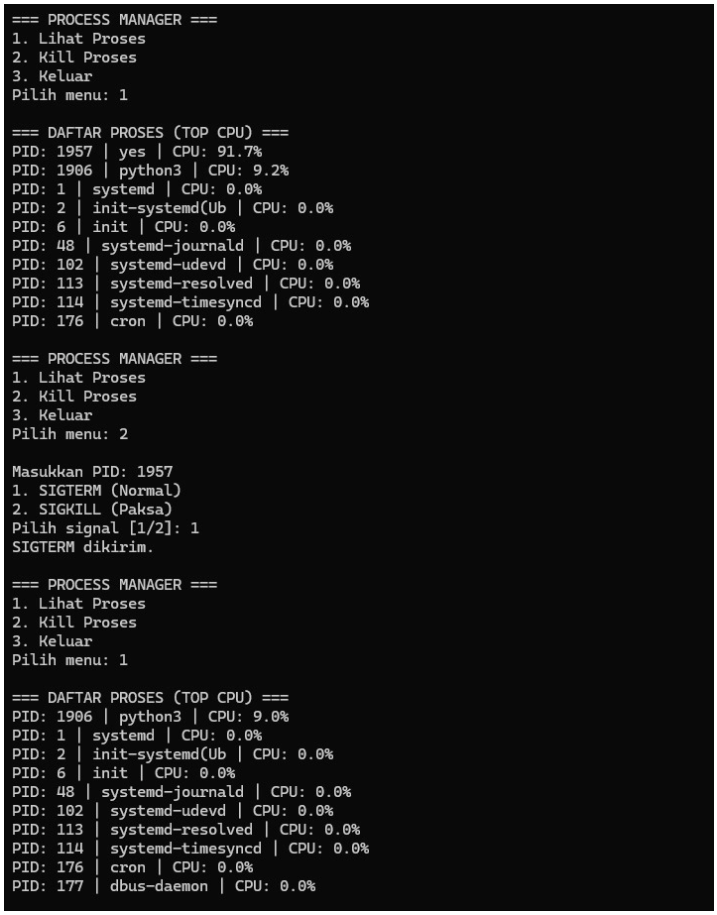
Pada tahap ini, aplikasi berhasil menampilkan penggunaan CPU total, CPU per-core, serta penggunaan RAM secara *real-time*. Pembaruan data berjalan stabil setiap satu detik dan tidak ditemukan kendala selama program dijalankan.

5.3 Pengujian Stress Sistem

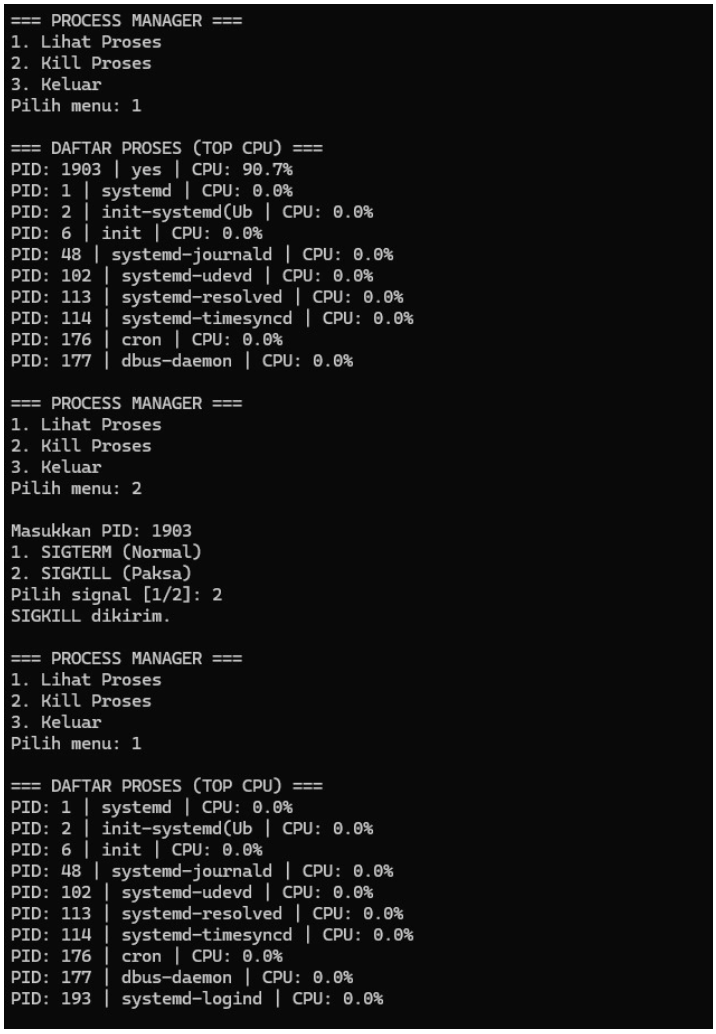
Pengujian *stress* dilakukan untuk mengetahui kestabilan aplikasi ketika sistem berada pada kondisi beban tinggi. Pengujian dilakukan dengan menjalankan perintah **stress** atau **stress-ng** pada terminal untuk memberikan beban pada CPU dan RAM.

