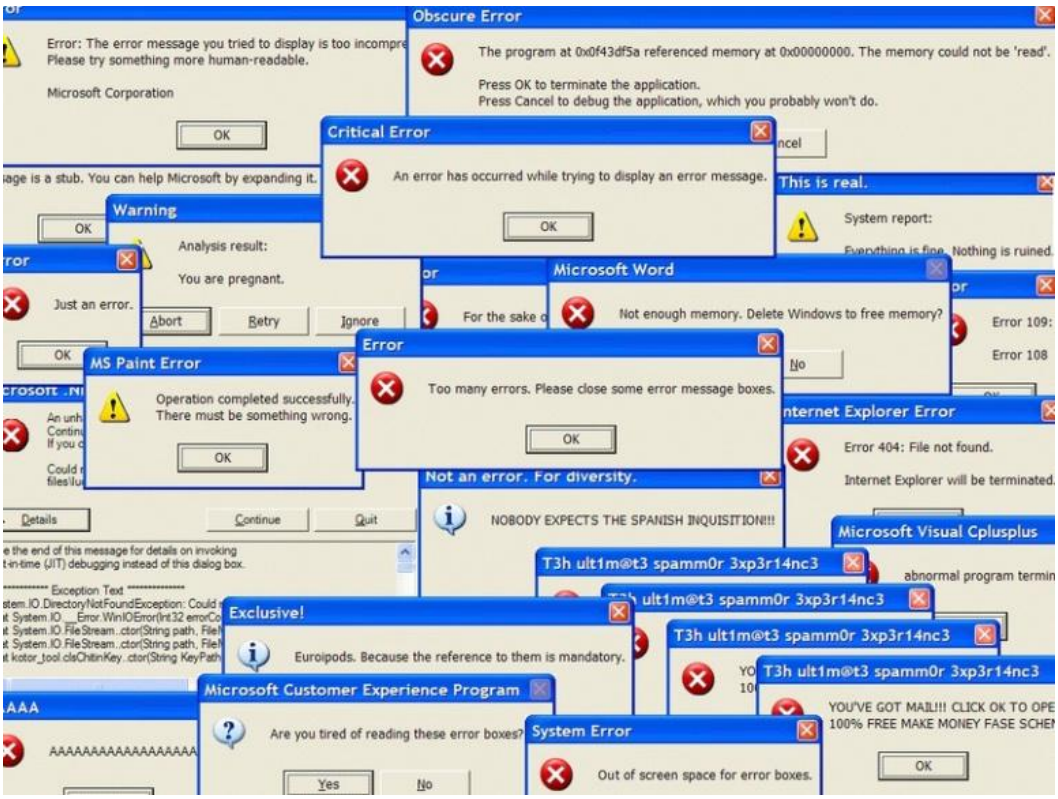
MODUL VIII
CCS 210 SISTEM OPERASI

Judul	KONKURENSI	
Penyusun	**Distribusi**	**Perkuliahan**
Nixon Erzed	**FASILKOM** **UNIVERSITAS ESA UNGGUL**	Pertemuan – VIII OL – 8

Materi

1. Pengantar Konkurensi
2. Mutex



KONKURENSI DASAR

(Konkurensi 2 proses)

Konkurensi merupakan landasan umum perancangan sistem operasi. Proses-proses disebut konkuren jika proses-proses berada pada saat yang sama. Dikatakan sebagai landasan umum perancangan sistem operasi karena dalam menciptakan suatu sistem operasi, sistem operasi tersebut umumnya harus bisa menjalankan beberapa proses (lebih dari satu proses) pada saat yang bersamaan.

Pada proses-proses yang konkuren atau berada pada saat yang bersamaan, terdapat beberapa masalah yang harus diselesaikan

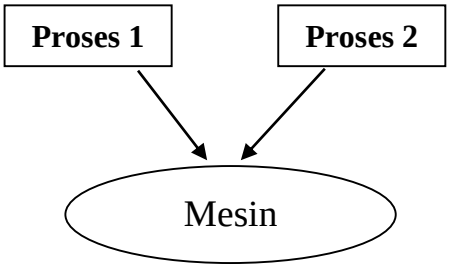
Konkurensi → persoalan proses-proses konkuren

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

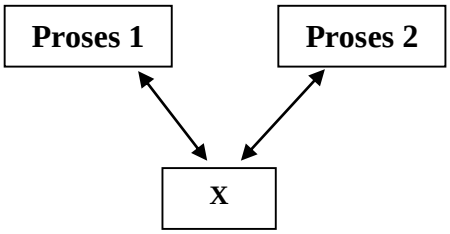
Terdapat beberapa kemungkinan keadaan proses konkuren, yaitu :

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.



