Nama : Muhammad Andes
Nim : 3202402021
Judul : Tugas 1

Bagian 1: Konsep Dasar Manajemen Proses

1. Jelaskan dengan bahasa Anda sendiri apa yang dimaksud dengan Process Control Block (PCB) dan mengapa komponen ini sangat kritis bagi Sistem Operasi?

   Process Control Block (PCB) itu ibarat catatan penting milik sistem operasi tentang sebuah proses. Setiap kali kita menjalankan program (misalnya buka YouTube, buka Word, atau main game), sistem operasi bakal bikin satu "lembar catatan" khusus untuk program itu. Nah, lembar catatan itulah yang disebut PCB.

   Di catatan ini tersimpan semua informasi yang dibutuhkan biar program bisa jalan dengan benar, misalnya:

   - Posisi terakhir program lagi ngerjain apa.
   - Data tentang memori yang dipakai.
   - File apa aja yang sedang dibuka.
   - Urutan siapa duluan yang boleh jalan kalau ada banyak program.

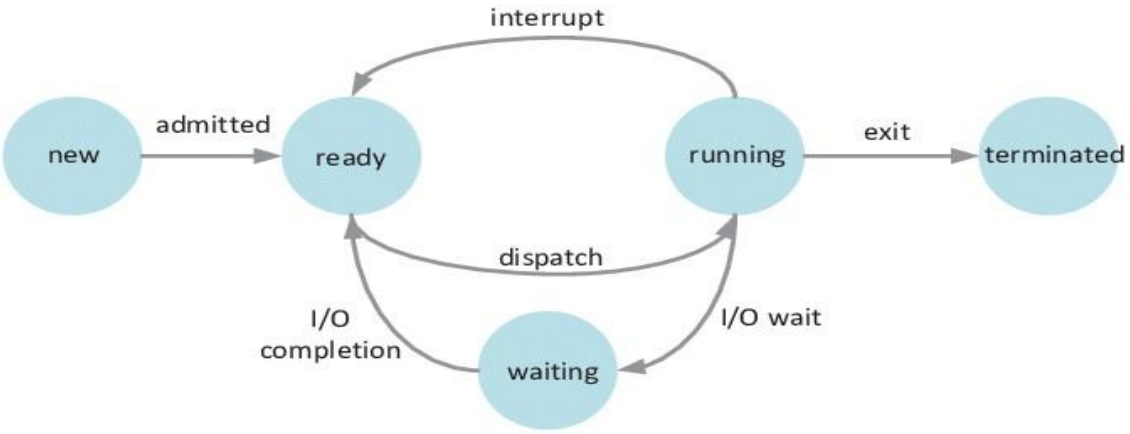
   Kenapa penting? Karena sistem operasi itu tugasnya mirip seperti manajer di kantor yang ngurus banyak karyawan sekaligus. Kalau manajer nggak punya catatan siapa karyawan mana, lagi ngerjain tugas apa, dan kapan harus lanjut, pasti kacau—kerjaan bisa ketukar atau ada yang hilang.

   Dengan adanya PCB, sistem operasi bisa:

   1. Menghentikan sementara program lalu melanjutkan lagi tanpa harus mulai dari awal.
   2. Ngatur banyak program sekaligus (multitasking) tanpa bikin mereka saling rebutan.
   3. Menjamin ketertiban supaya semua program dapat giliran dan nggak bentrok.

   Singkatnya, PCB itu catatan rahasia yang bikin komputer bisa kerja rapi dan teratur, meski ada banyak program jalan berbarengan.

2. Gambarkan diagram state (lifecycle) sebuah proses dan jelaskan secara singkat setiap transisi dari satu state ke state lainnya (contoh: dari new ke ready).



	Penjelasan
New → Ready	Proses sudah dibuat dan siap masuk ke antrian untuk dijalankan.
Ready → Running	Proses dipilih oleh CPU scheduler dan mulai dijalankan.
Running → Waiting	Proses membutuhkan input/output, jadi harus menunggu dulu.
Waiting → Ready	Setelah input/output selesai, proses siap dijalankan lagi.
Running → Ready	Proses dihentikan sementara untuk memberi giliran proses lain (preemption).
Running→Terminated	Proses selesai atau dihentikan secara paksa.

