LAPORAN MINGGU KE -13

MONITORING RESOURCE SYSTEM (LINUX TASK MANAGER ADVENCED)

Dosen Pengampu:

Ferdi Chahyadi, S.Kom., M.Cs

Disusun Oleh:

Siti Umayah	2401020018
Anika	2401020029
Aldi Saputra	2401020024
Fauziah Sal Sabillah	2401020034

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN

UNIVERSITAS MARITIM RAJA ALI HAJI

TANJUNGPINANG

2025

ABSTRAK

Laporan ini membahas progress pelaksanaan proyek Monitoring Resource System (Linux Task Manager Advanced) pada minggu ke-13 yang fokus pada implementasi monitoring CPU dan RAM secara real-time. Pada minggu sebelumnya, environment development telah berhasil disetup dengan lengkap termasuk instalasi Linux, Python, dan library psutil. Minggu ini dilanjutkan dengan implementasi core functionality aplikasi, yaitu monitoring penggunaan CPU dan RAM dengan berbagai parameter seperti CPU usage per-core, CPU frequency, total memory, used memory, available memory, swap memory, serta implementasi auto-update untuk refresh data secara berkala. Implementasi menggunakan Python dengan library psutil untuk mengakses informasi sistem dan library time untuk handling interval update. Hasil implementasi menunjukkan bahwa aplikasi berhasil menampilkan data monitoring CPU dan RAM secara real-time dengan akurasi yang baik dan update interval yang stabil. Testing dilakukan dengan berbagai skenario beban sistem untuk memastikan aplikasi dapat bekerja dengan baik dalam kondisi normal maupun high-load. Dengan selesainya implementasi monitoring CPU/RAM, proyek siap dilanjutkan ke tahap berikutnya yaitu implementasi kill process dan stress testing.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Monitoring resource sistem merupakan komponen fundamental dalam manajemen sistem operasi modern. CPU dan RAM adalah dua resource kritis yang penggunaannya perlu dipantau secara kontinyu untuk memastikan sistem berjalan optimal. CPU (Central Processing Unit) bertanggung jawab atas eksekusi instruksi program, sedangkan RAM (Random Access Memory) menyimpan data yang sedang diproses oleh CPU.

Dalam sistem operasi Linux, informasi tentang CPU dan RAM dapat diakses melalui filesystem virtual /proc dan /sys. Namun, parsing data mentah dari filesystem ini cukup kompleks dan memerlukan pemahaman mendalam tentang struktur data kernel. Library psutil menyediakan abstraksi high-level yang memudahkan akses ke informasi sistem ini tanpa perlu berinteraksi langsung dengan kernel interface.

Monitoring CPU mencakup beberapa aspek penting seperti overall CPU usage, per-core CPU usage, CPU frequency, load average, dan context switches. Sedangkan monitoring RAM mencakup total memory, used memory, available memory, buffer/cache, swap usage, dan memory percentage. Data-data ini perlu ditampilkan secara real-time dengan update interval yang konsisten untuk memberikan gambaran akurat tentang kondisi sistem.

1.2 Recap Minggu Sebelumnya

Pada minggu ke-12, telah berhasil dilakukan:

1. Instalasi sistem operasi Linux (Ubuntu 24.04 LTS)
2. Setup Python development environment
3. Instalasi library psutil dan dependensi lainnya
4. Testing dasar untuk memastikan library dapat mengakses informasi sistem
5. Dokumentasi lengkap proses instalasi

Environment development yang telah disiapkan pada minggu sebelumnya menjadi fondasi untuk implementasi pada minggu ini.

