IF3055 – Manajemen Proses
Deadlock

Henny Y. Zubir
STEI - ITB

Sumber Daya (1)

- **Sumber daya (resource)**: komoditas yg diperlukan oleh proses
- Sumber daya dapat berupa:
  - **serially reusable**, contoh: CPU, memory, ruang disk, perangkat I/O, file

    peroleh → gunakan → lepaskan
  - **consummable** – dibuat/ diperlukan oleh proses, contoh: pesan, buffer informasi, interrupt

    buat → peroleh → gunakan

    sumberdaya habis setelah digunakan, karena itu tidak ada pelepasan sumberdaya

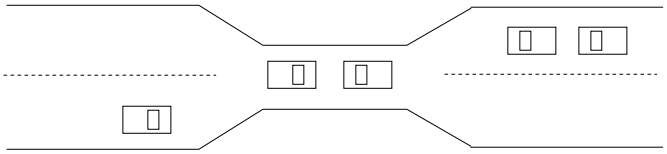
Sumber Daya (2)

- Sumber daya bersifat:
  - **Preemptible**, contoh: CPU, memori utama
  - **non-preemptible**, contoh: tape drives
- Sumber daya bisa digunakan secara:
  - **Bersama**, oleh beberapa proses
  - **Terdedikasi**, secara eksklusif oleh satu proses

Penggunaan Sumber Daya oleh Proses

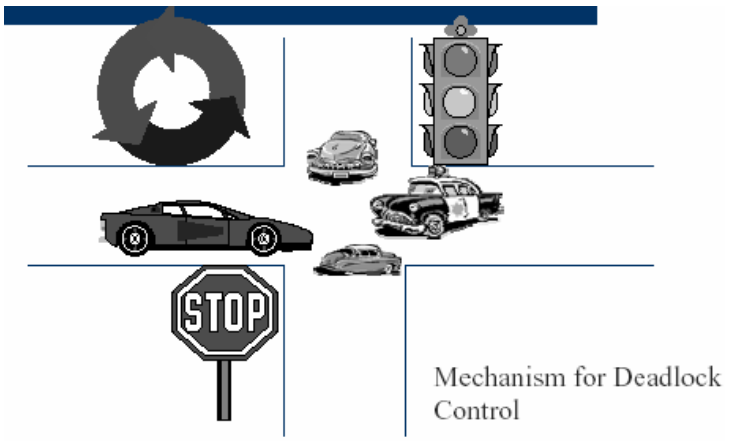
- Urutan event yg dilakukan dalam penggunaan sumber daya:
  1. permintaan sumber daya
  2. penggunaan sumber daya
  3. pelepasan sumber daya
- Harus menunggu jika request ditolak
  - proses yang meminta mungkin diblokir, atau
  - proses gagal dgn mengeluarkan kode kesalahan

Analogi Deadlock: Jembatan



- Lalu lintas hanya bisa satu arah pada satu saat
- Bagian jembatan dpt dianggap sebagai resource
- Jika terjadi deadlock, dapat diselesaikan jika salah satu mobil mundur (preempt atau rollback)
- Beberapa mobil mungkin harus mundur jika terjadi deadlock
- Kemungkinan starvation

Analogi Deadlock: Perempatan



Apa itu Deadlock?

- Definisi Formal:

  sekumpulan proses dikatakan deadlock jika setiap proses menunggu terjadinya event yang hanya bisa disebabkan oleh proses lain yg ada di kumpulan tsb
- Event yg ditunggu biasanya adalah pelepasan sumber daya yg digunakan
- Tidak ada satu pun proses yg bisa …
  - running
  - melepaskan sumber daya
  - dibangunkan
- Deadlock = Starvation?