3. Apa perbedaan mendasar antara proses (process) dan thread?
    1. Proses (Process)
        - Proses itu ibarat sebuah rumah.
        - Di dalam rumah ada ruangan-ruangan (memori, data, kode, file yang dibuka).
        - Setiap rumah berdiri sendiri, punya alamat sendiri, dan tidak bisa langsung ngambil barang dari rumah lain tanpa izin.
        - Jadi, proses adalah program yang sedang berjalan dan punya ruang memori sendiri.
    2. Thread
        - Thread itu ibarat orang-orang yang tinggal di dalam rumah tersebut.
        - Semua orang (thread) tinggal di rumah yang sama (satu proses), jadi mereka bisa saling berbagi dapur, ruang tamu, atau listrik (alias berbagi memori dan resource).
        - Tapi tiap orang (thread) bisa punya aktivitas masing-masing: ada yang masak, ada yang belajar, ada yang tidur.
        - Jadi, thread adalah jalur eksekusi yang lebih ringan, hidup di dalam sebuah proses, dan berbagi sumber daya proses itu.

    Perbedaan mendasar

    1. Isolasi memori
        - Proses: punya memori sendiri, tidak otomatis bisa mengakses memori proses lain.
        - Thread: berbagi memori dengan thread lain dalam satu proses.

2. Bobot (resource)
   - Proses: lebih “berat”, karena harus punya memori, resource, dan ruang sendiri.
   - Thread: lebih “ringan”, karena cuma numpang di dalam proses.
3. Komunikasi
   - Proses: komunikasi antar proses (IPC) lebih ribet karena butuh mekanisme khusus (misalnya pipe, socket).
   - Thread: komunikasi gampang, cukup baca/tulis di memori bersama.
4. Dampak error
   - Proses: kalau satu proses crash, proses lain tetap bisa jalan.
   - Thread: kalau satu thread error (misalnya nulis sembarangan ke memori), bisa bikin seluruh proses ikut rusak.

Bagian 2: Analisis Algoritma Penjadwalan

1. Data awal

Process	Burst Time (ms)	Priority
P1	8	3
P2	6	1
P3	4	2
P4	2	4

Semua arrival time (AT) = 0.

2. Tentukan urutan eksekusi

Urutkan berdasarkan **priority (kecil → besar)** karena semua tiba di t=0:

- Priority 1 → P2
- Priority 2 → P3
- Priority 3 → P1
- Priority 4 → P4

Jadi urutan eksekusi: **P2 → P3 → P1 → P4**

3. Gantt chart (timeline)

| P2 | P3 | P1 | P4 |

0 6 10 18 20 (waktu dalam ms)

Penjelasan singkat: P2 dari 0–6, P3 dari 6–10, P1 dari 10–18, P4 dari 18–20.

4. Hitung Completion Time (CT) langkah demi langkah

Mulai t = 0.

- Untuk **P2**:
  CT_2 = start(=0) + BT_2 = 0 + 6 = 6.
- Untuk **P3**:
  CT_3 = CT_2 + BT_3 = 6 + 4 = 10.
  (Langkah: 6 + 4 = 10)
- Untuk **P1**:
  CT_1 = CT_3 + BT_1 = 10 + 8 = 18.
  (Langkah: 10 + 8 = 18)
- Untuk **P4**:
  CT_4 = CT_1 + BT_4 = 18 + 2 = 20.
  (Langkah: 18 + 2 = 20)

5. Hitung Turnaround Time (TAT) = CT - AT (AT = 0)

- TAT_2 = 6 - 0 = 6.
- TAT_3 = 10 - 0 = 10.
- TAT_1 = 18 - 0 = 18.
- TAT_4 = 20 - 0 = 20.

6. Hitung Waiting Time (WT) = TAT - BT

- WT_2 = TAT_2 - BT_2 = 6 - 6 = 0.
- WT_3 = 10 - 4 = 6.
- WT_1 = 18 - 8 = 10.
- WT_4 = 20 - 2 = 18.

