Mata Kuliah : Sistem Operasi

Kode MK : IT-012336

8

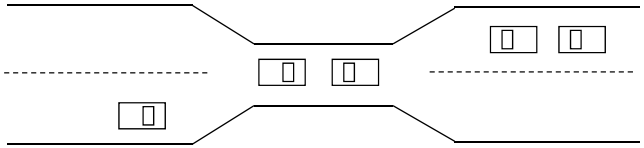
Deadlock

Tim Teaching Grant
Mata Kuliah Sistem Operasi

Masalah Deadlock

- Sekumpulan proses sedang blocked karena setiap proses sedang menunggu (antrian) menggunakan “resources” yang sedang digunakan (hold) oleh proses lain.
- Contoh:
  - OS hanya mempunyai akses ke 2 tape drives.
  - P1 dan P2 memerlukan 2 tape sekaligus untuk mengerjakan task (copy).
  - P1 dan P2 masing-masing hold satu tape drives dan sedang blocked, karena menunggu 1 tape drives “available”.

Contoh Persimpangan Jalan



- Hanya terdapat satu jalur
- Mobil digambarkan sebagai proses yang sedang menuju sumber daya.
- Untuk mengatasinya beberapa mobil harus *preempt* (mundur)
- Sangat memungkinkan untuk terjadinya *starvation* (kondisi proses tak akan mendapatkan sumber daya).

Resource-Allocation Graph

Sekumpulan vertex V dan sekumpulan edge E

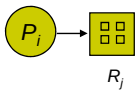
- V dipartisi ke dalam 2 tipe
  - $P = \{P_1, P_2, \ldots, P_n\}$, terdiri dari semua proses dalam sistem.
  - $R = \{R_1, R_2, \ldots, R_m\}$, terdiri dari semua sumberdaya dalam sistem
- request edge/permintaan edge : arah edge $P_1 \rightarrow R_j$
- assignment edge/penugasan edge – arah edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

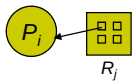
- Process

- Resource Type with 4 instances

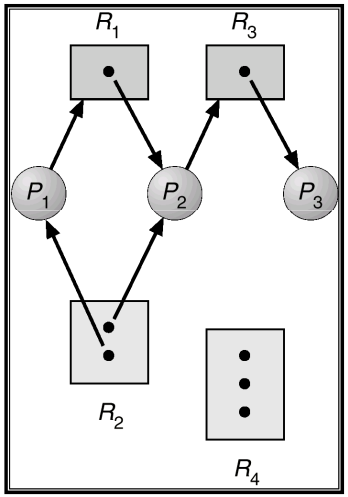
- P_i requests instance of R_j



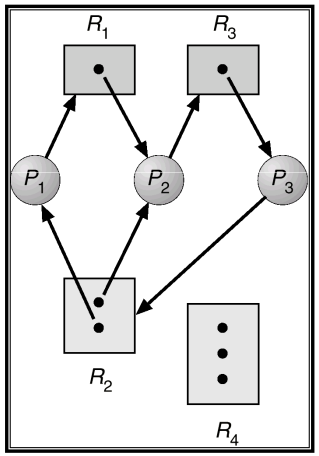
- P_i is holding an instance of R_j



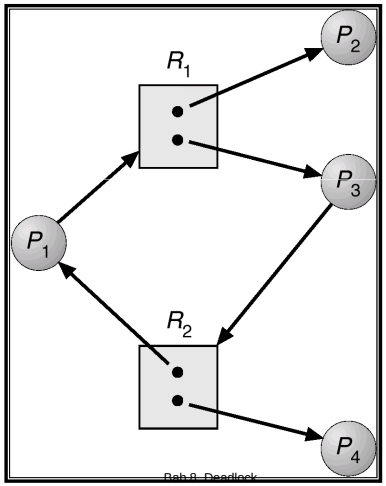
Contoh Resource Allocation Graph



Graf Resource Allocation Dengan Deadlock



Graf Resource Allocation dengan Cycle Tanpa Deadlock



Kondisi yang Diperlukan untuk Terjadinya Deadlock

- **Mutual Exclusion**
  - **Serially-shareable resources (mis. Buffer)**
  - Contoh: Critical section mengharuskan mutual exclusion (termasuk resource), sehingga potensi proses akan saling menunggu (blocked).
- **Hold & wait :**
  - **Situasi dimana suatu proses sedang hold suatu resource secara eksklusif dan ia menunggu mendapatkan resource lain (wait).**