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



Ilustrasi masalah

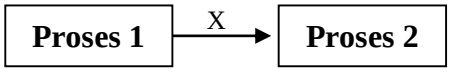
Misalkan proses 1 dan proses 2 bekerja dengan nilai kecepatan yang dibaca dari variabel x, proses yang membaca variabel berkewajiban untuk meng-update nilai x menjadi 2 kalinya.

Misalkan nilai awal *x = 10*

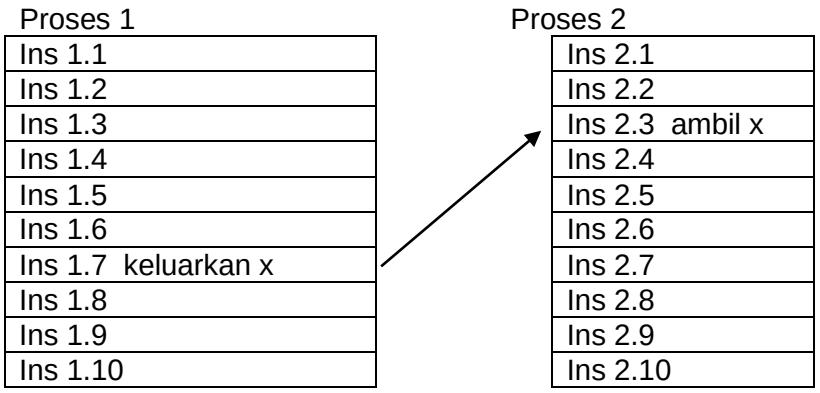
Sehingga proses yang pertama kali mendapatkan x akan bekerja dengan kecepatan 10
Jika proses 1 yang mendapatkan kesempatan awal membaca x dan kemudian mengubahnya menjadi 20
→ *proses 1 bekerja dengan kecepatan 10*
→ *proses 2 mendapatkan x = 20, bekerja dengan kecapatan 20 dan mengupdate x =40*

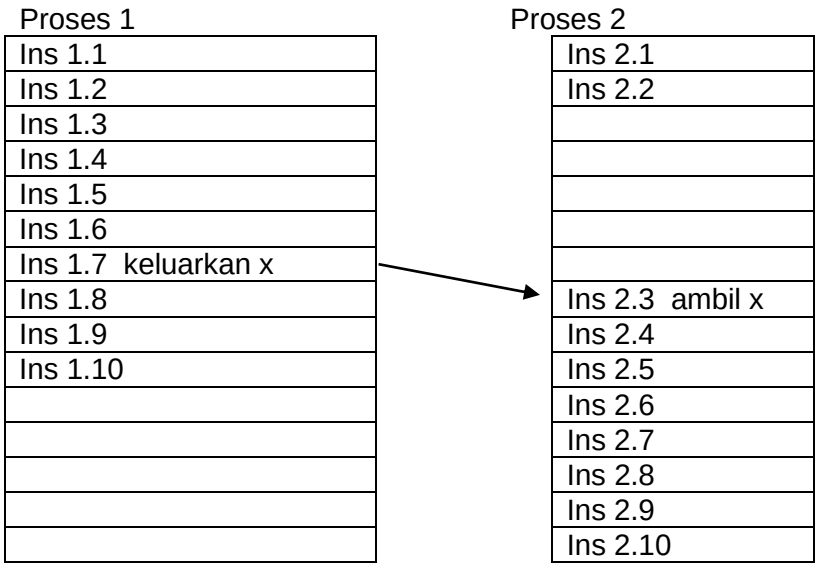
Jika proses 2 yang mendapatkan kesempatan awal membaca x dan kemudian mengubahnya menjadi 20
→ *proses 2 bekerja dengan kecepatan 10*
→ *proses 1 mendapatkan x = 20 dan bekerja dengan kecapatan 20 dan mengupdate x =40*

3. Saling berinterakasi secara langsung, terdapat komunikasi antar proses
   (→ output P_1 menjadi input P_2)
   → suatu proses tergantung pada proses lainnya



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





Persoalan-persoalan yang muncul pada proses konkuren

1. Starvation

   Startvation adalah keadaan dimana pemberian akses bergantian terus menerus, dan ada suatu proses yang tidak mendapatkan gilirannya. Juga dapat dimaksudkan bahwa kondisi bila beberapa proses-proses menunggu alokasi sumber daya sampai tak berhingga, sementara proses-proses lain dapat memperoleh alokasi sumber daya.