1.3 Scope Minggu ke-13

Pada minggu ke-13 ini, fokus utama adalah mengimplementasikan fungsi-fungsi monitoring CPU dan RAM yang akan menjadi core functionality aplikasi. Implementasi mencakup:

1. Fungsi monitoring CPU dengan berbagai parameter
2. Fungsi monitoring RAM dan swap memory
3. Mekanisme auto-update dengan interval yang dapat dikonfigurasi
4. Formatting dan display data yang user-friendly
5. Error handling untuk menangani edge cases
6. Testing dengan berbagai skenario beban sistem

1.4 Rumusan Masalah

Berdasarkan tujuan implementasi pada minggu ke-13, berikut adalah rumusan masalah yang perlu diselesaikan:

1. Bagaimana cara mengambil data CPU usage secara real-time dari sistem operasi Linux menggunakan psutil?
2. Bagaimana menampilkan CPU usage per-core untuk sistem multi-core processor?
3. Bagaimana mengimplementasikan monitoring RAM yang mencakup total memory, used memory, available memory, buffer/cache, dan swap memory?
4. Bagaimana membuat mekanisme auto-update yang dapat refresh data secara berkala tanpa membebani sistem?
5. Bagaimana menghitung dan menampilkan persentase penggunaan CPU dan RAM dengan akurat?
6. Bagaimana menangani exception dan error yang mungkin terjadi saat mengakses informasi sistem?
7. Bagaimana memformat dan menampilkan data monitoring dengan cara yang mudah dibaca dan informatif?

BAB II

TUJUAN PROJECT

2.1 Tujuan Umum

Mengembangkan aplikasi monitoring resource system berbasis Linux yang mampu memantau penggunaan CPU, RAM, disk, dan proses secara real-time, serta memiliki kemampuan untuk mengelola proses menggunakan signal handling.

2.2 Tujuan Khusus Minggu ke -13

Pada minggu ke-13 ini, tujuan khusus yang ingin dicapai adalah:

1. Mengimplementasikan fungsi monitoring CPU yang dapat menampilkan:
    a. Overall CPU usage (percentage)
    b. Per-core CPU usage untuk setiap logical processor
    c. CPU frequency (current, min, max)
    d. CPU count (physical dan logical cores)
    e. CPU load average
2. Mengimplementasikan fungsi monitoring RAM yang dapat menampilkan:
    a. Total memory system
    b. Used memory dan available memory
    c. Memory usage percentage
    d. Buffer dan cache memory
    e. Swap memory (total, used, free, percentage)
3. Membuat mekanisme auto-update dengan fitur:
    a. Interval update yang dapat dikonfigurasi
    b. Refresh data secara otomatis
    c. Clear screen untuk tampilan yang bersih
    d. Update timestamp untuk setiap refresh

4. Mengimplementasikan formatting data yang user-friendly:
    a. Konversi bytes ke satuan yang lebih mudah dibaca (GB, MB)
    b. Progress bar atau visual indicator
    c. Alignment dan spacing yang baik
    d. Color coding (opsional) untuk status
5. Menambahkan error handling yang robust:
    a. Try-catch untuk exception handling
    b. Fallback value jika data tidak tersedia
    c. Log error untuk debugging
6. Melakukan testing dengan berbagai skenario:
    a. Normal load condition
    b. High CPU load
    c. High memory usage
    d. Multiple concurrent processes

2.3 Output yang Diharapkan

Pada akhir minggu ke-13, diharapkan:

1. Aplikasi dapat menampilkan CPU usage secara real-time
2. Aplikasi dapat menampilkan RAM usage secara real-time
3. Auto-update berfungsi dengan baik tanpa lag atau freeze
4. Data ditampilkan dengan format yang mudah dibaca
5. Testing menunjukkan aplikasi stabil dalam berbagai kondisi
6. Kode terstruktur dengan baik dan mudah dipelihara
7. Dokumentasi lengkap untuk setiap fungsi

BAB III

METODOLOGI

3.1 Arsitektur Aplikasi

Aplikasi dirancang dengan pendekatan modular dan object-oriented untuk memudahkan maintenance dan pengembangan selanjutnya

3.2 Tools dan Library

Library Python yang Digunakan:

1. psutil: Mengakses informasi CPU, RAM, dan sistem
2. time: Handling interval update dan sleep
3. os: Operasi sistem seperti clear screen
4. datetime: Timestamp untuk logging
5. sys: System-specific parameters dan functions

Development Tools:

1. Text Editor: Visual Studio Code
2. Version Control: Git
3. Testing: Manual testing dengan berbagai skenario

3.3 Metode Implementasi

Tahap 1: Analisis Kebutuhan

1. Mengidentifikasi data apa saja yang perlu ditampilkan
2. Menentukan struktur data dan format output
3. Merancang flow program

