|  |  |  |
| --- | --- | --- |
| MODUL VIII  CCS 210 SISTEM OPERASI | | |
| Judul | KONKURENSI DASAR | |
| Penyusun | Distribusi | Perkuliahan |
| **Nixon Erzed** | **FASILKOM**  **UNIVERSITAS ESA UNGGUL** | Pertemuan – VIII  OL - 8 |

**KONKURENSI DASAR**

**(Konkurensi 2 proses)**

Konkurensi 🡪 persoalan proses-proses konkuren

🡪 Proses-proses disebut konkuren jika terdapat lebih dari satu proses pada saat sama dan membutuhkan layanan pemroses.

Terdapat beberapa kemungkinan keadaan proses konkuren, yaitu :

1. saling bebas, tapi terdapat persaingan untuk mendapatkan sumber daya (🡪 sumber daya dipakai secara eksklusif)

misal P1 dan P2 adalah proses konkuren, dan tidak data/informasi saling dipertukarkan, tapi P1 dan P2 tetap dieksekusi oleh mesin yang sama sehingga terjadi persaingan.

**Proses 2**

**Proses 1**

1. berinteraksi secara tidak langsung, melalui pemakaian sumber daya bersama (🡪 data bersama/variabel bersama) 🡪 saling mempengaruhi

**Proses 2**

**Proses 1**

**X**

Ilustrasi masalah

Misalkan proses 1 dan proses 2 bekerja dengan nilai kecepatan yang dibaca dari variabel x, proses yang membaca variabel berkewajiban untuk meng-update nilai x menjadi 2 kalinya.

Misalkan nilai awal x = 10

Sehingga proses yang pertama kali mendapatkan x akan bekerja dengan kecepatan 10

Jika proses 1 yang mendapatkan kesempatan awal membaca x dan kemudian mengubahnya menjadi 20

🡪 proses 1 bekerja dengan kecepatan 10

🡪 proses 2 mendapatkan x = 20 dan bekerja dengan kecapatan 20 dan mengupdate x =40

Jika proses 2 yang mendapatkan kesempatan awal membaca x dan kemudian mengubahnya menjadi 20

🡪 proses 2 bekerja dengan kecepatan 10

🡪 proses 1 mendapatkan x = 20 dan bekerja dengan kecapatan 20 dan mengupdate x =40

1. saling berinterakasi secara langsung, terdapat komunikasi antar proses (🡪 output P1 menjadi input P2 )

🡪 suatu proses tergantung pada proses lainnya

X

**Proses 2**

**Proses 1**

Output P1 menjadi masukan P2 , sehingga P2 tidak dapat diselesaikan jika P1 belum menghasilkan output x

Proses 1 Proses 2

|  |  |  |
| --- | --- | --- |
| Ins 1.1 |  | Ins 2.1 |
| Ins 1.2 |  | Ins 2.2 |
| Ins 1.3 |  | Ins 2.3 ambil x |
| Ins 1.4 |  | Ins 2.4 |
| Ins 1.5 |  | Ins 2.5 |
| Ins 1.6 |  | Ins 2.6 |
| Ins 1.7 keluarkan x |  | Ins 2.7 |
| Ins 1.8 |  | Ins 2.8 |
| Ins 1.9 |  | Ins 2.9 |
| Ins 1.10 |  | Ins 2.10 |

Proses 1 Proses 2

|  |  |  |
| --- | --- | --- |
| Ins 1.1 |  | Ins 2.1 |
| Ins 1.2 |  | Ins 2.2 |
| Ins 1.3 |  |  |
| Ins 1.4 |  |  |
| Ins 1.5 |  |  |
| Ins 1.6 |  |  |
| Ins 1.7 keluarkan x |  |  |
| Ins 1.8 |  | Ins 2.3 ambil x |
| Ins 1.9 |  | Ins 2.4 |
| Ins 1.10 |  | Ins 2.5 |
|  |  | Ins 2.6 |
|  |  | Ins 2.7 |
|  |  | Ins 2.8 |
|  |  | Ins 2.9 |
|  |  | Ins 2.10 |

Persoalan-persoalan yang muncul pada proses konkuren

1. Alokasi waktu pemroses 🡪 pada monoprosesor atau multiprosesor

Implementasi kebijakan penjadwalan 🡪 Scheduling

Kegagalan kebijakan penjadwalan menimbulkan resiko :

🡪 starvation ( proses terkucil ), inefisiensi, respons yang buruk, dll

Contoh :

*Pada implementasi algoritma SJF, proses yang secara ekstrim membutuhkan waktu proses relatif sangat panjang dibanding proses yang lain akan beresiko tidak kebagian waktu pemroses (terkucil)*

Cukup sulit menemukan algoritma yang memenuhi criteria penjadwalan untuk semua keadaan 🡪 resiko starvation .