Kondisi yang Diperlukan untuk Terjadinya Deadlock (cont.)

- **No-Preemption Resouce :**
  - Resource yang hanya dapat dibebaskan secara sukarela oleh proses yang telah mendapatkannya
  - Proses tidak dapat dipaksa (pre-empt) untuk melepaskan resource yang sedang di hold
- **Circular wait**
  - Situasi dimana terjadi saling menunggu antara beberapa proses sehingga membentuk waiting chain (circular)
  - Misalkan proses (P0, P1, .. Pn) sedang blok menunggu resources: P0 menunggu P1, P1 menunggu P2, .. dan Pn menunggu P0.

Metode Penanganan Deadlock

- **Deadlock Prevention**: Pencegahan adanya faktor-faktor penyebab deadlock
- **Deadlock Avoidance**: Menghindari dari situasi yang potensial dapat mengarah menjadi deadlock
- **Deadlock Detection**: Jika deadlock ternyata tidak terhindari maka bagaimana mendeteksi terjadinya deadlock, dilanjutkan dengan penyelamatan (recovery).

Deadlock Prevention

- Pencegahan: Faktor-faktor penyebab deadlock yang harus dicegah untuk terjadi
- 4 faktor yang harus dipenuhi untuk terjadi deadlock:
  - Mutual Exclusion: pemakaian resources.
  - Hold and Wait: cara menggunakan resources.
  - No preemption resource: otoritas/hak.
  - Circular wait: kondisi saling menunggu.
- Jika salah satu bisa dicegah maka deadlock pasti tidak terjadi!

Deadlock Prevention (1)

- **Tindakan preventif:**
  - **Batasi pemakaian resources**
  - **Masalah: sistim tidak efisien, tidak feasible**
- **Mutual Exclusion:**
  - tidak diperlukan untuk shareable resources
  - read-only files/data : deadlock dapat dicegah dengan tidak membatasi akses (not mutually exclusive)
  - tapi terdapat resource yang harus mutually exclusive (printer)

Deadlock Prevention (2)

- **Hold and Wait**
  - Request & alokasi dilakukan saat proses start (dideklarasikan dimuka program)
  - Request hanya bisa dilakukan ketika tidak sedang mengalokasi resource lain; alokasi beberapa resource dilakukan sekaligus dalam satu request
  - Simple tapi resource akan dialokasi walau tidak selamanya digunakan (low utilization) serta beberapa proses bisa mengalami starvation

Deadlock Prevention (3)

- **Mencegah Circulair Wait**
  - Pencegahan: melakukan total ordering terhadap semua jenis resource
  - Setiap jenis resource mendapatkan index yang unik dengan bilangan natural: 1, 2, . .
    - Contoh: tape drive=1, disk drive=5, printer=12
  - Request resource harus dilakukan pada resource-resource dalam urutan menaik (untuk index sama - request sekaligus)
  - Jika Pi memerlukan Rk yang berindeks lebih kecil dari yang sudah dialokasi maka ia harus melepaskan semua resource Rj yang berindeks >= Rk

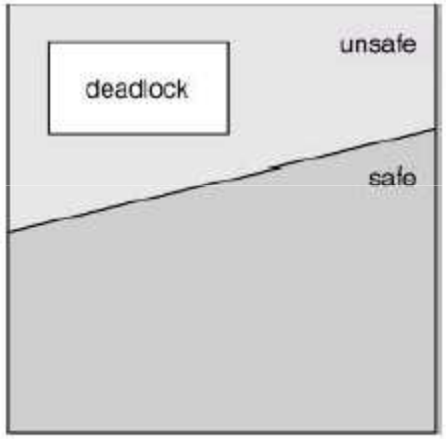
Deadlock Prevention (4)

