PERTEMUAN 7:
SINKRONISASI

A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan mengenai sinkronisasi, Anda harus mampu:

1.1 Pengertian Sinkronisasi

1.2 Race Condition

1.3 Semaphore

B. URAIAN MATERI

Tujuan Pembelajaran 1.1:
Sinkronisasi

Sinkronisasi

Akses bebarengan untuk berbagi dua bersama dapat mengakibatkan inkosistensi data. Pemeliharaan konsistensi data memerlukan mekanisme untuk memastikan eksekusi dari proses kerjasama. Shared memory merupakan solusi ke masalah bounded-butter yang mengijinkan paling banyak n-1 materi dalam buffer pada waktu yang sama. Suatu solusi, jika semua N buffer digunakan tidaklah sederhana. Dimisalkan kita memdifikasi producer-consumer code dengan menambahkan suatu variable counter, dimulai dari 0 dan masing-masing waktu tambahan dari suatu item baru diberikan kepada buffer.

Race Condition

Race Condition adalah situasi di mana beberapa proses mengakses dan memanipulasi data bersama pada saat besamaan. Nilai akhir dari data bersama tersebut tergantung pada proses yang terakhir selesai. Untuk mencegah race condition, proses-proses yang berjalan besamaan harus di disinkronisasi.

Dalam beberapa sistem operasi, proses-proses yang berjalan bersamaan mungkin untuk membagi beberapa penyimpanan umum, masing-masing dapat melakukan proses baca (read) dan proses tulis (write). Penyimpanan bersama (shared storage) mungkin berada di memori utama atau berupa sebuah berkas bersama, lokasi dari memori bersama tidak merubah kealamian dari komunikasi atau masalah yang muncul. Untuk mengetahui bagaimana

komunikasi antar proses bekerja, mari kita simak sebuah contoh sederhana, sebuah print spooler. Ketika sebuah proses ingin mencetak sebuah berkas, proses tersebut memasukkan nama berkas ke dalam sebuah spooler direktori yang khusus. Proses yang lain, printer daemon, secara periodik memeriksa untuk mengetahui jika ada banyak berkas yang akan dicetak, dan jika ada berkas yang sudah dicetak dihilangkan nama berkasnya dari direktori.

Masalah Critical Section

Kunci untuk mencegah masalah ini dan di situasi yang lain yang melibatkan shared memori, shared berkas, and shared sumber daya yang lain adalah menemukan beberapa jalan untuk mencegah lebih dari satu proses untuk melakukan proses writing dan reading kepada shared data pada saat yang sama. Dengan kata lain kita memutuhkan mutual exclusion, sebuah jalan yang menjamin jika sebuah proses sedang menggunakan shared berkas, proses lain dikeluarkan dari pekerjaan yang sama. Kesulitan yang terjadi karena proses 2 mulai menggunakan variabel bersama sebelum proses 1 menyelesaikan tugasnya.

Masalah menghindari race conditions dapat juga diformulasikan secara abstrak. Bagian dari waktu, sebuah proses sedang sibuk melakukan perhitungan internal dan hal lain yang tidak menggiring ke kondisi race conditions. Bagaimana pun setiap kali sebuah proses mengakses shared memory atau shared berkas atau melakukan sesuatu yang kitis akan menggiring kepada race conditions. Bagian dari program dimana shaed memory diakses disebut Critical Section atau Critical Region.

Walau pun dapat mencegah race conditions, tapi tidak cukup untuk melakukan kerjasama antar proses secara pararel dengan baik dan efisien dalam menggunakan shared data. Kita butuh 4 kondisi agar menghasilkan solusi yang baik:

- Tidak ada dua proses secara bersamaan masuk ke dalam citical section.
- Tidak ada asumsi mengenai kecepatan atau jumlah cpu.
- Tidak ada proses yang berjalan di luar critical secion yang dapat mengeblok proses lain.
- Tidak ada proses yang menunggu selamamya untuk masuk critical section.