Pemulihan starvation 🡪 digunakan penjadwal jangka panjang (yang mengintervensi antrian agar proses terkucil diprioritaskan)

1. Pemakaian sumber daya bersama (variabel bersama) 🡪 MUTEX (Mutual Exclusion)

Terdapat sumber daya bersama yang bersifat ekslusif, yang tidak boleh diakses oleh lebih dari satu proses pada suatu saat

Resiko 🡪 race condition 🡪 keadaan dimana hasil proses tidak

sesuai dengan dugaan/prediksi

Ilustrasi :

*Misalkan untuk menentukan kamar yang akan ditempati setiap mahasiswa (1 kamar untuk 1 mahasiswa), terdapat sebuah papan panduan nomor kamar. Pada papan tertulis nomor kamar yang boleh diisi. Setiap mahasiswa berebut membaca papan, setelah membaca nomor dipapan, nomor harus diupdate agar pembaca berikutnya tidak masuk kamar yang sama.*

*🡪 papan nomor adalah sumber daya bersama, dan bersifat kritis*

*menuntut pengaksesan ekslusif*

🡪 jika akses ekslusif gagal dijamin 🡪 race condition terjadi

Isi kamar menjadi tidak konsisten (tidak sesuai dengan prediksi)

Mutex 🡪 menjamin akses ekslusif terhadap seksi kritis

1. Persaingan mendapatkan sumber daya 🡪 DEAD LOCK
   * Proses-proses membutuhkan sekumpulan sumber daya untuk menyelesaikan jobnya
   * Terdapat lebih dari satu proses yang meminta layanan, terdapat banyak sumber daya
   * Beberapa sumber daya mungkin dibutuhkan oleh beberapa proses berbeda.
   * Terdapat kemungkinan proses-proses saling menguasai sumber daya dan menunggu sumber daya lain yang dikuasai proses lain

🡪 jika gagal dikendalikan 🡪 circular wait 🡪 Terjadilah deadlock

🡪 penyelesaian deadlock : Deteksi dan Recovery

1. Sinkronisasi proses

Pada proses-proses yang berinteraksi secara langsung, terjadi komunikasi berupa output suatu proses menjadi input proses lainnya. 🡪 terdapat ketentuan predesesor – suksesor

Jika proses-proses hadir pada saat yang sama dan meminta layanan prosesor : *harus disinkronkan*

Proses predesesor harus dilayani terlebih dahulu

Resiko 🡪 tidak sinkron

Output P1 menjadi masukan P2

**Proses 1**

**Proses 1**

**Proses 2**

ketentuan

proses datang bersama-sama

dan masuk antrian

**Mikro Prosesor**

**Proses 2**

tidak sinkron jika : P2 mendapat jatah waktu lebih dahulu

Penyelesaian masalah sinkronisasi : menyerahkan pada mekanisme diagram state 🡪

*jika suksesor running sebelum predesesor, maka akan ditemukan event menunggu suatu nilai input sehingga proses akan Blocked*

Dasar untuk penyelesaian proses-proses konkuren

1. Pemahaman tentang implementasi multitasking
2. pemrosesan paralel 🡪 algoritma paralel
3. pemahaman tentang alokasi sumber daya

# Penyelesaian masalah konkurensi :

|  |  |
| --- | --- |
| **Masalah** | **Solusi** |
| Alokasi waktu pemroses  🡪 bagaimana menjamin pencapaian keadaan terbaik pada pemenuhan kriteria penjadwalan | Menerapkan algoritma penjadwalan yang memenuhi criteria penjadwalan secara optimal.  (fairness, efficient, min. response time, min. turn around time, max. throughput)  Jika masih gagal 🡪 intervensi penjadwal jangka panjang |
| Pemakaian sumber daya bersama yang bersifat kritis  🡪 menjamin akses eksklusif | Menjamin Mutual Exclusion  🡪 algoritma2 mutex |
| Persaingan mendapatkan sumber daya  🡪 | Deteksi dan recovery kejadian deadlock |
| Sinkronisasi proses | Analisis hubungan proses-proses  oleh kernel 🡪 graf keterdahuluan  1. sulit diterapkan pada multiprogramming (kasusnya juga jarang) :  *diselesaikan dengan mekanisme siklus state proses*  2. terjadi pada pemrosesan parallel statement-statement suatu proses tunggal 🡪 terdapat graf keterdahuluan yang dapat diacu  🡪 mengacu pada Bernstein condition |

# MUTUAL EXCLUSION

*Mutual exclusion* : jaminan hanya satu proses yang mengakses sumber daya kritis pada suatu interval waktu tertentu.

### Critical section, race condition, mutual exclusion

Seksi kritis berkaitan dengan sumber daya kritis

🡪 sumber daya kritis :

sumber daya sistem komputer yang menuntut pengaksesan secara eksklusif, jika tidak dapat dijamin akses eksklusif tsb 🡪 kekacauan proses

🡪 seksi kritis

Aksi eksklusif yang terhadap sumber daya kritis

Kondisi Pacuan (race condition)

Kekacauan yang terjadi akibat akses eksklusif terhadap sumber daya kritis tidak dapat dijamin

Mutual Exclusion

🡪 menjamin akses eksklusif

Algoritma Mutex 🡪 algoritma untuk menjamin akses eksklusif

Ilustrasi pentingnya *mutual exclusion* :

* Ilustrasi ilustrasi eksekusi *daemon printer.*
* Ilustrasi aplikasi tabungan.

