PERTEMUAN 8:

DEADLOCK

A. TUJUAN PEMBELAJARAN

Pada bab ini akan dijelaskan mengenai deadlock, Anda harus mampu:

1.1 Pengertian Deadlock

1.2 Penyebab deadlock

1.3 Cara mengatasi deadlock

B. URAIAN MATERI

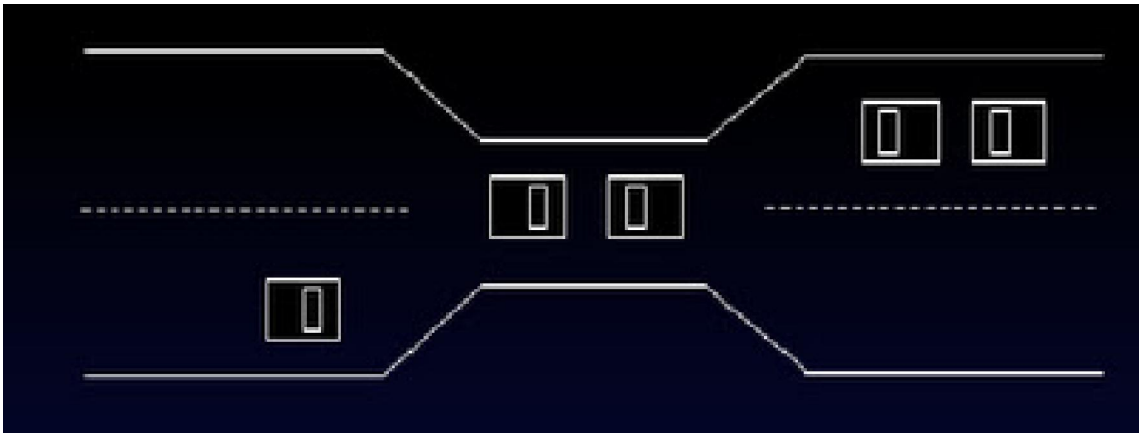
Tujuan Pembelajaran 1.1:
Deadlock

Pengertian Deadlock

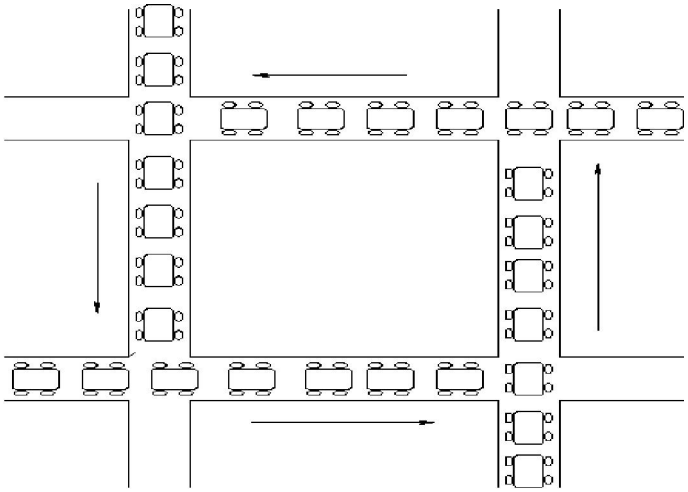
Deadlock dalam arti sebenarnya adalah kebuntuan. Kebuntuan yang dimaksud dalam sistem operasi adalah kebuntuan proses. Jadi Deadlock ialah suatu kondisi dimana proses tidak berjalan lagi atau tidak ada komunikasi lagi antar proses. Deadlock disebabkan karena proses yang satu menunggu sumber daya yang sedang dipegang oleh proses lain, proses lain itu pun sedang menunggu sumber daya yang dipegang olehnya. Dengan kata lain setiap proses dalam set menunggu untuk sumber yang hanya dapat dikerjakan oleh proses lain dalam set sedang menunggu.

Contoh Proses Deadlock

Contoh sederhananya ialah pada gambar berikut ini.



Contoh lain di persimpangan jalan.