Critical Section adalah sebuah segmen kode di mana sebuah proses yang mana sumber daya bersama diakses. Terdiri dari:

Entry Section: kode yang digunakan untuk masuk ke dalam critical section

Critical Section: Kode di mana hanya ada satu proses yang dapat dieksekusi pada satu waktu

Exit Section: akhir dari critical section, mengizinkan proses lain
Remainder Section: kode istirahat setelah masuk ke critical section.

Solusi ke Masalah Critical-Section

Ada bebrapa Solusi untuk mengatasi masalah Critical Section, yaitu:

- Mutual exclution

  Jika proses pi sedang mengeksekusi critical section-nya maka tidak ada proses lain yang dapat mengeksekusi dalam critical section mereka.

- Progress

  Jika tidak ada proses yang sedang dieksekusi dalam critical section dan ada beberapa proses yang ingin masuk ke critical section mereka, maka pemilihan proses yang akan masuk ke critical section berikutnya tidak bias ditunda.

- Bounded Waiting

  Suatu keterikatan harus ada pada sejumlah proses yang diijinkan masuk ke critical section mereka, setelah adanya proses yang meminta masuk ke critical section dan sebelum permintaan itu diterima.

  a. Asumsikan bahwa tiap proses mengeksekusi pada nonzero speed.

  b. Tidak ada asumsi mengenai kecepatan relative dan n proses.

Cara-cara memecahkan masalah

- Hanya dua proses, Po dan P1
- Struktur umum dari proses adalah Pi (proses lain Pj)

Bakery Algorithm

Critical section untuk n proses:

a. Sebelum memasuki critical Section-nya, proses menerima nomor pemilik nomor terkecil memasuki critical section.

b. Jika proses Pi dan Pj menerima nomor yang sama, jika i < j, maka Pi dilayani duluan, lainnya Pj dilayani duluan (if i< j, then Pi is served first; else Pj is served first).

c. Skema penomoran selalu menghasilkan angka –angka yang disebutkan satu per satu, yaitu 1,2,3,3,3,3,4,5….

Semaphore

Semaphore adalah pendekatan yang diajukan oleh Djikstra, dengan 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:

i. Semaphore dapat diinisialisasi dengan nilai non-negatif.

ii. Terdapat dua operasi terhadap semaphore, yaitu Down dan Up. Usulan asli yang disampaikan Djikstra adalah operasi P dan V.

- Operasi Down

  Operasi ini menurunkan nilai semaphore, jika nilai semaphore menjadi non-positif maka proses yang mengeksekusinya diblocked. Operasi Down adalah atomic, tak dapat diinterupsi sebelaum diselesaikan.Emnurunkan nilai, memeriksa nilai, menempatkan proses pada antrian dan memblocked sebagai instruksi tunggal. Sejak dimulai, tak ada proses alain yang dapat mengakses semaphore sampai operasi selesai atau diblocked.

- Operasi Up

  Operasi Up menakkan nilai semaphore. Jika satu proses atau lebih diblocked pada semaphore itu tak dapat menyelesaikan operasi Down, maka salah satu dipilih oleh system dan menyelesaikan operasi Down-nya. Urutan proses yang dipilih tidak ditentukan oleh Djikstra, dapat dipilih secara acak. Adanya semaphore mempermudah persoalan mutual exclusion. Skema penelesaian mutual exclusion mempunyai bagan sebagai berikut:

  Sebelum masuk critical section, proses melakukan Down. Bila berhasil maka proses masuk ke critical section. Bila tidak berhasil maka proses di-blocked atas semaphore itu. Proses yang diblocked akan dapat melanjutkan kembali bila proses yang ada di critical section keluar dan melakukan opersai up sehingga menjadikan proses yang diblocked ready dan melanjutkan sehingga opersi Down-nya berhasil.

6. Problem Klasik pada Sinkronisasi

Ada tiga hal yang selalu memjadi masalah pada proses sinkronisasi:

a. Problem Bounded buffer.
b. Problem Reades and Writer.
c. Problem Dining Philosophers.

Monitors

Solusi sinkronisasi ini dikemukakan oleh Hoare pada tahun 1974. Monitor adalah kumpulan prosedur, variabel dan struktur data di satu modul atau paket khusus. Proses dapat memanggil prosedur-prosedur kapan pun diinginkan. Tapi proses tak dapat mengakses struktur data internal dalam monitor secara langsung. Hanya lewat prosedur-prosedur yang dideklarasikan minitor untuk mengakses struktur internal.