**Ilustrasi Printer Daemon**

* ***Daemon*** printer adalah proses penjadwalan dan pengendalian percetakan berkas – berkas di printer sehingga seolah – olah printer dapat digunakan secara simultan oleh proses – proses.
* Daemon printer mempunyai ruang *disk* ( disebut direktori *spooler* ) untuk menyimpan berkas – berkas yang akan dicetak.
* Direktori *spooler* membagi *disk* menjadi sejumlah slot. Slot – slot diisi berkas yang akan dicetak. Terdapat variabel *in* menunjuk slot bebas di ruang *disk* yang akan dipakai menyimpan berkas yang ingin dijadwalkan untuk dicetak.
* Variabel *in* adalah sebuah variabel bersama dan bersifat kritis

|  |  |
| --- | --- |
|  | **Program Give – file – to – spooler;**  **Var**  **in : Integer;**  **berkas A, berkas B : File;**  **Procedure Store (Brks : File, nx– slot :Integer );**  **{ Untuk menyimpan berkas pada slot yang beralamat nx – slot }**  **Procedure Proses A;** Var   **next – Free – slot : Integer;** Begin **next – free – slot : = in;**  **Store ( berkas A, next – free – slot );**  **in : = next – free – slot + 1;**  **End-proses A;**  **Procedure Proses B;** Var **next – free – slot : Integer** Begin **next – free – slot : = in;**  **Store ( berkas B, next – free – slot );**  **in : = next – free – slot + 1;**  **End-proses B** Begin **in : = 0;**  **Repeat**  **Parbegin**  **Proses A;**  **Proses B;** Parend **Forever** End |

## Skenario yang Membuat Situasi Kacau

Misal proses A dan B ingin mencetak berkas, variabel **in** bernilai 9.

|  |  |  |
| --- | --- | --- |
| Clock | Proses A | Proses B |
| 1 | { in = 9 }  next – free – slot 🡸 in  {proses A dapat alokasi slot-9} |  |
| 2 | { Penjadwalan menjadwalkan B berjalan } | { in = 9 }  next – free – slot 🡸 in  {proses B juga dapat alokasi slot-9 |
| 3 |  | store berkas B to slot [ next – free – slot ]  { berkas B disimpan di slot ke – 9 } |
| 4 |  | in 🡸 next – free – slot + 1  { in 🡸 9 + 1 🡺 10 } |
| 5 | store berkas A to slot [ next – free – slot ]  ***{ berkas A disimpan di slot ke- 9, menimpa berkas B}*** | { penjadwal menjadwalkan A berjalan } |
| 6 | in 🡸 next – free – slot + 1  { in 🡸 9 + 1 = 10 } |  |

Skenario Sukses :

|  |  |  |
| --- | --- | --- |
| Clock | Proses A | Proses B |
| 1 | { in = 9 }  next – free – slot 🡸 in  {proses A dapat alokasi slot-9} |  |
| 2 | store berkas A to slot [ next – free – slot ]  { berkas A disimpan di slot ke- 9} |  |
| 3 | in 🡸 next – free – slot + 1  { in 🡸 9 + 1 = 10 } |  |
| 4 | { Penjadwalan menjadwalkan B berjalan } | { in = 10 }  next – free – slot 🡸 in  {proses B dapat alokasi slot-10 |
| 5 |  | store berkas B to slot [ next – free – slot ]  { berkas B disimpan di slot ke –10 } |
| 6 |  | In 🡸 next – free – slot + 1  { in 🡸 10 + 1 🡺 11 } |

Skenario sukses tidak mungkin selalu terpenuhi, karena **persaingan bebas proses-proses.**

### Kriteria Penyelesaian Mutual Exclusion

1. *Mutual exclusion* harus dijamin

Seksi kritis dijamin hanya diakses oleh satu proses pada satu saat

*Hanya satu proses pada satu saat yang diijinkan masuk critical section. Proses – proses lain dilarang masuk critical section yang sama pada saat telah terdapat proses masuk di critical section itu.*

1. Proses yang berada di *noncritical section*, dilarang mem *– blocked proses* – proses lain yang ingin masuk *critical section.*
2. Harus dijamin proses yang ingin masuk critical section tidak menunggu selama waktu yang tak berhingga. Atau tak boleh terdapat *deadlock* atau *startvation.*
3. Ketika tidak ada proses pada *critical section* maka proses yang ingin masuk *critical section* harus diijinkan masuk tanpa waktu tunda.
4. Tidak ada asumsi mengenai kecepatan relatif proses atau jumlah proses yang ada.

## Metoda-Metoda Dasar Penyelesaian MUTEX antara lain :

1. Metoda Variabel Lock Sederhana

Menggunakan Variabel Lock sebagai pengunci Seksi Kritis

Jika Lock = 1 maka proses tidak boleh memasuki Seksi Kritis

Jika Lock = 0 maka proses dipersilahkan memasuki Seksi Kritis dan segera menset Lock = 1 agar proses lain tidak memasuki Seksi Kritis.

Masalah : Proses paralel terjadi pada instruksi dasar, sehingga tidak dapat dipaksa suatu proses tidak memasuki seksi kritis ketika proses yang duluan masuk belum sempat menset Variabel Lock.

1. Metoda Pergantian Secara Ketat

Menggunakan Variabel Turn dengan nilai yang berbeda antara suatu proses dengan proses lainnya.

Masuknya proses-proses terjadi pada prosedur.

1. Algoritma Peterson
2. Semaphore

Semaphore adalah teknik penyelesaian Mutex dengan memanfaatkan Queue

**Metoda dengan Variabel Lock Sederhana (Metoda Naif)**

|  |  |
| --- | --- |
| ***Program Mutex – with – lock;***  ***Var***  *lock* ***: Integer****;*  ***Procedure*** *Enter\_critical\_section;*  *{ Mengerjakan kode – kode krisis }*  ***Procedure*** *Enter\_non\_critical\_section;*  *{ Mengerjakan kode – kode non krisis }*  ***Procedure*** *Proses A;* *Begin* ***While lock*** *<> 0* ***Do Begin End;***  *lock : = 1;*  *enter – critical – section;*  *lock : = 0;*  *enter\_non\_critical\_section;*  ***End;***  ***Procedure*** *Proses B;* *Begin* ***While lock <> 0 Do Begin End;***  *lock : = 1;*  *enter – critical – section;*  *lock : = 0;*  *enter\_non\_critical\_section;*  ***End;*** | *Begin*    *lock : = 0;*  *Repeat* ***Parbegin***  *Proses A;*  *Proses B;*  *Parend* ***Forever*** *End* |
| ***Procedure*** *Enter\_critical\_section;*  begin  *next – free – slot : = in*  *in : = next – free – slot + 1*  ***end;***  ***Procedure*** *Enter\_non\_critical\_section;*  *{ Mengerjakan kode – kode non krisis }*  *begin*  *store (berkas A ke next\_free\_slot);*  ***end*** |
|  | |