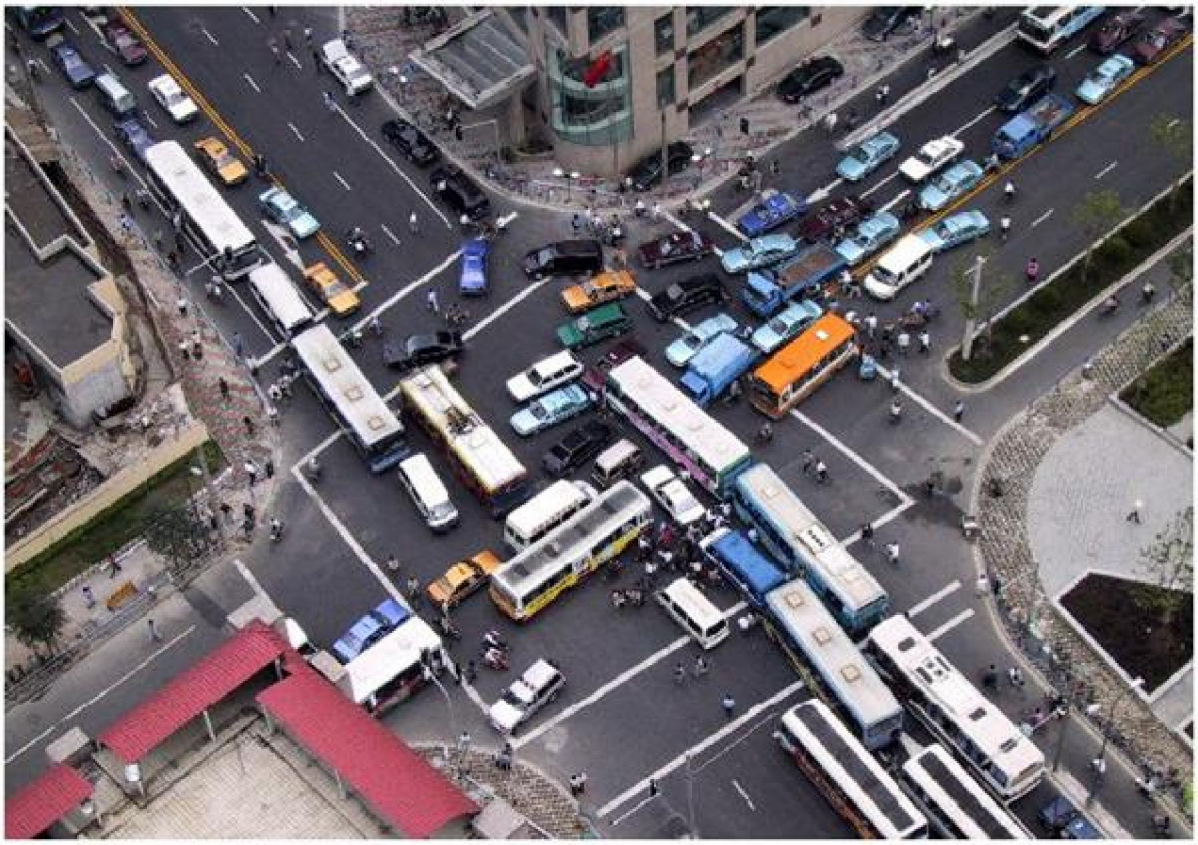
Dalam kasus ini setiap mobil bergerak sesuai nomor yang ditentukan, tetapi tanpa pengaturan yang benar, maka setiap mobil akan bertemu pada satu titik yang permanen (yang dilingkari) atau dapat dikatakan bahwa setiap mobil tidak dapat melanjutkan perjalanan lagi atau dengan kata lain terjadi Deadlock.

Kejadian Deadlock selalu tidak lepas dari sumber daya, bahwa hampir seluruhnya merupakan masalah sumber daya yang digunakan bersama-sama. Oleh karena itu, kita juga perlu tahu tentang jenis sumber daya, yaitu: sumber daya dapat digunakan lagi berulang-ulang dan sumber daya yang dapat digunakan dan habis dipakai atau dapat dikatakan sumber daya sekali pakai.