7. Jumlah dan rata-rata (langkah aritmetika terperinci)

Total Turnaround

TAT total = 6 + 10 + 18 + 20.
Langkah: 6 + 10 = 16 → 16 + 18 = 34 → 34 + 20 = 54.
Total TAT = **54**.

Rata-rata TAT = 54 ÷ 4.

Langkah bagi: 4 × 13 = 52 (sisa 2) → sisa 2/4 = 0.5 → 13 + 0.5 = **13.5 ms**.

Total Waiting

WT total = 0 + 6 + 10 + 18.

Langkah: 0 + 6 = 6 → 6 + 10 = 16 → 16 + 18 = 34.

Total WT = **34**.

Rata-rata WT = 34 ÷ 4.

Langkah bagi: 4 × 8 = 32 (sisa 2) → sisa 2/4 = 0.5 → 8 + 0.5 = **8.5 ms**.

8. Ringkasan hasil (tabel)

Process	BT	AT	CT	TAT = CT−AT	WT = TAT−BT
P2	6	0	6	6	0
P3	4	0	10	10	6
P1	8	0	18	18	10
P4	2	0	20	20	18

- **Total TAT** = 54 → **Avg Turnaround = 13.5 ms**
- **Total WT** = 34 → **Avg Waiting = 8.5 ms**

Bagian 3: Simulasi dan Pemikiran Kritis

1. Scenario: Sebuah sistem operasi modern menggunakan algoritma Completely Fair Scheduler (CFS) yang ada pada kernel Linux. Jelaskan konsep utama bagaimana CFS bekerja dan bagaimana ia berbeda dengan algoritma Round Robin klasik?

   Algoritma **CFS (Completely Fair Scheduler)** di Linux dasarnya ingin membuat pembagian CPU itu “adil” untuk semua proses. Ibaratnya semua proses dianggap punya antrian panjang, lalu CPU dibagi seperti kue: siapa yang sudah dapat bagian lebih banyak, giliran berikutnya dia harus menunggu lebih lama supaya proses lain yang belum kebagian bisa jalan.

   Bedanya dengan **Round Robin klasik** adalah:

   - Round Robin pakai **time slice tetap** (misalnya 10 ms per proses). Jadi proses jalan 10 ms, lalu digilir ke proses berikutnya, entah dia baru dapat CPU atau sudah sering.

Hasilnya kadang proses yang “ringan” tetap harus antre sama lama dengan proses yang “berat”.

- CFS lebih fleksibel, pakai konsep **virtual runtime**. Jadi CPU diberikan ke proses yang paling “sedikit” menggunakan CPU sejauh ini, bukan sekadar gilir paksa. Jadi jadwalnya lebih seimbang dan terasa lebih “fair”.

Simpelnya: Round Robin = gilir piring nasi satu-satu tanpa lihat siapa sudah kenyang, CFS = lihat siapa yang paling sedikit makan, dia yang duluan dikasih.

2. Scenario: Sebuah proses menghabiskan sebagian besar waktunya pada state Waiting (I/O Bound). Menurut Anda, apakah meningkatkan kecepatan CPU akan secara signifikan mempercepat eksekusi proses tersebut? Jelaskan alasan Anda dengan menghubungkannya pada konsep CPU-bound vs. I/O-bound processes.

   Kalau ada proses yang kebanyakan waktunya habis di **waiting (nunggu I/O)**, misalnya baca file, tunggu jaringan, atau tunggu user input, maka **nambah kecepatan CPU hampir gak ngaruh**. Soalnya masalah utama dia bukan CPU lambat, tapi I/O yang lama.

   Analogi gampangnya: bayangkan kamu pesan makanan di restoran. Waktu tunggu lama karena dapurnya masak lama (I/O). Walaupun kamu bisa makan super cepat (CPU lebih cepat), itu gak mempercepat makanan keluar dari dapur. Jadi proses tetap akan bottleneck di I/O.