**Skenario Sukses/Gagal**

|  |  |  |
| --- | --- | --- |
| Clock | Proses A | Proses B |
| 1 | **While lock** <> 0 **Do Begin End;**  **(Lock = 0 )<> 0 false 🡪 PA PASS** |  |
| 2 | Lock := 1 (mengunci) |  |
| 3 |  | **While lock** <> 0 **Do Begin End;** *(***PB** *blocked)*  **(Lock=1) <> 0 🡪 benar looping** |
| 4 |  | **While lock** <> 0 **Do Begin End;** *(***PB** *blocked))* |
| 5 | Mengerjakan Seksi Kritis (baca IN dan Update) |  |
| 6 |  | **While lock** <> 0 **Do Begin End;** *(blocked)* |
| 7 | Lock := 0 (melepas kunci) |  |
| 8 |  | **While lock** <> 0 **Do Begin End;** *(***PB** *PASS)* |

|  |  |  |
| --- | --- | --- |
| **Clock** | **Proses A** | **Proses B** |
|  | **While lock** <> 0 **Do Begin End; PASS** |  |
|  |  | **While lock** <> 0 **Do Begin End;** *(PASS)* |
|  | Lock := 1 (mengunci) |  |
|  |  | Lock := 1 (mengunci ULANG) KEKACAUAN TERJADI |
|  |  | Mengerjakan Seksi Kritis (baca IN ) |
|  | Mengerjakan Seksi Kritis (baca IN ) |  |
|  | Mengerjakan seksi kritis (Update IN) |  |
|  |  | Mengerjakan seksi kritis (Update IN) |
|  | Lock := 0 (melepas kunci) |  |
|  |  |  |

Kuliah 6

**Metoda Bergantian Secara Ketat**

Metoda ini mengatur supaya urutan giliran antara beberapa proses menjadi lebih adil. Metoda mengizinkan parallel dua proses pada saat yang sama.

**Parbegin**

**Inisialisasi T = 0**

**tdk**

ya

**Masuk SK**

**T ⇐ 0**

**Masuk SnK**

**tdk**

**ya**

**Masuk SK**

**T ⇐ 1**

**Masuk SnK**

**T<> 1**

**T <> 0**

***Repeat - Forever***

***Repeat - Forever***

**Parend**

**Data proses :**

**Id proses w.datang size paket**

|  |  |  |
| --- | --- | --- |
| **P0** | **0** | **10** |
| **P1** | **1** | **2** |
| **P2** | **4** | **2** |
| **P3** | **20** | **3** |

Status Awal : turn = 0, nomor slot kosong berikutnya = 4

**Digunakan sebuah variabel *turn* sebagai kendali seksi kritis :**

* **Jika variabel *turn* = 0 maka proses di jalur 0 diijinkan masuk seksi kritis, dan proses di jalur 1 blocked**
* **Jika variabel *turn* = 1 maka proses di jalur 1 diijinkan masuk seksi kritis dan proses di jalur 0 blocked**

|  |  |
| --- | --- |
| *Program Mutex – with – strict – alternation;* *Var* *turn : Integer;*  *Procedure enter – critical – section;*  *{ Mengerjakan kode – kode kritis }*  *Procedere enter – noncritical – section;*  *{ Mengerjakan kode – kode tak kritis }*  *Procedure Proses 0;*  *Begin*  *Repeat*  ***While turn <> 0 Do Begin End****;*  *enter – critical – section;*  *turn : = 1;*  *enter – noncritical – section;*  *Forever*  *End;* | *Procedure Proses 1;*  *Begin*  *Repeat*  ***While turn <> 1 Do Begin End****;*    *enter – critical – section;*  *turn : = 0;*  *enter – noncritical – section;*  *Forever*  *End;*  *Begin*  *turn : = 0;*  *Parbegin*  *Proses 0;*  *Proses 1;*  *Parend*  *End.* |

Metoda ini dapat menghidarkan *race condition* tapi gagal menyelesaikan Mutex karena proses yang berada di seksi non kritis dapat menghalangi proses lain masuk ke seksi kritis

Kelemahannya 🡪 looping busy waiting hanya memeriksa nilai kunci, dan tidak memeriksa apakah apakah proses yang menghalangi berada di SK atau bukan

**Data proses :**

**Id proses w.datang size paket**

|  |  |  |
| --- | --- | --- |
| **P0** | **0** | **10** |
| **P1** | **1** | **2** |
| **P2** | **4** | **2** |
| **P3** | **20** | **3** |