Tahap 2: Implementasi CPU Monitoring

1. Implementasi fungsi get CPU usage overall
2. Implementasi fungsi get CPU usage per-core
3. Implementasi fungsi get CPU frequency

4. Implementasi fungsi get CPU info

Tahap 3: Implementasi RAM Monitoring

1. Implementasi fungsi get memory info
2. Implementasi fungsi get swap info
3. Implementasi formatting memory size

Tahap 4: Implementasi Auto-Update

1. Implementasi infinite loop dengan interval
2. Implementasi clear screen function
3. Implementasi update timestamp

Tahap 5: Testing dan Debugging

1. Testing normal operation
2. Testing dengan high load
3. Fix bugs dan optimize performance

Tahap 6: Documentation

1. Menambahkan docstring di setiap fungsi
2. Membuat user guide
3. Membuat laporan

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi CPU monitoring

4.1.1 Fungsi Get CPU Usage Overall

```
# cpu_monitor.py
import psutil


def get_cpu_usage(interval=1):
    """
    Mengambil CPU usage overall dalam persentase

    Args:
        interval (int): Interval waktu untuk sampling (detik)

    Returns:
        float: CPU usage percentage (0-100)
    """
    try:
        cpu_percent = psutil.cpu_percent(interval=interval)
        return cpu_percent
    except Exception as e:
        print(f"Error getting CPU usage: {e}")
        return 0.0
```

Penjelasan:

1. Fungsi psutil.cpu_percent() mengembalikan CPU usage dalam persentase
2. Parameter interval menentukan waktu sampling untuk menghitung usage
3. Interval minimal 0.1 detik untuk hasil yang akurat
4. Return value 0-100 (float)

4.1.2 Fungsi Get CPU Usage Per-Core

```
def get_cpu_per_core(interval=1):
    """
    Mengambil CPU usage untuk setiap core

    Args:
        interval (int): Interval waktu untuk sampling (detik)

    Returns:
        list: List CPU usage percentage untuk setiap core
    """
    try:
        cpu_per_core = psutil.cpu_percent(interval=interval,
percpu=True)
        return cpu_per_core
    except Exception as e:
        print(f"Error getting CPU per core: {e}")
return []
```

Penjelasan:

a) Parameter percpu=True mengembalikan list usage untuk setiap logical processor
b) Berguna untuk monitoring beban kerja di multi-core system
c) Return list dengan panjang sesuai jumlah logical cores

4.1.3 Fungsi Get CPU Info

```
def get_cpu_info():
    """
    Mengambil informasi detail CPU

    Returns:
        dict: Dictionary berisi informasi CPU
    """
    try:
        cpu_info = {
            'physical_cores': psutil.cpu_count(logical=False),
            'logical_cores': psutil.cpu_count(logical=True),
            'max_frequency': psutil.cpu_freq().max if
psutil.cpu_freq() else 0,
            'min_frequency': psutil.cpu_freq().min if
psutil.cpu_freq() else 0,
            'current_frequency': psutil.cpu_freq().current if
psutil.cpu_freq() else 0
```

```
        }

        return cpu_info

    except Exception as e:

        print(f"Error getting CPU info: {e}")

        return {}
```

Penjelasan:

a) cpu_count(logical=False): Jumlah physical cores
b) cpu_count(logical=True): Jumlah logical cores (termasuk hyperthreading)
c) cpu_freq(): Frekuensi CPU dalam MHz
d) Conditional check untuk menghindari error jika data tidak tersedia