- **Mencegah No-Preemption**
  - Jika proses telah mengalokasi resource dan ingin mengalokasi resource lain – tapi tidak diperoleh (wait) : maka ia melepaskan semua resource yang telah dialokasi.
  - Proses akan di-restart kelak untuk mencoba kembali mengambil semua resources

Deadlock Avoidance

- **Pencegahan:**
  - **Apabila di awal proses; OS bisa mengetahui resource mana saja yang akan diperlukan proses.**
  - **OS bisa menentukan penjadwalan yang aman (“safe sequence”) alokasi resources.**
- **Model:**
  - Proses harus menyatakan max. jumlah resources yang diperlukan untuk selesai.
- Algoritma “deadlock-avoidance” secara dinamik akan memeriksa alokasi resource apakah dapat mengarah ke status (keadaan) tidak aman (misalkan terjadi circulair wait condition)
- Jadi OS, tidak akan memberikan resource (walaupun available), kalau dengan pemberian resource ke proses menyebabkan tidak aman (unsafe).

Safe, unsafe , deadlock state



Safe state

- Prasyarat:
  - Proses harus mengetahui max. resource yang diperlukan (upper bound) => asumsi algoritma.
  - Proses dapat melakukan hold and wait, tapi terbatas pada sekumpulan resource yang telah menjadi “kreditnya”.
- Setiap ada permintaan resource, OS harus memeriksa
  - “jika resource diberikan”, dan terjadi “worst case” semua proses melakukan request “max. resource”
  - Terdapat “urutan yang aman” dari resources yang available, untuk diberikan ke proses, sehingga tidak terjadi deadlock.

Kondisi Safe

- Resources: 12 tape drive.

User	Max. need	Allocation
U1	4	1
U2	6	4
U3	8	5

A (Available): 12 - 10 = 2
Safe sequnce:
2 tape diberikan ke U2,
U2 selesai => Av = 6,
Berikan 3 tape ke U1,
Berikan 3 tape ke U3.
No deadlock.

Kondisi Unsafe

- Resources: 12 tape drive.

User	Max. need	Allocation
U1	4	1
U2	6	4
U3	8	6

A (Available): 12 - 11 = 1

Terdapat 1 tape available,
sehingga dapat terjadi Deadlock.

Algoritma Banker's

- Proses harus "declare" max. kredit resource yang diinginkan.
- Proses dapat block (pending) sampai resource diberikan.
- Banker's algorithm menjamin sistim dalam keadaan safe state.
- OS menjalankan Algoritma Banker's,
  - Saat proses melakukan request resource.
  - Saat proses terminate atau release resource yang digunakan => memberikan resource ke proses yang pending request.

Algoritma Banker's (2)

- **Metode :**
  1. Scan tabel baris per baris untuk menemukan job yang akan diselesaikan
  2. Tambahkan pada job terakhir dari sumberdaya yang ada dan berikan nomor yang available
- Ulangi 1 dan 2 hingga :
  - Tidak ada lagi job yang diselesesaikan (unsafe) atau
  - Semua job telah selesai (*safe*)

Algoritma Banker's (3)

- Misalkan terdapat: n proses dan m resources.
- Definisikan:
  - **Available**: Vector/array dengan panjang **m**.
    If **available [j] = k**, terdapat **k** instances resouce jenis **Rj** yang dapat digunakan.
  - **Max**: matrix **n x m**.
    If **Max [i,j] = k**, maka proses **Pi** dapat request paling banyak k instances resource jenis Rj.
  - **Allocation**: matrix **n x m**.
    If **Allocation[i,j] = k** maka **Pi** saat ini sedang menggunakan (hold) **k** instances Rj.
  - **Need**: matrix **n x m**.
    If **Need[i,j] = k**, maka **Pi** palaing banyak akan membutuhkan instance Rj untuk selesai.
  - **Need [i,j] = Max[i,j] – Allocation [i,j].**