Selama uji *stress* berlangsung, aplikasi Task Monitor tetap dijalankan secara bersamaan. Terlihat adanya peningkatan penggunaan CPU dan RAM sesuai dengan beban yang diberikan. Program tetap berjalan stabil dan mampu menampilkan perubahan penggunaan sumber daya sistem secara *real-time*.

Gambar 5.1 Proses uji stress CPU dan RAM pada sistem



Gambar 5.2 Proses uji stress CPU dan RAM pada sistem



Gambar 5.3 Proses uji stress CPU dan RAM pada sistem

5.4 Analisis Hasil Pengujian

Berdasarkan seluruh pengujian yang telah dilakukan, dapat dianalisis bahwa aplikasi *Task Monitor Sistem Operasi* mampu bekerja sesuai dengan tujuan yang telah ditetapkan. Program tidak mengalami *crash* selama pengujian, dan mekanisme monitoring berjalan dengan baik baik pada kondisi normal maupun saat sistem berada dalam kondisi *stress*.

Hasil ini menunjukkan bahwa implementasi monitoring CPU dan RAM menggunakan Python dan library psutil telah berhasil dan dapat dijadikan sebagai dasar pengembangan aplikasi monitoring yang lebih lanjut.

BAB VI

KESIMPULAN

Berdasarkan seluruh tahapan yang telah dilaksanakan dalam Final Project mata kuliah Sistem Operasi ini, dapat disimpulkan bahwa project *Task Monitor Sistem Operasi* berhasil dirancang, diimplementasikan, dan diuji sesuai dengan tujuan yang telah ditetapkan. Proses pengembangan aplikasi dilakukan secara bertahap dan sistematis, dimulai dari persiapan lingkungan dan instalasi library pendukung, penulisan dan implementasi source code, hingga tahap pengujian sistem.

Aplikasi Task Monitor yang dikembangkan mampu melakukan monitoring penggunaan CPU dan RAM pada sistem operasi Linux secara *real-time*. Informasi yang ditampilkan meliputi penggunaan CPU secara keseluruhan dan per-core, serta penggunaan memori yang mencakup total memori, memori terpakai, memori bebas, dan persentase penggunaan. Dengan adanya informasi ini, pengguna dapat mengetahui kondisi performa sistem secara langsung dan lebih mudah memahami bagaimana sistem operasi mengelola sumber daya.

DAFTAR PUSTAKA

DeWitt, T., Miller, N., Gross, T., Steenkiste, P., & Sutherland, D. (1997). *Remos: A resource monitoring system for network-aware applications* (No. CMUCS97194).

Dobbins, D. A. (1982). MONITOR PERFORMANCE TASK.

Danielsen, F., Eicken, H., Funder, M., Johnson, N., Lee, O., Theilade, I., ... & Burgess, N. D. (2022). Community monitoring of natural resource systems and the environment. *Annual review of environment and resources*, *47*(1), 637-670.

Chung, W. C., & Chang, R. S. (2009). A new mechanism for resource monitoring in grid computing. *Future Generation Computer Systems*, *25*(1), 1-7.

Zhang, X., Freschl, J. L., & Schopf, J. M. (2003, June). A performance study of monitoring and information services for distributed systems. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on* (pp. 270-281). IEEE.

Fang, S., Da Xu, L., Zhu, Y., Ahati, J., Pei, H., Yan, J., & Liu, Z. (2014). An integrated system for regional environmental monitoring and management based on internet of things. *IEEE Transactions on Industrial Informatics*, *10*(2), 1596-1605.