Properti-properti monitor adalah sebagai berikut:

a. Variabel-variabel data lokal, hanya dapat diakses oleh prosedur-prosedur dala monitor dan tidak oleh prosedur di luar monitor.
b. Hanya satu proses yang dapat aktif di monitor pada satu saat. Kompilator harus mengimplementasi ini(mutual exclusion).
c. Terdapat cara agar proses yang tidak dapat berlangsung di-blocked. Menambahkan variabel-variabel kondisi, dengan dua operasi, yaitu Wait dan Signal.
d. Wait: Ketika prosedur monitor tidak dapat berkanjut (misal producer menemui buffer penuh) menyebabkan proses pemanggil diblocked dan mengizinkan proses lain masuk monitor.
e. Signal: Proses membangunkan partner-nya yang sedang diblocked dengan signal pada variabel kondisi yang sedang ditunggu partnernya.
f. Versi Hoare: Setelah signal, membangunkan proses baru agar berjalan dan menunda proses lain.
g. Versi Brinch Hansen: Setelah melakukan signal, proses segera keluar dari monitor.

Dengan memaksakan disiplin hanya satu proses pada satu saat yang berjalan pada monitor, monitor menyediakan fasilitas mutual exclusion. Variabel-variabel data dalam monitor hanya dapat diakses oleh satu proses pada satu saat. Struktur data bersama dapat dilindungi dengan

menempatkannya dalam monitor. Jika data pada monitor merepresentasikan sumber daya, maka monitor menyediakan fasilitas mutual exclusion dalam mengakses sumber daya itu.

Proses Sinkronisasi

- Latar Belakang
- Masalah Critical Section
- Sinkronisasi Hardware
- Semaphores
- Monitors

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, pembagiantask, 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

- Menjamin konsistensi data:
    - Program/task-task dapat menghasilkan operasi yang benar setiap waktu
    - Deterministik: untuk input yang sama hasil harus sama (sesuai dengan logika/algroitma program).

- 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 (1)

- 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 (2)

- Shared data type item = ... ;
  var buffer array
  in, out: 0..n-1;
  counter: 0..n;
  in, out, counter := 0;

- Producer process

repeat
...
produce an item in nextp
...
while counter = n do no-op;
buffer [in] := nextp;
in := in + 1 mod n;
counter := counter +1;
until false;

Bounded Buffer (3)

- Consumer process

repeat
while counter = 0 do no-op;
nextc := buffer [out];
out := out + 1 mod n;
counter := counter – 1;
...
consume the item in nextc
...
until false;

Bounded Buffer (4)

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

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

        Load Reg1, Counter

        Add Reg1, 1

        Store Counter, Reg1

Bounded Buffer (5)

- “decrement counter”

    Load Reg2, Counter

    Subtract Reg2, 1

    Store Counter, Reg2

- Eksekusi P & C tergantung scheduler (dapat gantian)
    - T0: Producer : Load Reg1, Counter (Reg1 = 5)
    - T1: Producer : Add Reg1, 1 (Reg1 = 6)
    - T2: Consumer: Loag Reg2, Counter (Reg2 = 5)
    - T3: Consumer: Subtract Reg1, 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, ata 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

Solusi Masalah Critical Section

- Ide :

    - Mencakup pemakaian secara “exclusive” dari shared variable tersebut
    - Menjamin proses lain dapat menggunakan shared variable tersebut
- Solusi “critical section problem” harus memenuhi:
    - Mutual Exclusion: Jika proses Pi sedang “eksekusi” pada bagian “critical section” (dari proses Pi) maka tidak ada proses proses lain dapat “eksekusi” pada bagian critical section dari proses-proses tersebut.
    - Progress: Jika tidak ada proses sedang eksekusi pada critical section-nya dan jika terdapat lebih dari satu proses lain yang ingin masuk ke critical section, maka pemilihan siapa yang masuk ke critical section tidak dapat ditunda tanpa terbatas.