   Jadi peningkatan kecepatan CPU lebih terasa manfaatnya untuk proses **CPU-bound** (yang kerjanya ngitung terus, seperti kompresi video, rendering 3D, atau machine learning). Kalau prosesnya **I/O-bound**, yang lebih membantu justru mempercepat storage, jaringan, atau device I/O.

3. Pemikiran Kritis: Dalam konteks penjadwalan pada lingkungan multi-core processor, tantangan apa saja yang muncul yang tidak ada pada penjadwalan single-core? (Sebutkan minimal 2 dan jelaskan).

   Kalau pakai prosesor single-core, gampang: satu CPU, satu antrian, tinggal pilih siapa yang jalan. Tapi begitu **multi-core**, muncul tantangan baru:

   1. **Load Balancing (pembagian beban)**

      - Proses harus dibagi rata ke core-core supaya gak ada core yang sibuk banget sementara core lain nganggur.

- Tantangannya adalah memutuskan kapan harus mindahin proses dari core yang sibuk ke core yang lebih kosong, tanpa bikin overhead terlalu besar.

2. **Cache Affinity (lokalitas data di cache)**

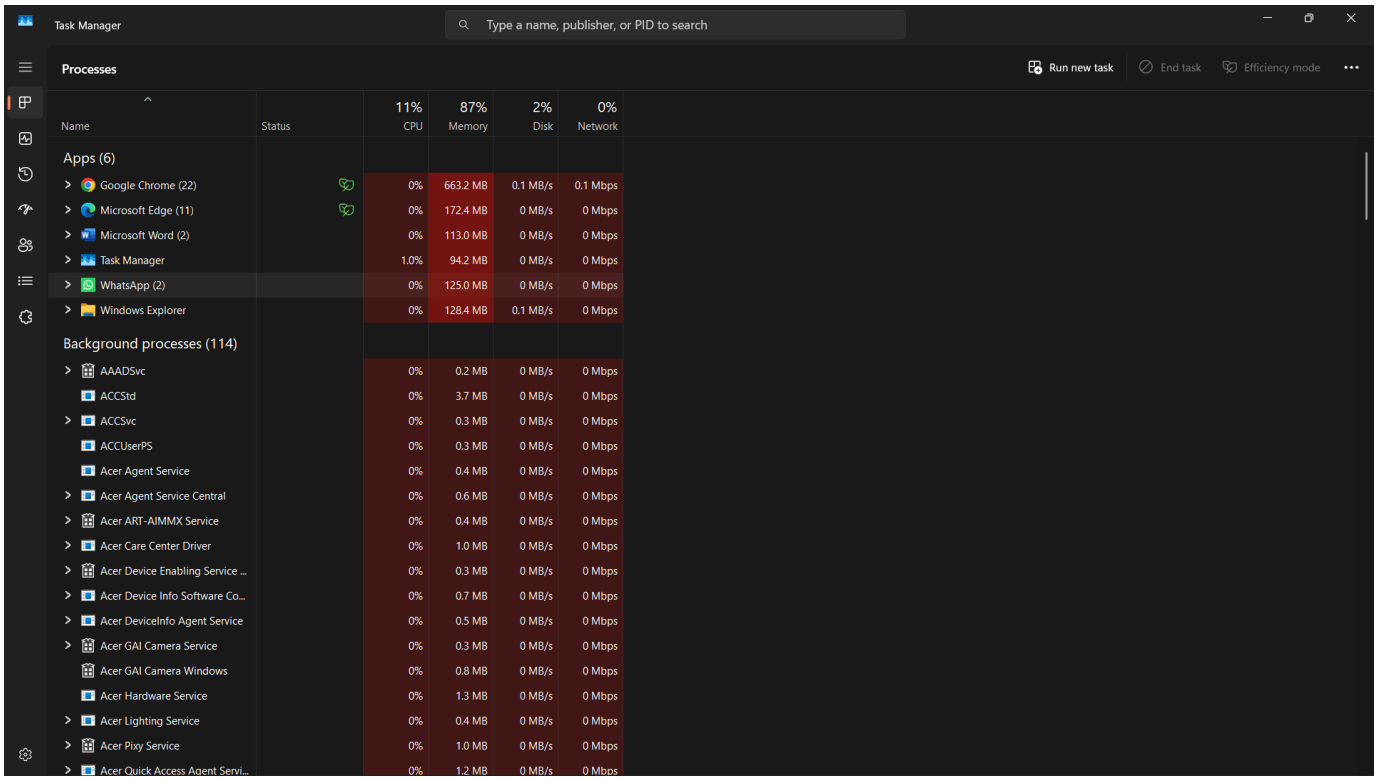
   Kalau proses pindah dari core A ke core B, data yang sebelumnya sudah ada di cache core A harus dimuat ulang ke cache core B. Ini bikin performa bisa drop. Jadi scheduler harus pintar: lebih baik proses tetap di core yang sama (kalau bisa), walaupun load balancingnya agak timpang sedikit.

