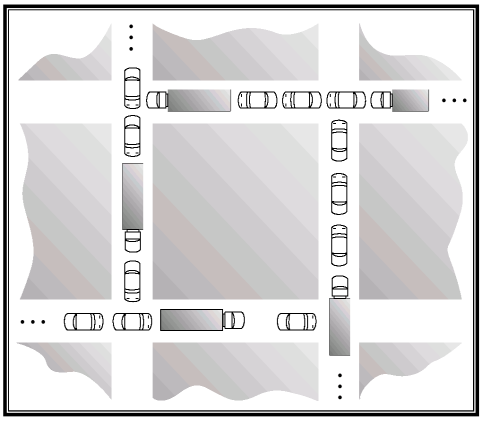
Sistem Operasi Komputer

Pertemuan VII – Deadlock

Pembahasan Deadlock

- Model sistem
- Karakteristik deadlock
- Metode penanganan deadlock
- Deadlock prevention (mencegah)
- Deadlock avoidance (menghindari)
- Deadlock detection (deteksi)
- Penyembuhan dari deadlock (deadlock recovery)
- Solusi kombinasi dalam penanganan deadlock

Contoh Nyata Sehari-hari



Solusi:

• Traffic light, yang mengijinkan arus ke satu arah atau ke arah lainnya dalam suatu waktu tertentu

• Semua mobil hanya boleh belok kiri / kanan

1. Hanya satu mobil boleh menempati setiap persimpangan pada suatu waktu (mutual exclusion)
2. Mobil boleh diam di persimpangan ketika menunggu untuk sampai ke persimpangan berikutnya (hold and wait)
3. Mobil tidak dapat dipindahkan dari tempatnya pada arus lalulintas, hanya dapat jalan ke depan (no preemption)
4. Kumpulan mobil dalam situasi deadlock termasuk juga mobil yang ada di tengah persimpangan (circular waiting)

Masalah deadlock

- Sekumpulan proses yang diblok, dimana setiap proses memegang satu resource dan menunggu resource lain dari proses dalam kumpulan proses yang sedang diblok tersebut, biasanya dari proses-proses atau resource yang *non-preemptive*
- Contoh: sistem memiliki 2 tape drive
  - P1 dan P2 memegang satu tape
  - P1 memerlukan tape yang dipegang P2
  - P2 memerlukan tape yang dipegang P1
- Contoh: Semaphore A dan B, inisialisasi 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*
…	…
signal(A)	*signal(B)*
signal(B)	*signal(A)*

Model Sistem

- Resource R1, R2, …, Rn

  Fisik: CPU cycles, memory space, perangkat I/O

  Logikal: files, semaphores, monitor
- Setiap resource Ri, terdiri atas sejumlah Wi perangkat
- Setiap proses memakai suatu resource, dengan urutan penggunaan
  - Request (system call)
  - Use
  - Release (system call)

Karakteristik deadlock

Kondisi-kondisi penimbul deadlock
(harus terjadi simultan keempatnya):

- Mutual exclusion: jika suatu proses menggunakan suatu resource, tidak ada proses lain yang boleh menggunakan resourse tersebut
- Hold and wait: pada saat suatu proses mengakses suatu resource, proses tersebut dapat meminta ijin untuk mengakses resource lain
- No preemption: jika suatu proses meminta ijin untuk mengakses resource, sementara resource tidak tersedia, maka permintaan tidak dapat dibatalkan
- Circular wait: jika proses Pi sedang mengakses resouce Ri, dan meminta ijin untuk mengakses resource Rj, dan pada saat bersamaan proses Pj sedang mengakses Rj dan minta ijin untuk mengakses resource Ri

Resource Allocation Graph (1)

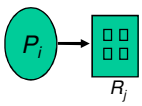
- Sekumpulan simpul V (vertex) dan arah E (edge)
  - V dikelompokkan menjadi
    - P = {P1, … , Pn}
    - R = {R1, …, Rm}
  - Request edge – arah dari Pi → Rj
  - Assignment edge – arah dari Rj → Pi

Resource Allocation Graph (2)

