MODUL PRAKTIKUM 06

TOPIK: SIMULASI PENGHINDARAN DEADLOCK (BANKER'S ALGORITHM)

Mata Kuliah: Sistem Operasi (INF-201) – Alokasi Waktu: 170 Menit

Program Studi Informatika – Fakultas Teknik – Universitas Bengkulu

---

1. CAPAIAN PEMBELAJARAN MODUL (SUB-CPMK 6)

Setelah menyelesaikan Modul Praktikum 06 ini, mahasiswa diharapkan mampu:

1. Menghitung matriks kebutuhan sumber daya Need = Max − Allocation.
2. Memprogram alur simulasi Banker's Algorithm untuk mendeteksi *Safe State*.
3. Memverifikasi keberadaan urutan aman eksekusi proses (*Safe Sequence*).

2. PRASYARAT TUGAS MEMBACA

Sebelum memulai percobaan laboratorium ini, mahasiswa **wajib** telah membaca bab referensi berikut:

- **[OSTEP]** Chapter 32: *Common Concurrency Bugs (Deadlock)* (hal. 379–394).
- **[xv6 RISC-V]** Chapter 6: *Locking: Deadlock Avoidance* (hal. 66–68).

3. LANDASAN TEORI RINGKAS

3.1 4 Syarat Mutlak Deadlock (Coffman Conditions)

Deadlock terjadi jika dan hanya jika 4 kondisi berikut terpenuhi secara bersamaan: (1) Mutual Exclusion; (2) Hold and Wait; (3) No Preemption; dan (4) Circular Wait.

3.2 Banker's Algorithm: Inisialisasi Matriks

Diberikan n proses dan m tipe sumber daya:

- Allocation$[i][j]$: Jumlah sumber daya j yang sedang dialokasikan ke proses i.
- Max$[i][j]$: Jumlah maksimum sumber daya j yang dibutuhkan proses i.
- Need$[i][j]$ = Max$[i][j]$ − Allocation$[i][j]$: Sisa sumber daya j yang dibutuhkan proses i.

4. KODE ACUAN UTUH (EXECUTABLE C LISTING: `banker.c`)

```c
#include <stdio.h>
#include <stdbool.h>

#define P 5 // Jumlah Proses
#define R 3 // Jumlah Tipe Sumber Daya

int main() {
    int alloc[P][R] = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}};
    int max[P][R]   = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};
```

```c
    int avail[R]    = {3, 3, 2};

    int need[P][R];
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    bool finish[P] = {false};
    int safe_seq[P];
    int work[R];
    for (int i = 0; i < R; i++) work[i] = avail[i];

    int count = 0;
    while (count < P) {
        bool found = false;
        for (int p = 0; p < P; p++) {
            if (!finish[p]) {
                bool possible = true;
                for (int j = 0; j < R; j++) {
                    if (need[p][j] > work[j]) { possible = false; break; }
                }
                if (possible) {
                    for (int k = 0; k < R; k++) work[k] += alloc[p][k];
                    safe_seq[count++] = p;
                    finish[p] = true;
                    found = true;
                }
            }
        }
        if (!found) { printf("Sistem berada dalam UNSAFE STATE!\n"); return
            0; }
    }

    printf("Sistem SAFE! Safe Sequence: ");
    for (int i = 0; i < P; i++) printf("P%d ", safe_seq[i]);
    printf("\n");
    return 0;
}
```

Listing 1: Program Simulasi Banker's Algorithm C (`banker.c`)

5. PROSEDUR LANGKAH KERJA PERCOBAAN

1. Kompilasi dan jalankan `banker.c`: `gcc banker.c -o banker && ./banker`.
2. Uji dengan mengubah nilai alokasi P_0 agar melebihi sumber daya dan amati deteksi *Unsafe State*.

6. TUGAS PRAKTIKUM & PERTANYAAN ANALISIS LAPORAN

1. **Analisis Lock Ordering**: Mengapa mematok urutan penguncian (*Lock Ordering*) yang konsisten dapat menggagalkan syarat *Circular Wait*?