Status Awal : turn = 0, nomor slot kosong berikutnya = 4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Clock** | **P0** | **P1** | **P2** | **P3** | **SK** | **proses dieksekusi** | **Deskripsi** |
|  | **Jalur 0** |  |  |  | Idle | P0 | Turn <> 0 => 0<>0 = F; P0 diizinkan masuk SK |
|  | **In SK** | **Jalur 1** |  |  | P0 | P0 | In SK => get slot P0= 4 |
|  | **In SK** | **blocked** |  |  | P0 | P1 | Turn <> 1 => 0<>1 = T; P1 blocked |
|  | **In SK** | **blocked** |  |  | P0 | P0 | In SK => update slot = 5 |
|  | **In SK** | **blocked** | **Delay** |  | P0 | P1 | Turn <> 1 => 0<>1 = T; P1 blocked |
|  | **In SK** | **blocked** | **Delay** |  | P0 out | P0 | Turn <= 1 |
|  | **In SNK** | **In SK** | **Delay** |  | P1 | P1 | Turn <> 1 => 1<>1 = F; P1 diizinkan masuk SK |
|  | **In SNK** | **In SK** | **Delay** |  | P1 | P0 | P0 In SNK (1) |
|  | **In SNK** | **In SK** | **Delay** |  | P1 | P1 | In SK => get slot P1=5 |
|  | **In SNK** | **In SK** | **Delay** |  | P1 | P0 | P0 In SNK (2) |
|  | **In SNK** | **In SK** | **Delay** |  | P1 | P1 | In SK => update Slot = 6 |
|  | **In SNK** | **In SK** | **Delay** |  | P1 | P0 | P0 In SNK (3) |
|  | **In SNK** | **In SK** | **Delay** |  | P1 out | P1 | Turn <= 0 |
|  | **In SNK** | **inSNK** | **Delay** |  | Idle | P0 | P0 In SNK (4) |
|  | **In SNK** | **inSNK** | **Delay** |  | Idle | P1 | P1 In SNK (1) |
|  | **In SNK** | **inSNK** | **Delay** |  | Idle | P0 | P0 In SNK (5) |
|  | **In SNK** | **inSNK** | **Delay** |  | Idle | P1 | P1 In SNK (2) P1 Selesai jalur 1 kosong |
|  | **In SNK** |  | **Jalur 1** |  | Idle | P0 | P0 In SNK (6) |
|  | **In SNK** |  | **blocked** |  | Idle | P2 | Turn <> 1 🡺 0 <> 1 = True  🡪 P2 blocked padahal SK idle |
|  | **In SNK** |  | **Blocked** |  | Idle | P0 | P0 In SNK (7) |
|  | **In SNK** |  | **blocked** | **Delay** | Idle | P2 | Turn <> 1 🡺 0 <> 1 = True  🡪 P2 blocked padahal SK idle |
|  | **In SNK** |  | **blocked** | **Delay** | Idle | P0 | P0 In SNK (8) |
|  | **In SNK** |  | **blocked** | **Delay** | Idle | P2 | Turn <> 1 🡺 0 <> 1 = True  🡪 P2 blocked padahal SK idle |
|  | **In SNK** |  | **blocked** | **Delay** | Idle | P0 | P0 In SNK (9) |
|  | **In SNK** |  | **blocked** | **Delay** | Idle | P2 | Turn <> 1 🡺 0 <> 1 = True  🡪 P2 blocked padahal SK idle |
|  | **In SNK** |  | **blocked** | **Delay** | Idle | P0 | P0 In SNK (10) P0 Selesai  jalur 0 kosong P3 masuk jalur 0 |
|  |  |  | **blocked** | **Jalur 0** | Idle | P3 |  |
|  |  |  | **blocked** |  |  |  |  |
|  |  |  | **blocked** |  |  |  |  |
|  |  |  | **blocked** |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**Turn = 0 nomor Slot 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Clock** | **Action** |  | **P0** | **P1** | **P2** | **P3** |
|  | T<> 0 = F P0 Pass | P0 | Jalur 0 |  |  |  |
|  | T<> 1 = T P1 Blocked | P1 |  | Jalur 1 |  |  |
|  | P0 in SK get slot 3, slot => 4 | P0 |  |  |  |  |
|  | T<> 1 = T P1 Blocked | P1 |  |  | delay |  |
|  | T⇐1 | P0 |  |  | delay |  |
|  | T<> 1 = F P1 Pass | P1 |  |  | delay |  |
|  | P0 in SNK (1) | P0 |  |  | delay |  |
|  | P1 in SK get slot 4, slot => 5 | P1 |  |  | delay |  |
|  | P0 in SNK (2) | P0 |  |  | delay |  |
|  | T⇐ 0 | P1 |  |  | delay |  |
|  | P0 in SNK (3) | P0 |  |  | delay |  |
|  | P1 in SNK (1) | P1 |  |  | delay |  |
|  | P0 in SNK (4) | P0 |  |  | delay |  |
|  | P1 in SNK (2) selesai | P1 |  |  | Jalur 1 |  |
|  | P0 in SNK (5) | P0 |  |  |  |  |
|  | T<> 1 = True : P2 blocked | P2 |  |  | Terkunci padahal SK kosong |  |
|  | P0 in SNK (6) | P0 |  |  |  |
|  | T<> 1 = True : P2 blocked | P2 |  |  |  |
|  | P0 in SNK (7) | P0 |  |  |  |
|  | T<> 1 = True : P2 blocked | P2 |  |  |  |
|  | P0 in SNK (8) | P0 |  |  | delay |
|  | T<> 1 = True : P2 blocked | P2 |  |  | delay |
|  | P0 in SNK (9) | P0 |  |  | delay |
|  | T<> 1 = True : P2 blocked | P2 |  |  | delay |
|  | P0 in SNK (10) selesai | P0 |  |  | Locked | Jalur 0 |
|  | T<> 1 = True : P2 blocked | P2 |  |  | Locked |  |
|  | T<> 0 = false : P3 pass | P3 |  |  |  | P3 pass |
|  | T<> 1 = True : P2 blocked | P2 |  |  |  | P3 pass |
|  | P3 in SK get slot 5, slot => 6 | P3 |  |  |  | P3 pass |
|  | T<> 1 = True : P2 blocked | P2 |  |  |  | P3 pass |
|  | T⇐1 | P3 |  |  |  | P3 pass |
|  | T<> 1 = False : P2 Pass | P2 |  |  |  | P3 pass |
|  | P3 in SNK (1) | P3 |  |  |  | P3 pass |
|  | P2 in SK get slot 6, slot => 7 | P2 |  |  |  | P3 pass |