3. (tambahan bonus) **Sinkronisasi antar-core**

   Kadang proses yang jalan di core berbeda harus saling komunikasi. Kalau koordinasinya gak diatur baik, bisa muncul deadlock, race condition, atau overhead tinggi karena sering “saling tunggu”.

Bagian 4: Aplikasi Praktis

Taks Manager



Beberapa kolom penting yang muncul di tampilan itu:

1. **Name**

   - Menunjukkan nama aplikasi atau proses yang sedang berjalan.

- Misalnya *Google Chrome, Microsoft Edge, WhatsApp*. Ini membantu user mengenali aplikasi apa yang memakai resource.

2. **Status**
    - Menampilkan kondisi proses saat ini. Contohnya bisa ada simbol *suspended* (ditangguhkan) ketika Windows menghentikan sementara proses agar hemat resource. Kalau kosong, berarti proses berjalan normal.
3. **CPU (%)**
    - Persentase penggunaan CPU oleh proses tersebut dari total kapasitas CPU yang ada.
    - Contoh: "1%" berarti proses hanya memakai 1% dari kemampuan prosesor. Kalau nilainya tinggi (misalnya >80%), biasanya proses itu sangat berat.
4. **Memory**
    - Menunjukkan berapa besar RAM yang sedang digunakan proses tersebut.
    - Ditampilkan dalam satuan MB/GB. Contoh: Google Chrome pakai 663.2 MB RAM.
5. **Disk**
    - Menampilkan aktivitas baca/tulis ke storage (HDD/SSD).
    - Kalau proses banyak membaca/menulis data ke disk, kolom ini akan menunjukkan kecepatan transfer (MB/s).
6. **Network** (tambahan)
    - Menunjukkan seberapa besar proses tersebut memakai jaringan (upload/download).
    - Ditampilkan dalam Mbps. Contoh: WhatsApp sedang 0 Mbps berarti tidak aktif pakai internet pada saat itu.

Saya ambil contoh **Google Chrome (22)**:

- **Nama**: Google Chrome
- **CPU**: 0% (artinya saat itu Chrome sedang tidak aktif menghitung atau idle).
- **Memory**: 663.2 MB → cukup besar, karena Chrome terkenal banyak pakai RAM (apalagi dengan banyak tab).

- **Disk**: 0.1 MB/s → ada sedikit aktivitas baca/tulis ke penyimpanan.
- **Network**: 0.1 Mbps → artinya Chrome sedang melakukan sedikit komunikasi data (mungkin sync atau background refresh).

State-nya bisa dikatakan **Running tapi idle**. Kenapa?

- Karena prosesnya masih hidup, tidak *suspended*, tapi CPU usage = 0%. Jadi proses tidak sedang intensif menghitung, hanya menunggu event (misalnya user membuka tab baru atau menunggu data jaringan). Dalam konsep OS, ini lebih mirip ke **Waiting/Sleeping** (menunggu input I/O atau event).

Jadi ringkasnya:

- 5 kolom yang penting: **Name, Status, CPU, Memory, Disk, Network**.
- Contoh Chrome: saat itu **CPU usage = 0%**, RAM besar (663.2 MB), sedang dalam kondisi **waiting** (tidak aktif menghitung, hanya menunggu event I/O).