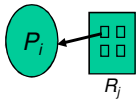
- Process

- Resource Type dengan 4 instances

- P_i meminta R_j

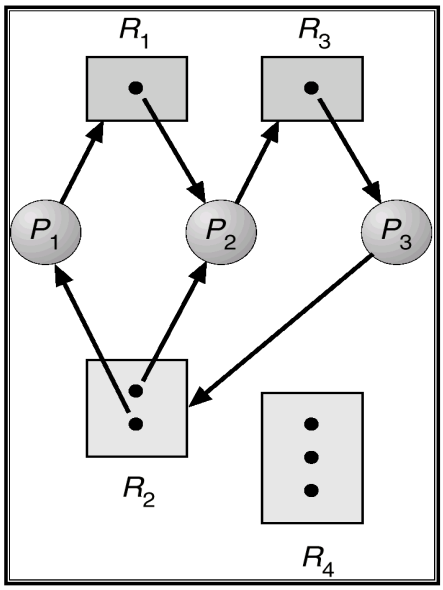


- P_i memegang satu instance dari R_j



Contoh Resource Alocation Graph

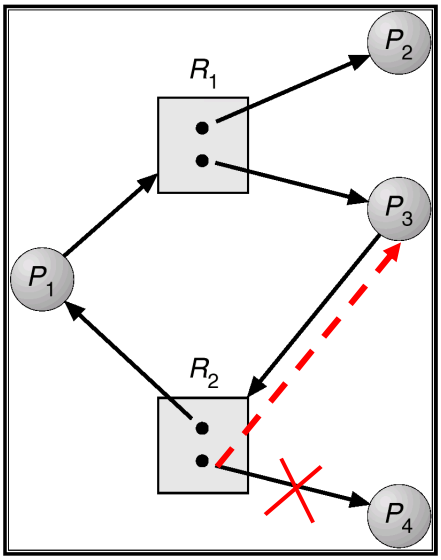
• Siklus → deadlock mungkin terjadi.
• Jika hanya satu instance per resource, maka pasti terjadi deadlock
• Jika ada lebih dari satu instance, terjadi deadlock jika tidak ada proses yang dapat melepaskan resource untuk dialokasikan ke proses lainnya



Terjadi deadlock

P2 → P3 → P1 → P2

RAG dengan siklus tanpa deadlock



Siklus
P1 → R1 → P3 → R2 → P1

Siklus dapat diputuskan, dengan:
membebaskan satu resource R2 dari P4, dan dialokasikan ke P3

Adanya siklus merupakan syarat perlu namun bukan merupakan syarat cukup terjadinya deadlock

Metode penanganan deadlock

- Menggunakan satu protokol yang meyakinkan bahwa sistem tidak akan pernah mengalami deadlock → deadlock prevention atau avoidance
- Mengijinkan sistem mengalami deadlock, namun kemudian harus segera dapat memperbaikinya → deadlock detection and recovery
- Mengabaikan semua permasalahan bersama-sama, dan menganggap bahwa deadlock tidak akan pernah terjadi, digunakan dalam berbagai SOK, termasuk Unix dan Windows → deadlock ingoring and recovery

Deadlock prevention (1)

Kondisi untuk mengatasi deadlock dengan cara meyakinkan bahwa paling sedikit satu dari kondisi deadlock tidak terjadi

1. Mutual exclusion (buat *resource shareable*)
   - Non-shareable → mutex diperlukan, co: printer
   - Shareable → tidak perlu mutex, co: read-only file
2. Hold and Wait (melepas *resource* pada saat *request*)
   - Proses harus melepas resource yang dibawanya sebelum meminta resource lainnya
   - Low resource utilization: banyak resource dialokasikan namun tidak digunakan dalam waktu yang lama
   - Mungkin terjadi starvation: permintaan tidak dilayani untuk resource yang popular karena selalu dialokasikan untuk proses lain

Deadlock prevention (2)