4.1.4 Fungsi Display CPU Info

```
def display_cpu_info(interval=1):

    """

    Menampilkan informasi CPU secara formatted



    Args:

        interval (int): Interval untuk sampling CPU usage

    """

    print("=" * 70)

    print("CPU MONITORING")

    print("=" * 70)
```

```
    # Get CPU info
    cpu_info = get_cpu_info()
    print(f"Physical Cores: {cpu_info.get('physical_cores', 'N/A')}")
    print(f"Logical Cores: {cpu_info.get('logical_cores', 'N/A')}")

    # Get CPU frequency
    if cpu_info.get('current_frequency', 0) > 0:
        print(f"Current Frequency: {cpu_info['current_frequency']:.2f}
MHz")
        print(f"Min Frequency: {cpu_info['min_frequency']:.2f} MHz")
        print(f"Max Frequency: {cpu_info['max_frequency']:.2f} MHz")

    print("-" * 70)

    # Overall CPU usage
    cpu_usage = get_cpu_usage(interval)
    print(f"Overall CPU Usage: {cpu_usage:.2f}%")
    print(create_progress_bar(cpu_usage, 50))
```

```
    print("-" * 70)



    # Per-core CPU usage

    print("CPU Usage Per Core:")

    cpu_per_core = get_cpu_per_core(interval)

    for i, usage in enumerate(cpu_per_core):

        print(f"  Core {i}: {usage:5.1f}% {create_progress_bar(usage,
30)}")



    print("=" * 70)
```

4.2 Implementasi RAM Monitoring

4.2.1 Fungsi Get Memory Info

```
# ram_monitor.py

import psutil



def get_memory_info():

    """

    Mengambil informasi memory system



    Returns:
```

```
        dict: Dictionary berisi informasi memory
    """
    try:
        memory = psutil.virtual_memory()
        memory_info = {
            'total': memory.total,
            'available': memory.available,
            'used': memory.used,
            'free': memory.free,
            'percent': memory.percent,
            'buffers': memory.buffers if hasattr(memory, 'buffers') else
0,
            'cached': memory.cached if hasattr(memory, 'cached') else 0
        }
        return memory_info
    except Exception as e:
        print(f"Error getting memory info: {e}")
        return {}
```

Penjelasan:

a) virtual_memory() mengembalikan statistik memory system
b) total: Total physical memory
c) available: Memory yang tersedia untuk allocate
d) used: Memory yang sedang digunakan
e) percent: Persentase memory yang digunakan
f) buffers dan cached: Memory yang digunakan untuk buffer dan cache (Linux)

4.2.2 Fungsi Get Swap INFO

```
def get_swap_info():
    """
    Mengambil informasi swap memory

    Returns:
        dict: Dictionary berisi informasi swap
    """
    try:
        swap = psutil.swap_memory()
        swap_info = {
            'total': swap.total,
            'used': swap.used,
            'free': swap.free,
            'percent': swap.percent
```

```
        }

        return swap_info

    except Exception as e:

        print(f"Error getting swap info: {e}")

        return {}
```

Penjelasan:

a) swap_memory() mengembalikan statistik swap memory
b) Swap adalah virtual memory yang disimpan di disk
c) Digunakan ketika RAM fisik penuh

4.2.3 Fungsi Format Memory Size

```
def format_bytes(bytes_value):

    """

    Mengkonversi bytes ke format yang mudah dibaca



    Args:

        bytes_value (int): Nilai dalam bytes



    Returns:

        str: String formatted (e.g., "2.5 GB")

    """
```

```
    for unit in ['B', 'KB', 'MB', 'GB', 'TB']:
        if bytes_value < 1024.0:
            return f"{bytes_value:.2f} {unit}"
        bytes_value /= 1024.0
    return f"{bytes_value:.2f} PB"
```

Penjelasan:

a) Mengkonversi bytes ke satuan yang lebih mudah dibaca
b) Iterasi melalui unit: B → KB → MB → GB → TB
c) Divide by 1024 untuk setiap konversi