Zanikolas, S., & Sakellariou, R. (2005). A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, *21*(1), 163-188.

LAMPIRAN

—Konfigurasi dan Lingkungan Sistem

- Sistem Operasi : Linux (Ubuntu)
- Media Instalasi : Oracle VirtualBox
- Bahasa Pemrograman : Python 3
- Editor Teks : nano
- Library Pendukung :
  - psutil
  - pip3
- Tools Pengujian :
  - stress / stress-ng

—Source Code Program Task Monitor

```
# Linux Task Manager (Monitoring CPU & RAM)

# Project 3 - Sistem Operasi

# Minggu 13 : Implementasi Monitoring Resource


import psutil                   # Library untuk monitoring resource sistem

import time                     # Untuk pengaturan delay waktu

import os                       # Untuk perintah sistem (clear terminal)

from collections import deque   # Struktur data ring buffer


# Ring buffer untuk meniyimpan histori CPU dan RAM dan Digunakan sebagai persiapan pembuatan grafik

cpu_history = deque(maxlen=10) # Menyimpan 10 data CPU terakhir

ram_history = deque(maxlen=10) # Menyimpan 10 data RAM terakhir


# Loop utama monitoring (real-time)

while True:

        os.system("clear") # Membersihkan tampilan terminal setiap iterasi


        # CPU Monitoring

        cpu_total = psutil.cpu_percent(interval=1) # Mengambil pesentase penggunaan CPU total
```

```
        cpu_per_core = psutil.cpu_percent(percpu=True) # Mengambil persentase penggunaan CPU per core


        # RAM Monitoring
        mem = psutil.virtual_memory() # Mengambil informasi penggunaan memori


        # Menyimpan data ke dalam ring buffer
        cpu_history.append(cpu_total)
        ram_history.append(mem.percent)


        # Output ke terminal
        print("=== MONITORING CPU ===")
        print(f"CPU Total : {cpu_total}%")


        # Menampilkan penggunaan CPU tiap core
        for i, core in enumerate(cpu_per_core):
                print(f"Core {i} : {core}%")



        print("\n=== MONITORING RAM ===")
        print(f"Total RAM       : {mem.total / (1024**3):.2f} GB")
        print(f"Used RAM        : {mem.used / (1024**3):.2} GB")
        print(f"Free RAM        : {mem.available / (1024**3):.2f} GB")
        print(f"Usage           : {mem.percent}%")


        # Menampilkan data CPU & RAM terakhir (ring buffer))
        print("\n=== HISTORI ( 10 DATA TERAKHIR) ===")
        print("CPU History:", list(cpu_history))
        print("RAM History:", list(ram_history))


        # Delay tambahan agar tampilan stabil
        time.sleep(1)
```