Kondisi yang Menyebabkan Deadlock

1. Mutual Exclusion

   Proses mengklaim akses eksklusif terhadap sumber daya yang mereka butuhkan
2. Kondisi hold-and-wait

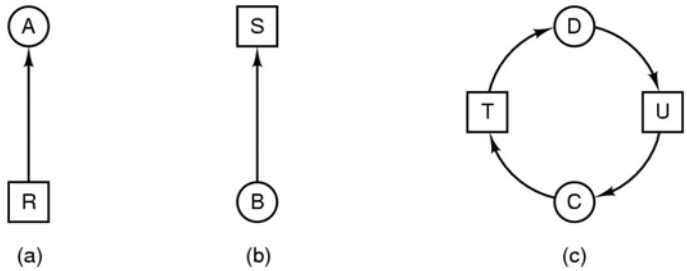
   Proses yg dialokasikan satu sumber daya dapat meminta sumber daya lainnya
3. Kondisi no-preemption

   Sumber daya yang telah dialokasikan tidak dapat diambil dengan paksa
4. Kondisi circular-wait

   Berupa rantai sirkuler yg terdiri dari 2/lebih proses, masing-masing menunggu sumber daya yg dibutuhkan oleh proses lain pd rantai tsb

Pemodelan Deadlock

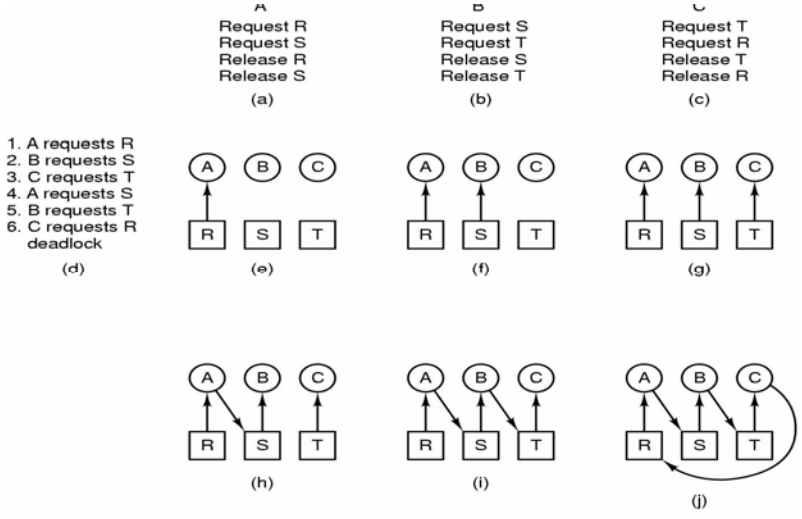
- Dimodelkan dgn graf berarah



(a) (b) (c)

  - Sumber daya R dialokasikan ke proses A
  - Proses B meminta/menunggu sumber daya S
  - Proses C dan D deadlock thd sumber daya T dan U

Pemodelan Deadlock: Contoh (2)



Penanganan Deadlock

- Algoritma Ostrich
  - aAbaikan masalah deadlock sama sekali
- Pencegahan (prevention)
  - merancang sistem sedemikian sehingga deadlock tdk mungkin terjadi (meniadakan salah satu dari 4 kondisi yg menyebabkan deadlock)
- Penghindaran (avoidance)
  - memungkinkan terjadinya deadlock, namun melakukan pencegahan ketika deadlock hampir terjadi
- Pendeteksian (detection) & Pemulihan (recovery)
  - mendeteksi terjadinya deadlock dan proses serta sumber daya mana yang terlibat
  - Setelah deadlock dideteksi, proses yg menyebabkan deadlock diselesaikan shg sumber daya yg diaksesnya bisa digunakan oleh proses lain

Algoritma Ostrich

- Jangan lakukan apa pun, cukup restart sistem
  (ostrich: benamkan kepala ke pasir, dan pura2 tidak ada masalah sama sekali)
- Dilakukan jika:
  - Deadlock jarang terjadi
  - algoritma deadlock lainnya biayanya lebih tinggi
- Diterapkan oleh Windows dan UNIX
- Trade off
  - kenyamanan (convenience) vs keakuratan (correctness)