**Metode Penyelesaian Peterson (2 proses)**

**(perbaikan metoda Bergantian Secara Ketat)**

**Mekanisme kerja Alg. Peterson adalah sebagai berikut :**

Terdapat dua variabel kendali seksi kritis :

* Array interested[ i ] dengan type Boolean 🡪 true, false 🡪 i = 0 dan 1
* Variabel bersama Turn

Proses 0

Interested [ 0 ] untuk mengidentifikasikan status proses\_0

Jika Interested [ 0 ] bernilai true berarti proses\_0 ada dan akan/sedang masuk\_SK

Dan jika Turn = 0 maka proses\_0 memasuki seksi kritis dan memblok proses\_1

Tapi jika Turn = 1 dan proses\_1 ada, maka proses\_0 akan diblok

Looping kendali : While interested [1] and turn = 1 do begin end

Jika Interested [ 0 ] bernilai false berarti proses di jalur 0 tidak ada atau sedang di SnK

Proses 1

Interested [ 1 ] untuk mengidentifikasikan status proses\_1

Jika Interested [ 1 ] bernilai true berarti proses\_1 ada akan/sedang masuk\_SK

Dan jika Turn = 1 maka proses\_1 memasuki seksi kritis dan memblok proses\_0

Tapi jika Turn = 0 dan proses\_0 ada, maka proses\_1 akan diblok

Looping kendali : While interested [0] and turn = 0 do begin end

**Note :**

Ketika datang proses\_0, Interested [ 0 ] akan diberi nilai true, dan turn = 1

Ketika datang proses\_1 Interested [ 1 ] akan diberi nilai true, dan turn = 0

Jika dua proses datang secara parallel, maka proses yang terakhir melaksanakan

intruksi pengisian nilai turn, akan diblok.

|  |  |
| --- | --- |
| 1  2  3  4  5 | Algoritma Peterson;  Const n = 2  Var  Interested : array [0..n-1] of Boolean  Turn : integer |
| 6  7 | Prosesdur Masuk\_SK;  { mengerjakan instruksi-instruksi di Seksi Kritis }  n<= vb  vb <= n +1  Prosedur Masuk\_SnK;  { mengerjakan instruksi-instruksi di Seksi non Kritis} |
| 8  9  10  11  12  13  14  15  16  17  18 | Prosedur Proses\_0;  Begin  **Interested [ 0 ] ⇐ true**  **Turn ⇐ 1**  **tdk**  **ya**  **Masuk SK**  **Interested [ 0 ] ⇐ false**  **Masuk SnK**  Repeat  Interested[0] ⇐ true  Turn ⇐ 1  Periksa Proses 1 &  Turn = 1  While interested [1] and turn = 1 do begin end  Masuk\_SK;  Interested [0] ⇐ false  Masuk\_SnK  Forever  End-Proses\_0  Repeat forever |
| 19  20  21  22  23  24  25  26  27  28  29 | Prosedur Proses\_1;  Begin  Repeat  Interested[1] ⇐ true  Turn ⇐ 0  While interested [0] and turn = 0 do begin end  Masuk\_SK;  Interested [1] ⇐ false  Masuk\_SnK  Forever  End-Proses\_1 | |
| 30  31  32  33  34  35  36  37  38 | Begin  Interested [ 0 ] ⇐ false  Interested [ 1 ] ⇐ false  Turn ⇐ 1  Parbegin  Proses\_0  Proses\_1  Parend  End-Peterson | |

Algoritma penyelesain Peterson dapat dideskripsikan sebagai berikut :

**Flowchart dari Algoritma Peterson**

Parbegin

**Interested [ 0 ] ⇐ true**

**Turn ⇐ 1**

**tdk**

**ya**

**Masuk SK**

**Interested[0] ⇐ false**

**Masuk SnK**

**Interested[1] ⇐ true**

**Turn ⇐ 0**

**tdk**

**ya**

**Masuk SK**

**Interested[1] ⇐ false**

**Masuk SnK**

Periksa Proses 0 &

Turn = 0

Parend

Periksa Proses 1 &

Turn = 1

***Repeat Forever Repeat Forever***

**Data proses :**

|  |  |  |
| --- | --- | --- |
| **P0** | **0** | **10** |
| **P1** | **1** | **2** |
| **P2** | **4** | **2** |
| **P3** | **20** | **3** |