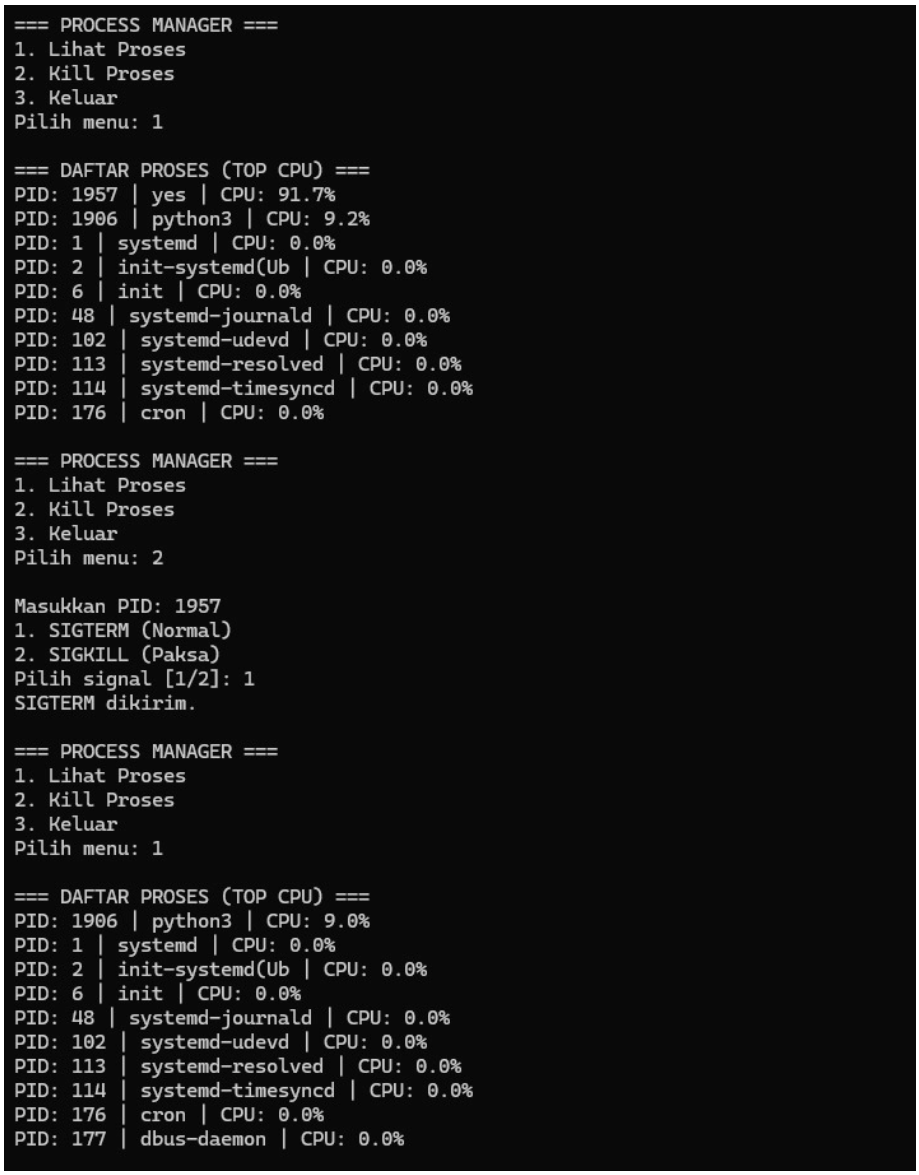
—Screenshot Pembuatan File Program

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

—Screenshot Isi Program

```
  GNU nano 7.2                                                                linux_task_
# Linux Task Manager (Monitoring CPU & RAM)
# Project 3 - Sistem Operasi
# Minggu 13 : Implementasi Monitoring Resource

import psutil                   # Library untuk monitoring resource sistem
import time                     # Untuk pengaturan delay waktu
import os                       # Untuk perintah sistem (clear terminal)
from collections import deque   # Struktur data ring buffer

# Ring buffer untuk meniyimpan histori CPU dan RAM dan Digunakan sebagai persiapan pembuatan grafik
cpu_history = deque(maxlen=10) # Menyimpan 10 data CPU terakhir
ram_history = deque(maxlen=10) # Menyimpan 10 data RAM terakhir

# Loop utama monitoring (real-time)
while True:
        os.system("clear") # Membersihkan tampilan terminal setiap iterasi

        # CPU Monitoring
        cpu_total = psutil.cpu_percent(interval=1) # Mengambil pesentase penggunaan CPU total
        cpu_per_core = psutil.cpu_percent(percpu=True) # Mengambil persentase penggunaan CPU per core

        # RAM Monitoring
        mem = psutil.virtual_memory() # Mengambil informasi penggunaan memori

        # Menyimpan data ke dalam ring buffer
        cpu_history.append(cpu_total)
        ram_history.append(mem.percent)

        # Output ke terminal
        print("=== MONITORING CPU ===")
        print(f"CPU Total : {cpu_total}%")

        # Menampilkan penggunaan CPU tiap core
        for i, core in enumerate(cpu_per_core):
                print(f"Core {i} : {core}%")


        print("\n=== MONITORING RAM ===")
        print(f"Total RAM       : {mem.total / (1024**3):.2f} GB")
        print(f"Used RAM        : {mem.used / (1024**3):.2} GB")
        print(f"Free RAM        : {mem.available / (1024**3):.2f} GB")
        print(f"Usage           : {mem.percent}%")

        # Menampilkan data CPU & RAM terakhir (ring buffer))
        print("\n=== HISTORI ( 10 DATA TERAKHIR) ===")
        print("CPU History:", list(cpu_history))
        print("RAM History:", list(ram_history))

        # Delay tambahan agar tampilan stabil
        time.sleep(1)
```

—Screenshot Uji Stress System



```
=== 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%
```