Sumber daya ini tidak habis dipakai oleh proses mana pun.Tetapi setelah proses berakhir, sumber daya ini dikembalikan untuk dipakai oleh proses lain yang sebelumnya tidak kebagian sumber daya ini. Contohnya prosesor, Channel I/O, disk, semaphore. Contoh peran sumber daya jenis ini pada terjadinya Deadlock ialah misalnya sebuah proses memakai disk A dan B, maka akan terjadi Deadlock jika setiap proses sudah memiliki salah satu disk dan meminta disk yang lain. Masalah ini tidak hanya dirasakan oleh pemrogram tetapi oleh seorang yang merancang sebuah

sistem operasi. Cara yang digunakan pada umumnya dengan cara memperhitungkan dahulu sumber daya yang digunakan oleh proses-proses yang akan menggunakan sumber daya tersebut. Contoh lain yang menyebabkan Deadlock dari sumber yang dapat dipakai berulang-ulang ialah berkaitan dengan jumlah proses yang memakai memori utama.

Contoh proses Deadlock di kehidupan nyata.



Model Deadlock

Pada sistem terdapat beberapa sumber daya (resource) yang digunakan untuk proses-proses untuk menyelesaikan task. Sumber daya yang pada sistem terdiri dari tipe resource CPU cycle, ruang memori, perangkat I/O yang disebut dengan tipe sumber daya R1, R2, . . ., Rm. Setiap tipe sumber daya Ri mempunyai beberapa anggota Wi.

Setiap proses yang menggunakan sumber daya menjalankan urutan operasi sebagai berikut :

- meminta (request) : meminta sumber daya

- memakai (use) : memakai sumber daya
- melepaskan (release) : melepaskan sumber daya

1. Syarat Terjadinya Deadlock.

   Suatu proses bisa dikatakan Deadlock jika proses tersebut memenuhi empat syarat. Syarat-syarat tersebut adalah:

   a. Mutual Exclusion.

      Kondisi yang pertama adalah mutual exclusion yaitu Proses memiliki hak milik pribadi terhadap sumber daya yang sedang digunakannya. Jadi, hanya ada satu proses yang menggunakan suatu sumber daya. Proses lain yang juga ingin menggunakannya harus menunggu hingga sumber daya tersebut dilepaskan oleh proses yang telah selesai menggunakannya. Suatu proses hanya dapat menggunakan secara langsung sumber daya yang tersedia secara bebas.

   b. Hold and Wait.

      Kondisi yang kedua adalah hold and wait yaitu Beberapa proses saling menunggu sambil menahan sumber daya yang dimilikinya. Suatu proses yang memiliki minimal satu buah sumber daya melakukan request lagi terhadap sumber daya. Akan tetapi, sumber daya yang dimintanya sedang dimiliki oleh proses yang lain. Pada saat yang sama, kemungkinan adanya proses lain yang juga mengalami hal serupa dengan proses pertama cukup besar terjadi. Akibatnya, proses-proses tersebut hanya bisa saling menunggu sampai sumber daya yang dimintanya dilepaskan. Sambil menunggu, sumber daya yang telah dimilikinya pun tidak akan dilepas. Semua proses itu pada akhirnya saling menunggu dan menahan sumber daya miliknya.

   c. No Preemption.

      Kondisi yang selanjutnya adalah no preemption yaitu Sebuah sumber daya hanya dapat dilepaskan oleh proses yang memilikinya secara sukarela setelah ia selesai menggunakannya. Proses yang menginginkan sumber daya tersebut harus menunggu sampai sumber daya tersedia, tanpa bisa merebutnya dari proses yang memilikinya.

   d. Circular Wait.

Kondisi yang terakhir adalah circular wait yaitu Kondisi membentuk siklus yang berisi proses-proses yang saling membutuhkan. Proses pertama membutuhkan sumber daya yang dimiliki proses kedua, proses kedua membutuhkan sumber daya milik proses ketiga, dan seterusnya sampai proses ke n-1 yang membutuhkan sumber daya milik proses ke n. Terakhir, proses ke n membutuhkan sumber daya milik proses yang pertama. Yang terjadi adalah proses-proses tersebut akan selamanya menunggu.