   Hal ini disebabkan bias pada kebijaksanaan atau strategi alokasi sumber daya. Kondisi seperti ini harus dihindari pada sistem operasi karena tidak adil, tapi dikehendaki penghindaran dilakukan seefisien mungkin. Penanganan ini merupakan persoalan yang sulit untuk menemukan kriteria yang benar, adil dan efesien dalam suatu strategi Sistem Operasi.

   Perhatikan contoh berikut:

   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 .

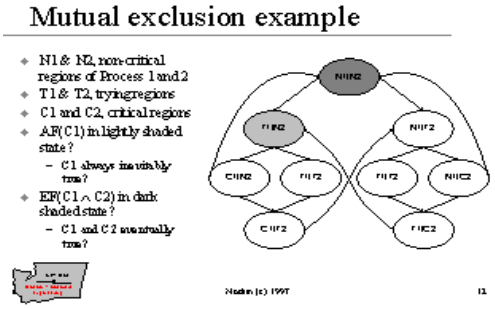
   Pemulihan starvation → digunakan penjadwal jangka panjang (yang mengintervensi antrian agar proses terkucil diprioritaskan)

2. MUTEX (Mutual Exclusion)

   Mutual exclusion adalah jaminan hanya satu proses yang mengakses sumber daya pada satu interval waktu tertentu, dimana sumber daya tersbut tidak dapat dipakai bersama pada saat bersamaan.

   → Pemakaian sumber daya bersama (variabel bersama)
   → 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 :

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. 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
yaitu isi kamar menjadi tidak konsisten (tidak sesuai dengan prediksi)

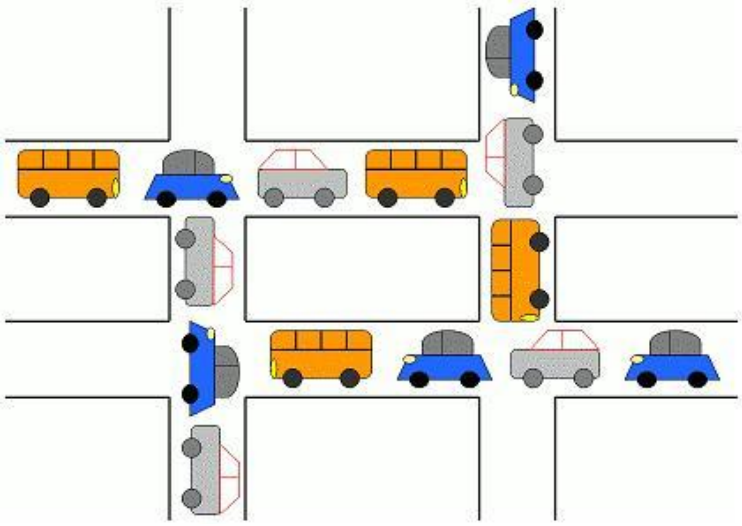
Mutex → menjamin akses ekslusif terhadap seksi kritis

3. DEAD LOCK

→ Persaingan mendapatkan sumber daya

Deadlock adalah suatu kondisi dimana dua proses atau lebih tidak dapat meneruskan eksekusinya oleh pemroses. Pada umumnya deadlock terjadi karena proses mengalami startvation, yaitu suatu job yang sedang dieksekusi dan eksekusi job tersebut tidak ada hentinya, tidak diketahui kapan berhentinya proses tersebut atau bahkan job yang antri bisa dikatakan mempunyai status mati, padahal proses-proses lain sedang menunggu sumber daya proses.

Kondisi Deadlock merupakan kondisi terparah karena banyak proses dapat terlibat dan semuanya tidak dapat mengakhiri prosesnya secara benar.



- 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

4. Sinkronisasi proses

Pada proses-proses yang berinteraksi secara langsung, terjadi komunikasi berupa output suatu proses menjadi input proses lainnya.