4.2.4 Fungsi Display Memory Info

```
def display_memory_info():
    """
    Menampilkan informasi memory secara formatted
    """
    print("\n" + "=" * 70)
    print("RAM MONITORING")
    print("=" * 70)
    # Get memory info
    mem_info = get_memory_info()
```

```
    if mem_info:
        print(f"Total Memory: {format_bytes(mem_info['total'])}")
        print(f"Available Memory:
{format_bytes(mem_info['available'])}")
        print(f"Used Memory: {format_bytes(mem_info['used'])}")
        print(f"Free Memory: {format_bytes(mem_info['free'])}")
        print(f"Memory Usage: {mem_info['percent']:.2f}%")
        print(create_progress_bar(mem_info['percent'], 50))

        if mem_info['buffers'] > 0 or mem_info['cached'] > 0:
            print(f"\nBuffers: {format_bytes(mem_info['buffers'])}")
            print(f"Cached: {format_bytes(mem_info['cached'])}")

    print("-" * 70)

    # Get swap info
    swap_info = get_swap_info()

    if swap_info and swap_info['total'] > 0:
        print("SWAP MEMORY")
```

```
        print(f"Total Swap: {format_bytes(swap_info['total'])}")
        print(f"Used Swap: {format_bytes(swap_info['used'])}")
        print(f"Free Swap: {format_bytes(swap_info['free'])}")
        print(f"Swap Usage: {swap_info['percent']:.2f}%")
        print(create_progress_bar(swap_info['percent'], 50))
    else:
        print("SWAP MEMORY: Not configured")

    print("=" * 70)
```

4.3 Implementasi Utility Functions

4.3.1 Progress Bar Function

```
# utils.py

def create_progress_bar(percentage, bar_length=50):
    """
    Membuat visual progress bar (tanpa emoji)

    Args:
        percentage (float): Nilai persentase (0-100)
```

```
        bar_length (int): Panjang progress bar


    Returns:
        str: String progress bar
    """
    filled_length = int(bar_length * percentage / 100)
    bar = '#' * filled_length + '-' * (bar_length - filled_length)
    return f"[{bar}] {percentage:.1f}%"
```

Penjelasan:
Menghitung bagian progress bar berdasarkan persentase
• # menunjukkan progres yang sudah terisi
• - menunjukkan progres yang belum terisi
• Menampilkan persentase dalam bentuk teks

4.3.2 Clear Screen Function

```
import os
def clear_screen():
    """
    Clear terminal screen
    """
    os.system('clear' if os.name != 'nt' else 'cls')
```

Penjelasan:

a) Clear screen untuk tampilan yang bersih
b) clear untuk Linux/Mac
c) cls untuk Windows

4.3.3 Get Timestamp Function

```
from datetime import datetime

def get_timestamp():

    """

    Get current timestamp



    Returns:

        str: Formatted timestamp

    """

    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

4.4 Implementasi Main Program

```
# monitor.py

import time

from cpu_monitor import display_cpu_info

from ram_monitor import display_memory_info

from utils import clear_screen, get_timestamp
```

```
def main():
    """
    Main program untuk monitoring resource
    """
    update_interval = 2  # Update setiap 2 detik

    print("Resource Monitoring System")
    print("Press Ctrl+C to exit")
    time.sleep(2)

    try:
        while True:
            clear_screen()

            # Display header
            print("=" * 70)
            print(f"RESOURCE MONITORING SYSTEM - {get_timestamp()}")
            print("=" * 70)