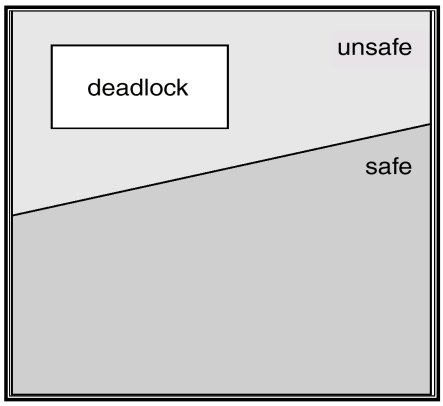
3. No preemption (melepas *resource* pada saat *waiting*)
   - Pembebasan semua resource yang dipegang suatu proses apabila proses ingin mengakses suatu resource lain, dan tidak dapat langsung dipenuhi
   - Resource dengan preemption ditambahkan pada proses yang ingin mengakses resource lain tersebut
   - Proses dimulai kembali apabila sudah mendapatkan kembali semua resource yang dilepaskan termasuk resource yang ingin diakses
4. Circular wait (*request* berurutan)
   - Memberi nomor pada setiap resource yang ada
   - Setiap proses boleh mengakses resource secara berurutan dari nomor rendah ke tinggi
   - Contoh: resource R1, … , R5. Jika P0 sedang mengakses R2, maka P0 hanya boleh request R3, … , R5. Jika P1 sedang mengakses R3, maka P1 hanya boleh request R4 atau R5

Deadlock Avoidance

- Memberikan informasi tambahan yang berhubungan dengan resource-resource yang akan diminta
- Meyakinkan bahwa tidak akan terjadi circular wait
- Status resource informasi:
  - # resource yang tersedia
  - # alokasi resource
  - # maksimum resource yang dibutuhkan proses

Deadlock avoidance – Safe state

- Status safe → sistem dapat mengalokasikan resource untuk tiap proses (sampai # maks.) dalam urutan yang tepat tanpa terjadinya deadlock
- Safe → tidak ada deadlock
- Unsafe → mungkin terjadi deadlock
- Avoidance → meyakinkan bahwa sistem tidak pernah memasuki keadaan status unsafe



Contoh: safe state

- 12 tape drive dan 3 proses P0, P1 dan P3
- Pada suatu saat *ti*

Proses	***Max Need***	***Current Need***
P0	10	5
P1	4	2
P2	9	2

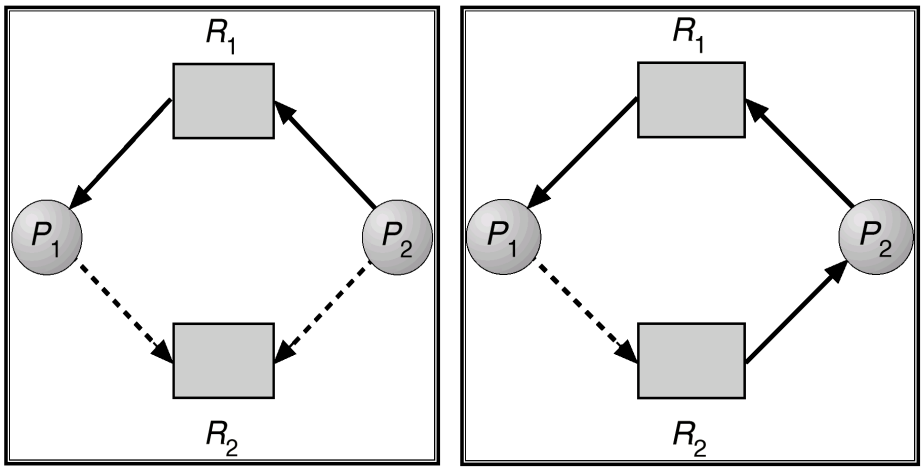
sisa 3 tape drive (*available*)

- Urutan status safe pada saat *ti* tersebut: *< P1, P0, P2 >*
- *Safe↔*
  # max need for all Pj's ≤ currently available resources + current need by all Pj's

Algoritma RAG

- Request edge: $Pi \longrightarrow Rj$
- Assignment edge: $Rj \longrightarrow Pi$
- Claim edge: $Pi \dashrightarrow Rj$, proses Pi boleh meminta resource Rj suatu saat di masa depan
- Pada saat dibutuhkan claim edge, dikonversikan menjadi request edge
- Jika suatu resource dilepas, maka assigment edge dikonversikan menjadi claim edge
- Claim $Pi \dashrightarrow Rj$ boleh ditambahkan pada graph jika semua edge yang berhubungan dengan *Pi* berupa claim edge

Contoh Algoritma RAG



Deadlock avoidance

Kondisi unsafe (siklus)

Algoritma Banker (1)

Misalkan ada ***n*** proses dalam sistem dan ***m*** tipe resources, terdapat data struktur sebagai berikut:

- Available (# resource yang tersedia pada suatu saat) → suatu vector dengan panjang *m*
- Max → matriks *n x m* yang mendefinisikan maksimum permintaan (*request*) untuk tiap-tiap proses
- Allocation → matriks *n x m* yang mendefinisikan jumlah resource untuk tiap-tiap tipe yang sedang dialokasikan untuk tiap proses
- Need → matriks *n x m* yang menunjukkan sisa resource yang dibutuhkan untuk tiap proses

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j]$$

Algoritma Safety (Banker)

1. Work dan Finish → vektor dengan panjang masing-masing **m** (#tipe resource) dan **n** (#proses), inisialisasi:
   - Work := Available
   - Finish [i] := false, untuk i = 1,2,..,n
2. Cari i sedemikian rupa, sehingga:
   - Finish[i] := false
   - Need(i) ≤ Work
3. Work : = Work + Allocation(i)
   - Finish[i] := true
   - Go to step 2
4. Jika Finish[i] := true untuk semua i, maka sistem dalam keadaan safe

Algoritma Resource-Request

- Misalkan Request(i) adalah vektor request untuk proses Pi. Jika Request $_i$ [j] = k, maka proses Pi ingin agar k ada segera untuk tipe Rj.
- Jika resource diminta oleh Pj, maka akan terjadi
  1. Jika Request(i) ≤ Need(i), goto step 2. Jika tidak maka terjadi kesalahan, proses berada di luar klaim maksimum
  2. Jika Request(i) ≤ Available, goto step 3. Jika tidak Pi harus menunggu karena resource tidak tersedia
  3. Modifikasi data struktur:
     - Available = Available – Request(i)
     - Allocation(i) = Allocation(i) + Request(i)
     - Need(i) = Need(i) – Request(i)

Contoh (1)

- 5 proses P_0 .. P_4; 3 resource types
- *A* (10 instances), *B* (5 instances), dan *C* (7 instances).
- Keadaan pada T_0:

	Allocation	*Max*	*Need*	*Available*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 3	3 3 2 (*free*)
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

Contoh (2)

- Penambahan Request_1 = (1,0,2) pada *T1*
- Isi dari matriks Need (lihat contoh sebelumnya)

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Langkah-langkah:

Algoritma Resource-Request:

Request_1 ≤ Need_1; (1,0,2) ≤ (1,2,2)

Request_1 ≤ Available; (1,0,2) ≤ (3,3,2)

Update data struktur:

Available = (old)Available – Request_1 = (2,3,0)

Allocation_1 = (old)Allocation_1 + Request_1
= (3,0,2)

Need_1 = (old)Need_1 – Request_1 = (0,2,0)

Aplikasikan algoritma Banker

Contoh (3)

- Keadaan baru dengan adanya Request_1:

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Dengan algoritma safety menunjukkan bahwa urutan $<P_1, P_3, P_4, P_0, P_2>$ memenuhi kondisi safe.

- Bagaimana dengan request (3, 3, 0) untuk P4 pada *T2* ?
- Bagaimana dengan request (0, 2, 0) untuk P0 pada *T2* ?

Contoh (4)

- Request_4 = (3,3,0)
  - Request_4 ≤ Need_4 ??? (3,3,0) ≤ (4,3,1)
  - Request_4 ≤ Available ??? (3,3,0) ≤ (2,3,0)

    Tidak, maka sistem tidak dapat memenuhi permintaan (resource tidak tersedia)
- Request_0 = (0,2,0)
  - Request_0 ≤ Need_0 ??? (0,2,0) ≤ (7,4,0)
  - Request_0 ≤ Available ??? (0,2,0) ≤ (2,3,0)
  - Update situasi (data struktur)
  - Aplikasikan algoritma safety

Contoh (5)

- Request_0 = (0,2,0)
- Available = (2,3,0) – (0,2,0) = (2,1,0)

	Allocation	*Need*	*Max*	*Available*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 3 0	7 2 3	7 5 3	2 1 0
P_1	3 0 2	0 2 0	3 2 2	
P_2	3 0 1	6 0 0	9 0 2	
P_3	2 1 1	0 1 1	2 2 2	
P_4	0 0 2	4 3 1	4 3 3	

- Algoritma safety <P1, P3, …, … ????> → sistem *unsafe*

  Setelah P1, available = (5,1,2)
  Setelah P3, available = (7,2,3)