**Turn = 0 nomor Slot berikutnya 🡺 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Clock** | **Action** |  | **P0** | **P1** | **P2** | **P3** |
|  | Int[0] ⇐ True | P0 | Jalur 0 |  |  |  |
|  | Int[1] <= true | P1 |  | Jalur 1 |  |  |
|  | Turn = 1 | P0 |  |  |  |  |
|  | Turn = 0 | P1 |  |  | delay |  |
|  | Int[1] ∧Turn =1 : T ∧ T= T  P0 blocked | P0 | Blocked |  | delay |  |
|  | Int[0] ∧Turn =0 : T ∧ F = F  P1 PASS | P1 | Blocked | Pass | delay |  |
|  | Int[1] ∧Turn =1 : T ∧ T= T  P0 blocked | P0 | Blocked | Pass | delay |  |
|  | P1 In SK get slot 3; slot => 4 | P1 | Blocked | Pass | delay |  |
|  | Int[1] ∧Turn =1 : T ∧ T= T  P0 blocked | P0 | Blocked |  | delay |  |
|  | Int[1] <= F | P1 | Blocked | Out SK | delay |  |
|  | Int[1] ∧Turn =1 : F ∧ T= F  P0 Pass | P0 | Pass |  | delay |  |
|  | P1 in SNK (1) | P1 |  |  | delay |  |
|  | P0 in SK get slot 4; slot => 5 | P0 |  |  | delay |  |
|  | P1 in SNK (2) selesai | P1 |  | Finish | Jalur 1 |  |
|  | Int[0] <= false | P0 |  |  |  |  |
|  | Int[1] <= true | P2 |  |  |  |  |
|  | P0 in SnK (1) | P0 |  |  |  |  |
|  | Turn = 0 | P2 |  |  |  |  |
|  | P0 in SnK (2) | P0 |  |  |  |  |
|  | Int[0] ∧Turn =0 : F∧ T = F  P2 PASS | P2 |  |  |  |  |
|  | P0 in SnK (3) | P0 |  |  |  | DELAY |
|  | P2 in SK get slot 5; slot => 6 | P2 |  |  |  | DELAY |
|  | P0 in SnK (4) | P0 |  |  |  | DELAY |
|  | Int[1] <= false | P2 |  |  |  | DELAY |
|  | P0 in SnK (5) | P0 |  |  |  | DELAY |
|  | P2 in SnK (1) selesai | P2 |  |  |  | Jalur 1 |
|  | P0 in SnK (6) | P0 |  |  |  |  |
|  | Int [1] <= true | P3 |  |  |  |  |
|  | P0 in SNK (7) | P0 |  |  |  | Jalur 1 |
|  | Turn = 0 | P3 |  |  |  |  |
|  | P0 in SNK (8) | P0 |  |  |  |  |
|  | Int[0] ∧Turn =0 : F∧ T = F  P3 PASS | P3 |  |  |  | P3 pass |
|  |  | P0 |  |  |  | P3 pass |
|  |  | P2 |  |  |  | P3 pass |
|  |  | P3 |  |  |  | P3 pass |
|  |  | P2 |  |  |  | P3 pass |
|  |  | P3 |  |  |  | P3 pass |
|  |  | P2 |  |  |  | P3 pass |

Turn 🡸 0

|  |  |  |
| --- | --- | --- |
| Clock | Jalur 0 🡪 PA | Jalur 1 🡪 PB |
|  |  | Interested[1] ⇐ true |
|  | Interested[0] ⇐ true |  |
|  | Turn 🡸 1 |  |
|  |  | Turn 🡸 0 |
|  | While interested [1] and turn = 1 do begin end T Δ F = F 🡪 PA Pass |  |
|  |  | While interested [0] and turn = 0 do begin end  T Δ T = T 🡪 PB  Blocked |
|  | Masuk SK |  |
|  |  | While interested [0] and turn = 0 do begin end  T Δ T = T 🡪 PB  Blocked |
|  | Proses A selesai dengan SK dan mengubah  🡪 Interested [0] ⇐ false |  |
|  |  | While interested [0] and turn = 0 do begin end  F Δ T = F 🡪 proses B Pass |
|  | Masuk S non K |  |
|  |  | Masuk SK |
|  |  | Proses 1 selesai dengan SK dan mengubah  🡪 Interested [1] ⇐ false |
|  | Masih S non K |  |
|  |  | Masuk S non K dan selesai 🡪 jalur 1 kosong |
|  |  | Jalur 1 Menerima Proses baru  Interested[1] ⇐ true |
|  | Masih S non K |  |
|  |  | Turn 🡸 0 |
|  |  | While interested [0] and turn = 0 do begin end  F Δ T = F 🡪 proses di jalur 1 Pass (bisa melanjutkan) |
|  |  |  |
|  |  |  |

Kuliah 7

**Semaphore**

*Deskripsi Semaphore*

Semaphore adalah pendekatan yang dikemukakan Dijkstra.

Prinsip semaphore sebagai berikut :

* Dua proses atau lebih dapat bekerja sama dengan menggunakan penanda – penanda sederhana.
* Proses dipaksa berhenti sampai proses memperoleh penanda tertentu.
* Sembarang kebutuhan koordinasi kompleks dapat dipenuhi dengan stuktur penanda yang sesuai kebutuhannya.
* Variabel khusus untuk penandaan ini disebut semaphore.