            # Display CPU info
```

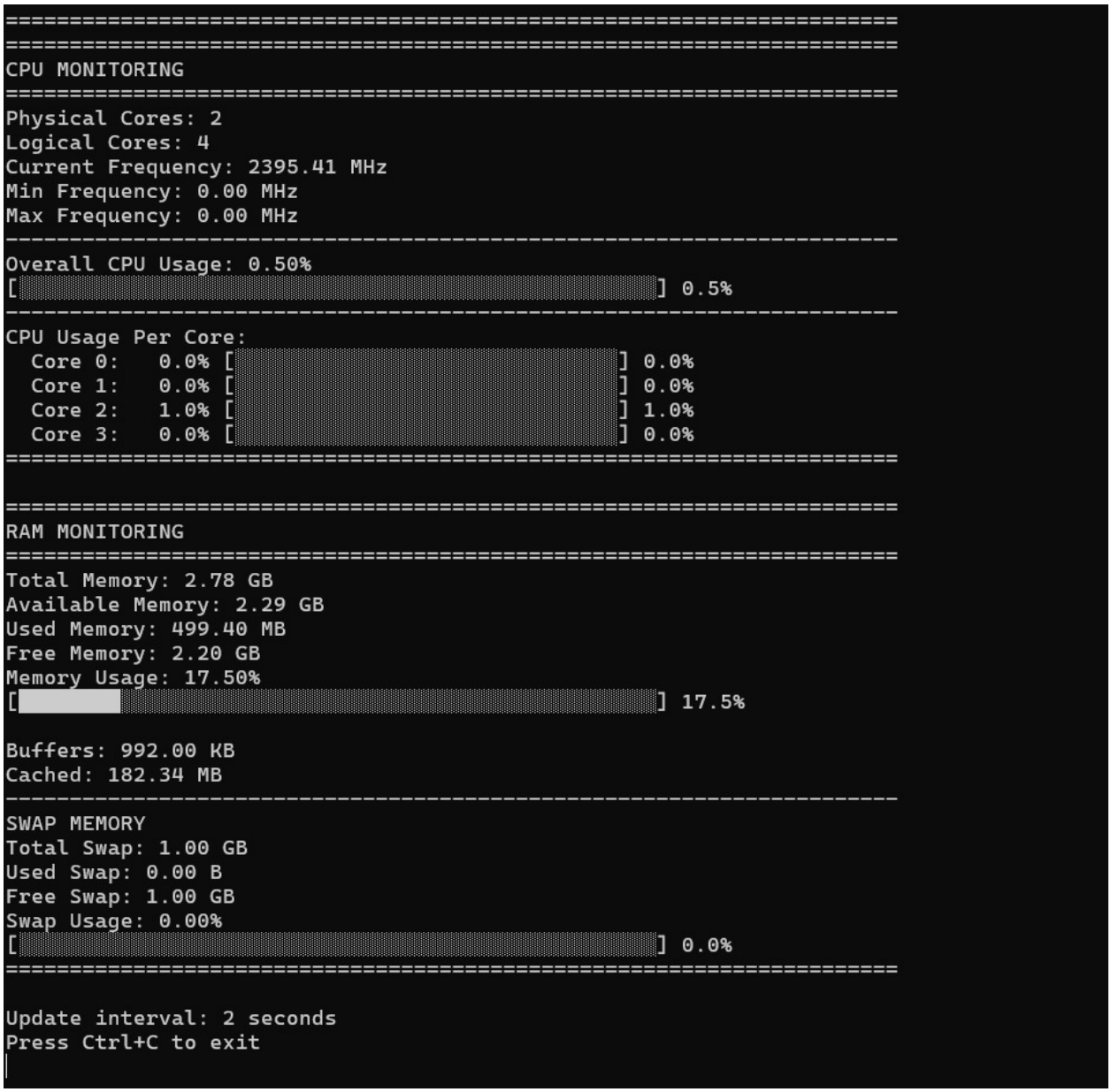
```
            display_cpu_info(interval=1)

            # Display Memory info
            display_memory_info()

            # Display footer
            print(f"\nUpdate interval: {update_interval} seconds")
            print("Press Ctrl+C to exit")

            # Wait before next update
            time.sleep(update_interval)

    except KeyboardInterrupt:
        print("\n\nMonitoring stopped by user.")
        print("Thank you for using Resource Monitoring System!")