Solusi (cont.)

- Bounded Waiting: Terdapat batasan berapa lama suatu proses harus menunggu giliran untuk mengakses “critical section” – jika seandainya proses lain yang diberikan hak akses ke critical section.
    - Menjamin proses dapat mengakses ke “critical section” (tidak mengalami starvation: proses se-olah berhenti menunggu request akses ke critical section diperbolehkan). Tidak ada asumsi mengenai kecepatan eksekusi proses proses n tersebut.

Solusi Sederhana : Kasus 2
Proses

- Hanya 2 proses
- Struktur umum dari program code Pi dan Pj:
- Software solution: merancang algoritma program untuk solusi critical section
    - Proses dapat mengunakan “common var.” untuk menyusun algoritma tsb.

Algoritma 1

- Shared variables:
    - int turn;

initially turn = 0

turn - i ⇒P i dapat masuk ke criticalsection

- Process P i

```
do {
    while (turn != i) ;
        critical section
        turn = j;
                reminder section
        } while (1);
```

- Mutual exclusion terpenuhi, tetapi menentang progress

Algoritma 2

- Shared variables
  - boolean flag[2];
    initially flag [0] = flag [1] = false.
  - flag [i] = true ⇒P i siap dimasukkan ke dalam critical section

- Process P i

```
do {
flag[i] := true;
while (flag[j]) ;
        critical section
flag [i] = false;
        remainder section
} while (1);
```

- Mutual exclusion terpenuhi tetapi progress belum terpenuhi.

Algoritma 3

- Kombinasi shared variables dari algoritma 1 and 2.

- Process P i

```
do {
flag [i]:= true;
        turn = j;
while (flag [j] and turn = j) ;
critical section
flag [i] = false;
remainder section
} while (1);
```

- Ketiga kebutuhan terpenuhi, solusi masalah critical section pada dua proses

Algoritma Bakery

- Critical section untuk n proses
- Sebelum proses akan masuk ke dalam “critical section”, maka proses harus mendapatkan “nomor” (tiket).
- Proses dengan nomor terkecil berhak masuk ke critical section.
    - Jika proses P i dan P j menerima nomor yang sama, jika i < j, maka P i dilayani pertama; jika tidak P j dilayani pertama
- Skema penomoran selalu dibuat secara berurutan, misalnya 1,2,3,3,3,3,4,5...

Algoritma Bakery (2)

- Notasi <≡ urutan lexicographical (ticket #, process id #)
    - (a,b) < c,d) jika a < c atau jika a = c and b < d
    - max (a 0 ,..., a n-1) dimana a adalah nomor, k, seperti pada k ≥ a i untuk i - 0, ..., n – 1
- Shared data var choosing: array [0..n – 1] of Boolean number: array [0..n – 1] of integer,
- Initialized: choosing =: false ; number => 0

Algoritma Bakery (3)

```
do {
choosing[i] = true;
        number[i] = max(number[0], number[1], ..., number [n –
        1])+1;
choosing[i] = false;
                for (j = 0; j < n; j++) {
                while (choosing[j]) ;
while ((number[j] != 0) && (number[j,j] < number[i,i])) ;
                }
                critical section
        number[i] = 0;
remainder section
        } while (1);
```

Sinkronisasi Hardware

- Memerlukan dukungan hardware (prosesor)
- Dalam bentuk “instruction set” khusus: test-and-set
- Menjamin operasi atomik (satu kesatuan): test nilai dan
- ubah nilai tersebu
- Test-and-Set dapat dianalogikan dengan kode:

Test-and-Set (mutual exclusion)

- Mutual exclusion dapat diterapkan:
- Gunakan shared data,
    - variabel: lock: boolean (initially false)
    - lock: menjaga critical section
- Process Pi:

```
do {
while (TestAndSet(lock)) ;
critical section
```

lock = false;
remainder section
}

Semaphore

- Perangkat sinkronisasi yang tidak membutuhkan busy waiting
- Semaphore S – integer variable
  - Dapat dijamin akses ke var. S oleh dua operasi atomik:
    - wait (S): while S ≤ 0 do no-op;
      S := S – 1;
    - signal (S): S := S + 1;