*Terdapat dua operasi terhadap semaphore yaitu Down dan Up 🡪 atomik (sebuah procedure yang harus dieksekusi dalam 1 paket waktu 🡪 tidak dapat di-preempt)*

*Semaphore diimplementasikan dengan dukungan sebuah Queue*

**Operasi Down**

* Operasi ini menurunkan nilai semaphore.
* Jika nilai semaphore menjadi nonpositif maka proses di-blocked.

|  |
| --- |
| Procedure Down ( Var s : semaphore );  Begin  s : = s – 1 ;  If s < 0 Then  Begin  Tempatkan proses pada antrian untuk semaphore s  Proses di – blocked  End;  End; |

**Operasi Up**

Operasi Up menaikkan nilai semaphore, jika nilai semaphore nonpositif atau 0 pindahkan satu proses dari antrian dan menempatkan satu proses ke senarai ready

**Semaphore non prositif atau 0🡪 menunjukkan antrian tidak kosong**

|  |
| --- |
| Procedure Up ( Var s : semaphore )  Begin  s : = s + 1 ;  If s < = 0 Then  Begin  Pindahan satu proses P dari antrian untuk semaphore s  Tempatkan proses P di senarai ready  End;  End; |

**Semaphore 🡪**

**sebuah variabel sederhana yang nilai-nilai nya mengidentifikasi keadaan:**

* **status SK 🡪 kosong atau isi**
* **status Queue 🡪 ada atau tidaknya proses didalam antrian**

**Mutual Exclusion dengan Semaphore**

Adanya semaphore mempermudah persoalan mutual exclusion.

Skema penyelesaian mutual exclusion mempunyai bagan program sebagai berikut:

|  |
| --- |
| Program Mutual – exclusion – with – Semaphore;  Const  N = ?;  **Procedure Down ( Var s : semaphore );**  **Begin**  **s : = s – 1 ;**  **If s < 0 Then**  **Begin**  **Tempatkan proses pada antrian untuk**  **semaphore s. Proses di – blocked,**  **menunggu diaktifkan oleh proses lain**  **melalui prosedur UP**  **End-if;**  **End**  **Procedure Up ( Var s : semaphore )**  **Begin**  **s : = s + 1 ;**  **If s < = 0 Then**  **Begin**  **Pindahkan satu proses P dari antrian untuk**  **semaphore S**  **Tempatkan proses P di senarai ready**  **End-if**  **End**  Var  s : semaphore;  Procedure enter – critical – section;  { Mengerjakan kode – kode kritis}  Procedure enter – noncritical – section;  { Mengerjakan kode – kode non kritis }  Procedure Proses ( i : integer );  Begin  Down ( s );  enter – critical – section;  Up ( s );  enter – noncritical – section;  End;  Begin  s : = 1 ; { init atau nilai awal semaphore}  i := 0  Repeat  Parbegin  Proses ( i ) ;  i = (i + 1) mod N  Parend  Forever  End. |

**Contoh**

**Kasus Printer Daemon, SK adalah :**

*Procedure enter – critical – section;*

*{ Mengerjakan kode – kode kritis}*

***Begin***

***dapat\_slot := next\_slot***

***next\_slot := dapat\_slot +1***

***end***

**Data proses :**

|  |  |  |
| --- | --- | --- |
| **Id proses** | **Waktu datang** | **Jumlah siklus waktu store berkas** |
| **P0** | **0** | **2** |
| **P1** | **1** | **2** |
| **P2** | **4** | **2** |
| **P3** | **20** | **3** |

**S = 1 nomor Slot berikutnya 🡺 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Clock** | **Action** |  | **P0** | **P1** | **P2** | **P3** |
|  | Down(s)  S= 0 🡺 P0 Pass | P0 | datang |  |  |  |
|  | Down(s)  S=-1 🡪 P1 masuk Queue | P1 | Down | Datang |  |  |
|  | P0 in SK  Dapat slot 3 | P0 | In SK | in queue |  |  |
|  | Next Slot = 4 | P0 | In SK | In queue |  |  |
|  | Down(s)  S = -2 🡪 P2 masuk Queue | P2 | Selesai  SK | In queue | Datang |  |
|  | Up(s)  S= -1 🡪 Q ≠ nihil 🡪  Bebaskan satu proses 🡪 P1 | P0 | Up | Keluar dari Queue | In queue |  |
|  | P1 in SK  Dapat slot 4 | P1 | In SNK | In SK | In queue |  |
|  | P0 in SNK  Store (1) | P0 | In SNK | In SK | In queue |  |
|  | Next slot = 5 | P1 | In SNK | In SK | In  Queue |  |
|  | P0 in SNK  Store (2) dan selesai | P0 | In SNK  selesai | Selesai SK |  |  |
|  | Up (s)  S = 0 🡪 Q ≠ nihil 🡪  Bebaskan satu proses 🡪 P2 | P1 |  | Up | Keluar dari Queue |  |
|  | P2 in SK  Dapat slot 5 | P2 |  | In SNK |  |  |
|  | P1 in SNK  Store (1) | P1 |  | In SNK |  |  |
|  | Next slot = 6 | P2 |  | In SNK |  |  |
|  | P1 in SNK  Store (2) dan selesai | P1 |  | In SNK  selesai |  |  |
|  | Up(s)  S = 1 🡺 Q = nihil | P2 |  |  |  |  |
|  | P2 in SNK  Store (1) | P2 |  |  |  |  |
|  | P1 in SNK  Store (2) dan selesai | P2 |  |  |  |  |
|  | Idle |  |  |  |  |  |
|  | Idle |  |  |  |  |  |
|  | P3 | P3 |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

* Sebelum masuk critical section, proses melakukan Down

🡪 output dari down : pass atau terblock.

* + Bila berhasil maka proses masuk critical section .
  + Bila tidak berhasil maka proses di – blocked pada semaphore itu.
* Proses yang di – blocked akan dapat melanjutkan kembali bila proses di critical section keluar dan melakukan operasi Up sehingga menjadikan proses yang di – blocked ready dan berlanjut sehingga operasi Down – nya berhasil.