Algoritma Safety

- Let *Work* and *Finish* be vectors of length *m* and *n*,respectively. Initialize:
  Work* := *Available* // *resource yang free
  Finish* [*i*] = *false* for *i* = 1,3, …, *n.
- Find and *i* such that both: // penjadwalan alokasi resource
  (a) *Finish* [*i*] = *false // asume, proses belum complete*
  (b) *Needi £ Work // proses dapat selesai, ke step 3*
  If no such *i* exists, go to step 4.
- *Work* := *Work* + *Allocation**i*** // proses dapat selesai
  Finish[*i*] := *true*
  go to step 2.
- If *Finish* [*i*] = true for all *i*, then the system is in a safe state.

Algoritma Safety (2)

- Terdapat 3 proses: n = 3, 1 resource: m = 1
- Jumlah resource m = 12.
- Snapshot pada waktu tertentu:

User	Max. need	Allocation
U1	4	1
U2	6	4
U3	8	5

Max: **Allocation:**
[4 **[1**
6 **4**
8] **5]**

Available:
[2]

Algoritma Safety (3)

Need [i,j] = Max[i,j] – Allocation [i,j].

Need: **Max:** **Allocation:**
[3 = **[4** - **[1**
2 **6** **4**
3] **8]** **5]**

Let Need[3]; Max[3]; Aloc[3]; Finish[3]; Work [1];

1. Work = Available; // Work = 2;
   Finish[0]=false, Finish[1]=false, Finish[3]=false;
2. do {
   FlagNoChange = false;
   for I=0 to 2 {
   if ((Finish[I] == false)) && (Need[I] <= Work) {
   Finis[I] = true;
   Work = Work + Aloc[I]; FlagNoChange = true;
   }
   } until (FlagNoChange);

Deadlock Detection

- Mencegah dan menghidari dari deadlock sulit dilakukan:
  - Kurang efisien dan utilitas sistim
  - Sulit diterapkan: tidak praktis, boros resources
- Mengizinkan sistim untuk masuk ke “state deadlock”
  - Gunakan algoritma deteksi (jika terjadi deadlock)
    - Deteksi: melihat apakah penjadwalan pemakaian resource yang tersisa masih memungkinkan berada dalam safe state (variasi “safe state”).
  - Skema recovery untuk mengembalikan ke “safe state”

Single Instance

- Gunakan: resource allocation graph
  - Node mewakili proses, arcus mewakili request dan hold dari resources.
  - Dapat disederhanakan dalam “wait-for-graph”
    - *Pi* → *Pj* if *Pi* is waiting for *Pj.*
  - Secara periodik jalankan algoritma yang mencari cycle pada graph:
    - Jika terdapat siklus (cycle) pada graph maka telah terjadi deadlock.

Recovery dari Deadlock

- Batalkan semua proses deadlock
- Batalkan satu proses pada satu waktu hingga siklus deadlock dapat dihilangkan
- Proses mana yang dapat dipilih untuk dibatalkan ?
  - Proses dengan prioritas
  - Proses dengan waktu proses panjang
  - Sumberdaya proses yang telah digunakan
  - Sumberdaya proses yang lengkap
  - Banyak proses yang butuh untuk ditunda
  - Apakah proses tersebut interaktif atau batch

Recovery dariDeadlock

- Pilih proses – meminimasi biaya
- Rollback – kembali ke state safe, mulai lagi proses dari state tersebut
- Starvation – proses yang sama selalu diambil sebagai pilihan, termasuk rollbak dalam faktor biaya

Pendekatan Kombinasi

- Kombinasi dari tiga pendekatan dasar
  - prevention
  - avoidance
  - detection
- Pemisahan sumberdaya ke dalam hirarki kelas