→ terdapat ketentuan predesesor – suksesor

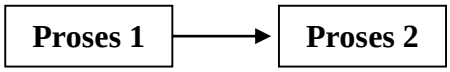
Jika proses-proses hadir pada saat yang sama dan meminta layanan prosesor : *harus disinkronkan*

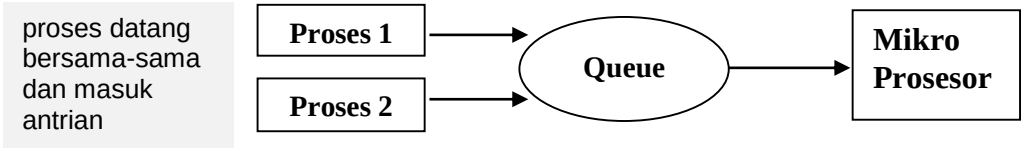
Proses predesesor harus dilayani terlebih dahulu, jika gagal dijamin → tidak sinkron

Contoh :

Output P1 menjadi masukan P2

ketentuan :





→ 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 →deadlock	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.

Keyword → 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  {Program Utama Give_file_to_Spoiler}
    in : = 0;
    REPEAT
        PARBEGIN
            PROSES A;
            PROSES B;
        PAREND
    FOREVER
END
```

Proses parallel A dan B dapat juga ditulis sebagai berikut

```
Procedure Proses A;
Var
    next_Free_slot : Integer;
Begin
    next_free_slot : = in;
    in : = next_free_slot + 1;
    Store ( berkas A, next_free_slot );
End-proses A;

Procedure Proses B;
Var
    next_free_slot : Integer
Begin
    next_free_slot : = in;
    in : = next_free_slot + 1;
    Store ( berkas B, next_free_slot );
End-proses B
```

Skenario yang Membuat Situasi Kacau

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

Perhatikan tabel penelusuran berikut ini ..

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.
2. Proses yang berada di *noncritical section*, dilarang mem – *blocked proses* – proses lain yang ingin masuk *critical section.*
3. Harus dijamin proses yang ingin masuk critical section tidak menunggu selama waktu yang tak berhingga. Atau tak boleh terdapat *deadlock* atau *startvation.*
4. Ketika tidak ada proses pada *critical section* maka proses yang ingin masuk *critical section* harus diijinkan masuk tanpa waktu tunda.
5. 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.

2. Metoda Pergantian Secara Ketat

   Menggunakan Variabel Turn dengan nilai yang berbeda antara suatu proses dengan proses lainnya.
   Masuknya proses-proses terjadi pada prosedur.

3. Algoritma Peterson
4. Semaphore
   Semaphore adalah teknik penyelesaian Mutex dengan memanfaatkan Queue

Metoda dengan Variabel Lock Sederhana (Metoda Naif)

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.

```
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  {Program Utama}

   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

Clock	Proses A	Proses B
1	**While lock** <> 0 **Do Begin End;** **(Lock = 0)<> 0 false → P_A PASS**	
2	Lock := 1 (mengunci)	
3		**While lock** <> 0 **Do Begin End;** *(**P_B** blocked)* **(Lock=1) <> 0 → benar looping**
4		**While lock** <> 0 **Do Begin End;** *(**P_B** 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;** *(**P_B** PASS)*

Skenario Gagal

Perhatikan masalah berikut :

→ 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.

Perhatikan contoh berikut :

Clock ke-1 Proses A dapat alokasi waktu proses
tapi pada Clock ke-2 Proses B dapat alokasi waktu proses
Akibatnya → Proses A dan Proses B sama-sama sudah berada diseksi kritis
dan → terjadi kekacauan karena tidak bisa dijamin hanya 1 proses yang masuk seksi kritis pada satu saat

Perhatikan tabel penelusuran dihalaman berikut ini ..

Clock	Proses A	Proses B
1.	**While lock** <> 0 **Do Begin End; PASS**	
2.		**While lock** <> 0 **Do Begin End;** *(PASS)*
3.	Lock := 1 (mengunci)	
4.		Lock := 1 (mengunci ULANG) → KEKACAUAN TERJADI
5.		Mengerjakan Seksi Kritis (baca IN)
6.	Mengerjakan Seksi Kritis (baca IN)	
7.	Mengerjakan seksi kritis (Update IN)	
8.		Mengerjakan seksi kritis (Update IN)
9.	Lock := 0 (melepas kunci)	
10.		