Pencegahan Deadlock (1)

- Mencegah salah satu dari 4 kondisi yg menyebabkan deadlock
- Mencegah kondisi mutual exclusion
  - penggunaan sumber daya secara eksklusif merupakan fitur penting utk sinkronisasi
  - hindari pengalokasian sumber daya jika tidak benar-benar diperlukan
  - usahakan sesedikit mungkin proses mengklaim sumber daya
  - spooling sumber daya (mis. printer)

Pencegahan Deadlock (2)

- Mencegah kondisi hold-and-wait
  - Proses harus meminta sumber daya sebelum mulai → tidak bisa mulai sebelum semua sumber daya yg dibutuhkan diperoleh
  - Masalah:
    - Sumber daya yg diperlukan mungkin tdk diketahui pada saat proses mulai
    - Mengikat sumber daya yg mungkin akan digunakan oleh proses lain
  - Alternatif:
    - Jika proses membutuhkan suatu sumber daya, lepaskan dulu semua sumber daya yg telah diperolehnya, setelah itu minta kembali semua yg dibutuhkan

Pencegahan Deadlock (3)

- Mencegah kondisi no-preemption
  - Jika proses yg sdg mengakses sumber daya meminta sumber daya lain yg tdk bisa segera dialokasikan utknya, maka semua sumber daya yg sdg dialokasikan pd proses tsb harus dilepas
  - Sumber daya yg di-preempted ditambahkan ke sumber daya yg ditunggu oleh proses
  - Proses akan di-restart hanya jika proses tsb dpt memperoleh semua sumber daya yg dibutuhkan
  - Masalah: dapat menimbulkan starvation

Pencegahan Deadlock (4)

- Minta satu sumber daya pd satu saat; lepaskan sumber daya yg sedang diakses jika meminta sumber daya berikutnya
- Pengurutan sumber daya secara global
  - Permintaan harus dilakukan secara terurut
  - Req(sumberdaya1), req(sumberdaya2)..
  - Mengapa tidak terjadi circular wait?

Pencegahan Deadlock: Rangkuman

Kondisi	**Solusi**
Mutual Exclusion	Spool semuanya
Hold-and-wait	Minta semua sumber daya di awal
No-preemption	Ambil sumber daya secara paksa
Circular Wait	Beri nomor urut sumber daya

Pencegahan Deadlock: Two-Phase Locking

- Fase 1
  - Proses mencoba mengunci semua record yg diperlukan, satu per satu
  - Jika record yg diperlukan dikunci, mulai kembali
  - (tidak ada aktifitas riil yg dilakukan pd fase 1)
- Jika fase 1 berhasil, mulai fase 2
  - Lakukan update
  - Lepaskan kunci
- Mirip dgn meminta semua sumber daya sekaligus
- Algoritma berfungsi jika programmer dapat mengatur
  - berhenti dan restart program

Penghindaran Deadlock (1)

- Sistem perlu memiliki informasi awal mengenai kebutuhan sumber daya
  - Tiap proses menyatakan kebutuhan maksimum tiap jenis sumber yg dibutuhkan
  - Algoritma deadlock-avoidance secara dinamis memeriksa state alokasi sumber daya untuk menjamin tdk terjadinya kondisi circular-wait
  - State alokasi sumber daya didefinisikan oleh banyaknya sumber daya yg tersedia dan yg dialokasikan, dan permintaan sumber daya maksimum oleh proses

Penghindaran Deadlock: Safe State

- Jika proses meminta sumber daya yg tersedia, sistem harus memutuskan apakah alokasi sumber daya ini akan menghasilkan safe state
- Sistem berada pada **safe state** jika terdapat urutan event yg tidak menyebabkan deadlock jika semua proses meminta sumber daya maksimum sekaligus
- Logikanya:
  - Jika sistem berada pd safe state → no deadlock
  - Jika sistem berada pd unsafe state → kemungkinan deadlock
  - Pencegahan: sistem tidak masuk ke unsafe state