Kelemahan Algoritma Banker

- Tidak semua proses mengetahui max resource
- Jumlah proses tidak tetap
- Beberapa resource terkadang bisa diambil dari sistem sewaktu-waktu, sehingga meskipun kelihatannya ada, namun kenyataannya tidak tersedia
- Menghendaki memberikan semua permintaan hingga waktu yang terbatas
- Proses seharusnya berjalan terpisah, sehingga urutan eksekusi tidak dibatasi oleh kebutuhan sinkronisasi proses
- Menghendaki client-server mengembalikan resource setelah batas tertentu

Deadlock Detection

- Algoritma deteksi: deadlock terjadi jika suatu permintaan tidak dapat ditangani segera
  - Single instance: jika resource allocation graph bersiklus
  - Multiple instance: Request_i ≥ Available
- Recovery:
  - Menggagalkan semua proses yang deadlock
  - Mem-*backup* semua proses yang deadlock dan me-restart semua proses tersebut
  - Menggagalkan semua proses yang deadlock secara berturut-turut hingga tidak ada deadlock
  - Menggagalkan pengalokasian resource secara berturut-turut hingga tidak ada deadlock

Kriteria penyingkiran proses

- Memiliki waktu proses (yang telah berjalan) kecil
- Jumlah hasil keluaran sedikit
- Mempunyai estimasi sisa waktu eksekusi besar
- Jumlah total sumberdaya terkecil yang telah dialokasikan
- Memiliki prioritas terkecil

Deadlock Recovery

- Ketika deadlock terdeteksi, maka ada beberapa alternatif pemecahan (recovery) oleh sebuah sistem komputer
  - Process termination
  - Resource preemption

Process termination

- Hentikan semua proses yang menyebabkan deadlock
- Hentikan proses yang bermasalah satu per satu, dengan bantuan algoritma deadlock detection
- Urutan penghentian proses (minimum cost):
  - Prioritas proses
  - Berapa lama proses sudah berlangsung, dan masih berapa lama lagi
  - Penggunaan resource oleh proses
  - Resource yang dibutuhkan dalam pelaksanaan proses
  - Berapa banyak proses yang harus dihentikan
  - Apakah proses interaktif atau batch

Resource preemption

- Memilih resource dan proses yang akan di-preemptive-kan
- Rollback → kembali ke suatu status safe, restart proses dari status tersebut
- Starvation → bagaimana meyakinkan bahwa tidak hanya beberapa resource atau proses yang akan di-preemptive-kan

Kombinasi penanganan deadlock

- Kombinasi tiga pemecahan dasar
  - Prevention (memutuskan salah satu syarat deadlock)
  - Avoidance (informasi tambahan untuk safety algortihm)
  - Detection (graph alokasi)

    Untuk setiap resource dalam sistem, pilih pemecahan mana yang optimal (???)
- Membagi resource ke dalam tingkatan
- Menggunakan teknik yang cocok untuk menangani deadlock dalam setiap tingkatan resource tersebut

Latihan soal (1)

1. Sebutkan faktor-faktor yang dapat menimbulkan deadlock !
2. Dengan menghindari adanya mutual exclusion apakah menjamin bahwa deadlock tidak akan terjadi? Jelaskan !
3. Kapan algoritma Resource Allocation Graph dapat digunakan untuk menunjukkan adanya deadlock?
4. Apakah dengan algoritma Banker dapat menjamin bahwa deadlock dapat dihindari? Jelaskan !

Latihan soal (2)

5. Suatu sistem memiliki 2 resource, yaitu: A (instances = 6 buah) dan B (instances = 11 buah). Ada 4 proses dalam sistem (P0, …, P3) dengan pengalokasian dan maks resource yang diperlukan, adalah:

	Alloc		Maks	
Proses	A	B	A	B
P0	2	1	5	3
P1	1	1	6	4
P2	0	3	4	4
P3	1	2	3	6

- Buatlah matriks Need !
- Tentukan vektor Available !
- Apakah sistem dalam status safe atau unsafe? Tunjukkan !
  Gunakan algoritma safety Banker !
- Jika ada tambahan Request_2 (1,0). Buatlah matriks Allocation dan Need serta vector Available yang baru. Apakah sistem dalam keadaan safe ? Berikan alasannya !