Contoh : n proses

- Shared variables
  - var mutex : semaphore
  - initially mutex = 1
- Process Pi

```
do {
wait(mutex);
critical section
        signal(mutex);
remainder section
        } while (1);
```

Implementasi Semaphore

- Didefinisikan sebuah Semaphore dengan sebuah record

```
typedef struct {
        int value;
struct process *L;
        } semaphore;
```

- Diasumsikan terdapat 2 operasi sederhana :
    - block menhambat proses yang akan masuk
    - wakeup(P) memulai eksekusi pada proses P yang di block

Implementasi Semaphore (2)

- Operasi Semaphore-nya menjadi :

```
wait(S):
        S.value--;
        if (S.value < 0) {
        add this process to S.L;
        block;
        }

signal(S):
        S.value++;
        if (S.value <= 0) {
        remove a process P from S.L;
        wakeup(P);
        }
```

Masalah Klasik Sinkronisasi

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

Bounded-Buffer Problem

- Shared data

```
        semaphore full, empty, mutex;
        Initially:
```

full = 0, empty = n, mutex = 1

Bounded-Buffer Problem :

Producer-Consumer

Readers-Writers Problem

- Shared data

  semaphore mutex, wrt;

  Initially

  mutex = 1, wrt = 1, readcount = 0

Readers-Writers Problem (2)

- Writters Process

  wait(wrt);

  ...

  writing is performed

  ...

  signal(wrt);

- Readers Process

  wait(mutex);

  readcount++;

  if (readcount == 1)

  wait(rt);

  signal(mutex);

  ...

  reading is performed

  ...

  wait(mutex);

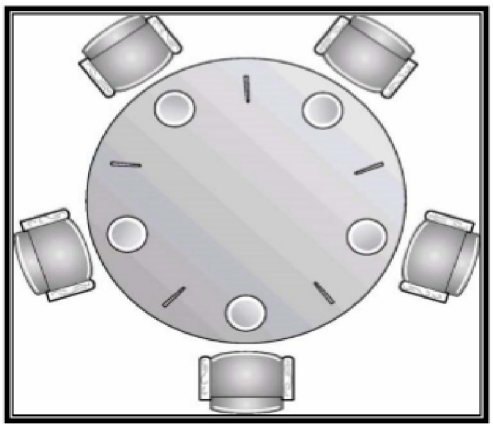
  readcount--;

  if (readcount == 0)

  signal(wrt);

signal(mutex):

Dining-Philosophers Problem



- Shared data

  semaphore chopstick[5];

  Semua inisialisasi bernilai 1

Dining-Philosophers Problem

- Philosopher i:

  do {

  wait(chopstick[i])

  wait(chopstick[(i+1) % 5])

  ...

  eat

  ...

  signal(chopstick[i]);

  signal(chopstick[(i+1) % 5]);

  ...

  think

  ...

  } while (1);

Solusi Tingkat Tinggi

- Motif:
    - Operasi wait(S) dan signal(S) tersebar pada code program
      => manipulasi langsung struktur data semaphore
    - Bagaimana jika terdapat bantuan dari lingkungan HLL (programming) untuk sinkronisasi ?
    - Pemrograman tingkat tinggi disediakan sintaks-sintaks khusus untuk menjamin sinkronisasi antar proses, thread
- Misalnya:
    - Monitor & Condition
    - Conditional Critical Region

Monitor

- Monitor mensinkronisasi sejumlah proses:
    - suatu saat hanya satu yang aktif dalam monitor dan yang lain menunggu
- Bagian dari bahasa program (mis. Java)
    - Tugas compiler menjamin hal tersebut terjadi dengan menerjemahkan ke “low level synchronization” (semphore, instruction set dll)
- Cukup dengan statement (deklarasi) suatu section/fungsi adalah monitor => mengharuskan hanya ada satu proses yang berada dalam monitor (section) tsb

Monitor (2)

```
Type monitor-name = monitor
        Variabel declarations
    Procedure entry P1 :(...);
            Begin ... end;
                    :
    Procedure entry Pn (...);
            Begin ... end;
           begin
```