Pencegahan Deadlock: Algoritma Banker (1)

- Diketahui:
  - maxc[i, j] = maksimum klaim utk Rj oleh pi
  - alloc[i, j] = banyak unit Rj yg dialokasikan ke pi
- Dapat menghitung banyaknya unit Rj yg tersedia:

  avail[j] = cj - $\sum_{0 \le i < n}$ alloc[i,j]
- Menentukan jika state aman/tidak berdasarkan informasi ini

Pencegahan Deadlock: Algoritma Banker (2)

- Salin tabel alloc[i,j] ke alloc'[i,j]
- Jika diketahui C, maxc, and alloc', hitung vektor sumber daya yg tersedia
- Hitung pi: maxc[i,j] – alloc'[i,j] ≤ avail[j] untuk 0 ≤ j < m and 0 ≤ i < n
  - Jika pi tidak ada, unsafe state
  - Jika alloc'[i,j]=0 untuk semua i dan j, safe state
- Set alloc'[i,j] ke 0; dealokasi semua sumber daya yg dimiliki pi; kembali ke langkah 2

Pencegahan Deadlock: Algoritma Banker (3)

- Contoh 1:

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

C = <8, 5, 9, 7>

•Compute total allocated

•Determine available units

avail = <8-7, 5-3, 9-7, 7-5>
= <1, 2, 2, 2>

•Can anyone's maxc be met?

maxc[2,0]-alloc'[2,0] = 5-4 = 1≤1 = avail[0]
maxc[2,1]-alloc'[2,1] = 1-0 = 1≤2 = avail[1]
maxc[2,2]-alloc'[2,2] = 0-0 = 0≤2 = avail[2]
maxc[2,3]-alloc'[2,3] = 5-3 = 2≤2 = avail[3]

•P_2 can exercise max claim

avail[0] = avail[0]+alloc'[2,0] = 1+4 = 5
avail[1] = avail[1]+alloc'[2,1] = 2+0 = 2
avail[2] = avail[2]+alloc'[2,2] = 2+0 = 2
avail[3] = avail[3]+alloc'[2,3] = 2+3 = 5

Pencegahan Deadlock: Algoritma Banker (4)

- Contoh 2:

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	0	0	0	0
p_3	0	2	1	0
p_4	1	0	3	0
Sum	3	3	7	2

C = <8, 5, 9, 7>

•Compute total allocated

•Determine available units

avail = <8-7, 5-3, 9-7, 7-5>
= <5, 2, 2, 5>

•Can anyone's maxc be met?

maxc[4,0]-alloc'[4,0] = 5-1 = 4≤5 = avail[0]
maxc[4,1]-alloc'[4,1] = 0-0 = 0≤2 = avail[1]
maxc[4,2]-alloc'[4,2] = 3-3 = 0≤2 = avail[2]
maxc[4,3]-alloc'[4,3] = 3-0 = 3≤5 = avail[3]

•P_4 can exercise max claim

avail[0] = avail[0]+alloc'[4,0] = 5+1 = 6
avail[1] = avail[1]+alloc'[4,1] = 2+0 = 2
avail[2] = avail[2]+alloc'[4,2] = 2+3 = 5
avail[3] = avail[3]+alloc'[4,3] = 5+0 = 5

Pendeteksian Deadlock

- Deteksi terjadinya deadlock (secara periodik atau acak), kemudian pulihkan
- Bisa lebih bebas dlm pengalokasian sumber daya
- Tidak ada klaim maksimum, safe/unsafe states
- Membedakan antara:
  - Sumber daya serially reusable: unit harus dilepaskan sebelum dialokasikan
  - Sumber daya consumable: tidak pernah melepaskan sumber daya;