if __name__ == "__main__":
    main()
```

Penjelasan:

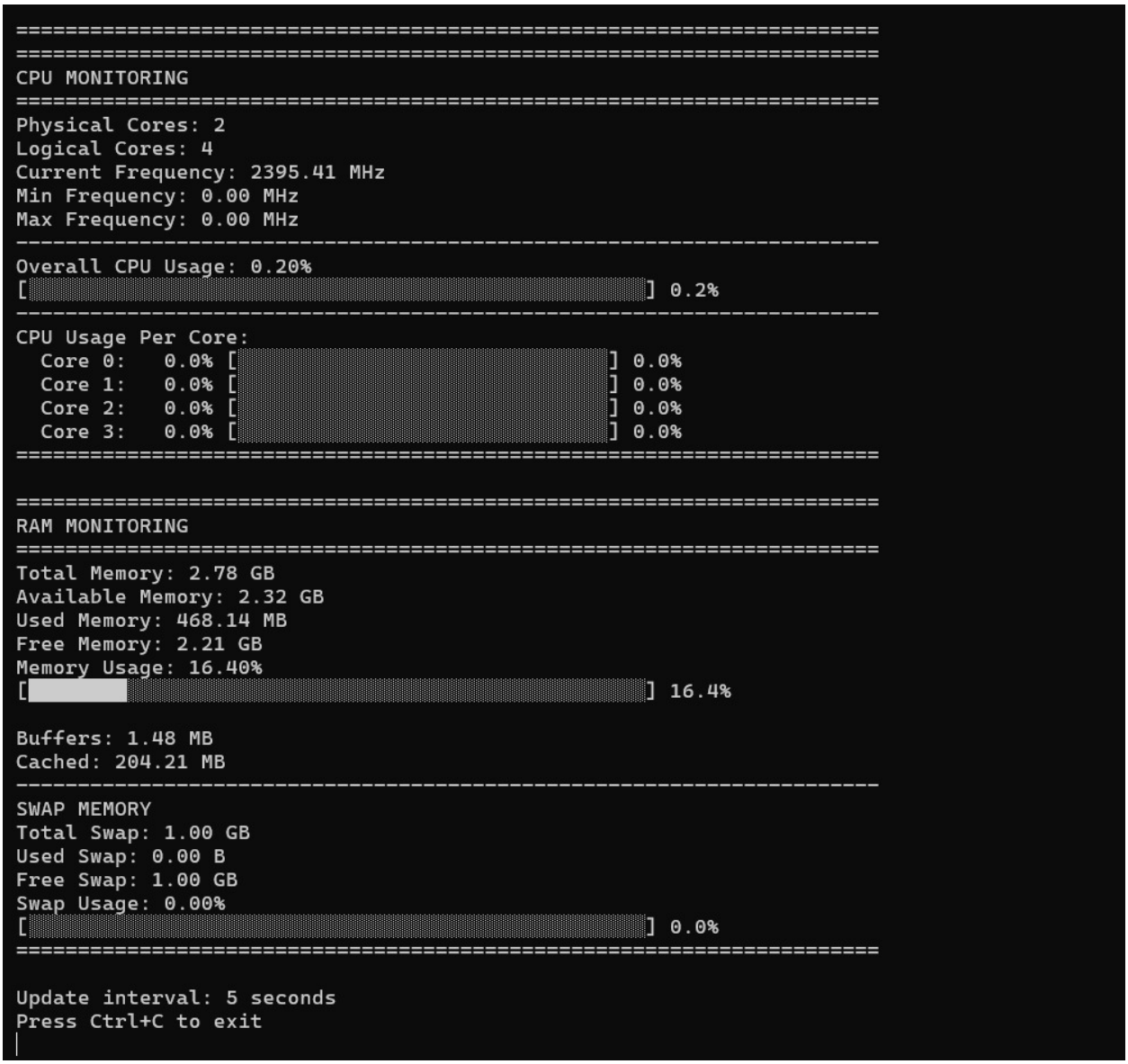
a) Infinite loop dengan while True
b) Clear screen setiap update untuk tampilan bersih
c) Display timestamp untuk tracking
d) KeyboardInterrupt untuk graceful exit dengan Ctrl+C
e) Sleep interval dapat dikonfigurasi

4.5 Hasil Testing

Sebelum di ubah waktu restart cek waktu monitoring nya



Waktu kami menguji kembali dengan mengubah waktu restart monitor nya, dari 2 detik menjadi 5 detik sehingga sistem akan restart setiap 5 detik.



BAB V

KESIMPULAN

Berdasarkan pelaksanaan implementasi pada minggu ke-13, dapat disimpulkan bahwa:

1. Implementasi monitoring CPU berhasil dilakukan dengan lengkap, mencakup overall CPU usage, per-core CPU usage, CPU frequency, dan informasi CPU lainnya. Semua data dapat diambil dan ditampilkan dengan akurat menggunakan library psutil.
2. Implementasi monitoring RAM berhasil dilakukan dengan komprehensif, mencakup total memory, used memory, available memory, buffer/cache, dan swap memory. Format data dalam bytes berhasil dikonversi ke satuan yang lebih mudah dibaca (GB, MB, KB).

Mekanisme auto-update berfungsi dengan baik dengan interval yang dapat di