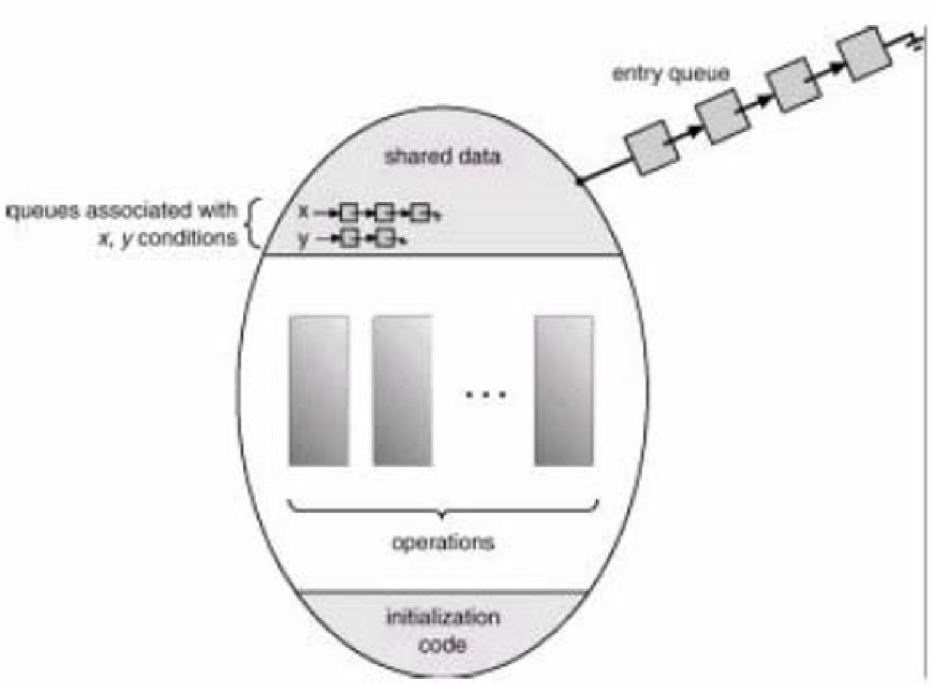
Monitor (3)

- Proses-proses harus disinkronisasikan di dalam monitor:
  - Memenuhi solusi critical section.
  - Proses dapat menunggu di dalam monitor.
  - Mekanisme: terdapat variabel (condition) dimana proses dapat menguji/menunggu sebelum mengakses “critical section” var x, y: condition

Monitor (4)

- Condition: memudahkan programmer untuk menulis code pada monitor.
  Misalkan : var x: condition ;
- Variabel condition hanya dapat dimanipulasi dengan operasi:
  wait() dan signal()
  - x.wait() jika dipanggil oleh suatu proses maka proses tsb. akan suspend - sampai ada proses lain yang memanggil:
    x. signal()
  - x.signal() hanya akan menjalankan (resume) 1 proses saja yang sedang menunggu (suspend) (tidak ada proses lain yang wait maka tidak berdampak apapun)

Skema Monitor



C. SOAL LATIHAN/TUGAS

1. Jelaskan pengertian racecondition?
2. Jelaskan problem klasik pada sinkronisasi?
3. Jelaskan properti-properti monitor?

D. DAFTAR PUSTAKA

Buku

Bambang Hariyanto. 1997. Sistem Operasi, Bandung:Informatika Bandung.

Dali S. Naga. 1992. Teori dan Soal Sistem Operasi Komputer,Jakarta: Gunadarma.

Operating System Concepts (6th or 7th Edition). Silberschatz, Galvin, Gagne, ISBN: 0-471-25060-0. Wiley

Silberschatz Galvin. 1995. 4 Edition Operating System Concepts: Addison Wesley.

Sri Kusumadewi. 2000. Sistem Operasi. Yogyakarta: J&J Learning.

Tanenbaum, A.1992. Modern Operating Systems.New York: Prentice Hall

Link and Sites:

http://www.ilmukomputer.com
http://vlsm.bebas.org
http://www.wikipedia.com