Badrus Zaman

# Synchronizations

# Agenda

- Latar Belakang
- Masalah Critical Section
- Semaphores

# Istilah Penting

- Atomic Operation
- Critical Section
- Deadlock
- Livelock
- Mutual Exclusion
- Race Condition
- Starvation

# Overview (1)

- *Proteksi OS:*
  - *Independent* process tidak terpengaruh atau dapat mempengaruhi eksekusi/data proses lain.
- "Concurrent Process"
  - OS: mampu membuat banyak proses pada satu saat
  - Proses-proses bekerja-sama: sharing data, pembagian task, passing informasi dll
  - Proses => mempengaruhi proses lain dalam menggunakan data/informasi yang sengaja di-"share"
- *Cooperating* process – sekumpulan proses yang dirancang untuk saling bekerja-sama untuk mengerjakan task tertentu.

# Overview (2)

- Keuntungan kerja-sama antar proses
  - Information sharing
    - File, DB → digunakan bersama
  - Computation speed-up
    - parallel proses
  - Modularity
    - aplikasi besar → dipartisi dalam banyak proses.
  - Convenience
    - kumpulan proses → tipikal lingkungan kerja.
- "Cooperating Process"
  - Bagaimana koordinasi antar proses? Akses/Update data
  - Tujuan program/task: integritas, konsistensi data dapat dijamin

# Latar Belakang

- Concurrent access to shared data may result in data inconsistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
  - Program/task-task dapat menghasilkan operasi yang benar setiap waktu
  - Deterministik: untuk input yang sama hasil harus sama (sesuai dengan logika/algroitma program).
- Suppose that we wanted to provide a solution to the consumer-producer problem that fills **all** the buffers. We can do so by having an integer **count** that keeps track of the number of full buffers. Initially, count is set to 0. It is incremented by the producer after it produces a new buffer and is decremented by the consumer after it consumes a buffer.

- Contoh: Producer – Consumer
  - Dua proses
    - producer => menghasilkan informasi;
    - consumer => menggunakan informasi
  - Sharing informasi:
    - buffer => tempat penyimpanan data
    - *unbounded-buffer*, penempatan tidak pada limit praktis dari ukuran buffer
    - *bounded-buffer* diasmumsikan terdapat ukuran buffer yang tetap

# Bounded Buffer

- Implementasi buffer:
  - IPC: komunikasi antar proses melalui messages membaca/menulis buffer
  - Shared memory: programmer secara eksplisit melakukan "deklarasi" data yang dapat diakses secara bersama.
  - Buffer dengan ukuran n => mampu menampung n data
    - Producer mengisi data buffer => increment "counter" (jumlah data)
    - Consumer mengambil data buffer => decrement "counter"
    - Buffer, "counter" => shared data (update oleh 2 proses)

# Bounded Buffer

- Apakah terdapat jaminan operasi akan benar jika berjalan concurrent?
- Misalkan: counter = 5
  - Producer: counter = counter + 1;
  - Consumer: counter = counter - 1;
  - Nilai akhir dari counter?

# Producer

```
while (true) {
        /*  produce an item and put in nextProduced
    */
        while (count == BUFFER_SIZE)
                ; // do nothing
            buffer [in] = nextProduced;
            in = (in + 1) % BUFFER_SIZE;
            count++;
}
```

# Consumer

```
while (true)  {
        while (count == 0)
              ; // do nothing
              nextConsumed =  buffer[out];
               out = (out + 1) % BUFFER_SIZE;
                count--;
                    /*  consume the item in nextConsumed
}
```

- Operasi concurrent P & C =>
  - Operasi dari high level language => sekumpulan instruksi mesin:
- **"increment counter"**

  **Load Reg1, Counter**

  **Add Reg1, 1**

  **Store Counter, Reg1**
- **"decrement counter"**

  **Load Reg2, Counter**

  **Subtract Reg2, 1**

  **Store Counter, Reg2**

- Eksekusi P & C tergantung scheduler (dapat gantian)
- Consider this execution interleaving with "counter = 5" initially:
  - T0: Producer : Load Reg1, Counter (Reg1 = 5)
  - T1: Producer : Add Reg1, 1 (Reg1 = 6)
  - T2: Consumer: Loag Reg2, Counter (Reg2 = 5)
  - T3: Consumer: Subtract Reg2, 1 (Reg2 = 4)
  - T4: Producer: Store Counter, Reg1 (Counter = 6)
  - T5: Consumer: Store Counter, Reg2 (Counter = 4)

# Race Condition

- Concurrent C & P
  - Shared data "counter" dapat berakhir dengan nilai: 4, atau 5, atau 6
  - Hasilnya dapat salah dan tidak konsisten
- Race Condition:
  - Keadaan dimana lebih dari satu proses meng-update data secara "concurrent" dan hasilnya sangat bergantung dari urutan proses mendapat jatah CPU (run)
  - Hasilnya tidak menentu dan tidak selalu benar
  - Mencegah race condition: sinkronisasi proses dalam meng-update shared data

# Sinkronisasi

- Sinkronisasi:
  - Koordinasi akses ke shared data, misalkan hanya satu proses yang dapat menggunakah shared var.
  - Contoh operasi terhadap var. "counter" harus dijamin di-eksekusi dalam satu kesatuan (atomik) :
    - *counter* := *counter* + 1;
    - *counter* := *counter* - 1;
  - Sinkronisasi merupakan "issue" penting dalam rancangan/implementasi OS (shared resources, data, dan multitasking).

# Masalah Critical Section

- n proses mencoba menggunakan shared data bersamaan
- Setiap proses mempunyai "code" yang mengakses/ manipulasi shared data tersebut => "critical section"
- Problem: Menjamin jika ada satu proses yang sedang "eksekusi" pada bagian "critical section" tidak ada proses lain yang diperbolehkan masuk ke "code" critical section dari proses tersebut.
- Structure of process *Pi*

```
repeat
   entry section
        critical section
   exit section
         reminder section
until false;
```

# Solution to Critical-Section Problem

- Ide :
  - Mencakup pemakaian secara "exclusive" dari shared variable tsb
  - Menjamin proses lain dapat menggunakan shared variable tsb
- Tiga Solusi "critical section problem":
  - **Mutual Exclusion** - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
  - **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
  - **Bounded Waiting** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

# Semaphore

- Prinsip bahwa dua proses atau lebih dapat bekerja sama dengan menggunakan penanda-penanda sederhana. Seperti proses dapat dipaksa berhenti pada suatu saat, sampai proses mendapatkan penanda tertentu itu. Sembarang kebutuhan koordinasi kompleks dapat dipenuhi dengan struktur penanda yang cocok untuk kebutuhan itu. Variabel khusus untuk penanda ini disebut semaphore.
- Semaphore mempunyai dua sifat, yaitu:
  1. Semaphore dapat diinisialisasi dengan nilai non-negatif.
  2. Terdapat dua operasi terhadap semaphore, yaitu Down dan Up.

# Classical Problem of Synchronizations

- Create an article (review) about classical problem of synchronizations.
- The review contain at least: **background**, **goals**, **problems**, and **solutions**.
- File type : pdf
- Distribution of task as follows:
  - **Number of Task** = (**Last 3 digit NIM/Username**) **MOD 3 + 1**
  - For MSU students use **AULA Username**

# Topics

1. Bounded-Buffer Problem (Producer-Consumer Problem)
2. Readers and Writers Problem
3. Dining-Philosophers Problem