Ketiga syarat pertama merupakan syarat perlu bagi terjadinya deadlock. Keberadaan deadlock selalu berarti terpenuhi kondisi-kondisi diatas, tidak mungkin terjadi deadlock bila tidak ada ketiga kondisi itu. Deadlock terjadi berarti terdapat ketiga kondisi itu, tetapi adanya ketiga kondisi itu belum berarti terjadi deadlock.

Deadlock baru benar-benar terjadi bila syarat keempat terpenuhi. Kondisi keempat merupakan keharusan bagi terjadinya peristiwa deadlock. Bila salah satu dari kondisi tidak terpenuhi maka deadlock tidak terjadi.

2. Penanganan Deadlock

   Apabila suatu sistem atau proses mengalami Deadlock, maka semua itu bisa diatasi. Ada beberapa cara untuk menangani proses terjadinya Deadlock tersebut. Yaitu:

   a. Pengabaian Deadlock.

      Maksud dari pengabaian di sini adalah sistem mengabaikan terjadinya deadlock dan pura-pura tidak tahu kalau deadlock terjadi. Dalam penanganan dengan cara ini dikenal istilah ostrich algorithm. Pelaksanaan algoritma ini adalah sistem tidak mendeteksi adanya deadlock dan secara otomatis mematikan proses atau program yang mengalami deadlock. Kebanyakan sistem operasi yang ada mengadaptasi cara ini untuk menangani keadaan deadlock. Cara penanganan dengan mengabaikan deadlock banyak dipilih karena kasus deadlock tersebut jarang terjadi dan relatif rumit dan kompleks untuk diselesaikan. Sehingga biasanya hanya diabaikan oleh sistem untuk kemudian diselesaikan masalahnya oleh user dengan cara melakukan terminasi dengan Ctrl+Alt+Del atau melakukan restart terhadap komputer.

b. Pencegahan Deadlock.

Pencegahan deadlock dapat dilakukan dengan cara mencegah salah satu dari empat karakteristik atau syarat terjadinya deadlock. Berikut ini akan dibahas satu per satu cara pencegahan terhadap empat karakteristik tersebut.

- Mutual Exclusion.

  Kondisi mutual exclusion pada sumber daya adalah sesuatu yang wajar terjadi, yaitu pada sumber daya yang tidak dapat dibagi (non-sharable). Sedangkan pada sumber daya yang bisa dibagi tidak ada istilah mutual exclusive. Jadi, pencegahan kondisi yang pertama ini sulit karena memang sifat dasar dari sumber daya yang tidak dapat dibagi.

- Hold and Wait.

  Untuk kondisi yang kedua, sistem perlu memastikan bahwa setiap kali proses meminta sumber daya, ia tidak sedang memiliki sumber daya lain. Atau bisa dengan proses meminta dan mendapatkan sumber daya yang dimilikinya sebelum melakukan eksekusi, sehingga tidak perlu menunggu.

- No Preemption.

  Pencegahan kondisi ini dengan cara membolehkan terjadinya preemption. Maksudnya bila ada proses yang sedang memiliki sumber daya dan ingin mendapatkan sumber daya tambahan, namun tidak bisa langsung dialokasikan, maka akan preempted. Sumber daya yang dimiliki proses tadi akan diberikan pada proses lain yang membutuhkan dan sedang menunggu. Proses akan mengulang kembali eksekusinya setelah mendapatkan semua sumber daya yang dibutuhkannya, termasuk sumber daya yang dimintanya terakhir.

- Circular Wait.

  Kondisi 'lingkaran setan' ini dapat 'diputus' dengan jalan menentukan total kebutuhan terhadap semua tipe sumber daya yang ada. Selain itu, digunakan pula mekanisme enumerasi terhadap tipe-tipe sumber daya yang ada. Setiap proses yang akan meminta sumber daya harus meminta sumber daya dengan urutan yang menaik. Misalkan sumber daya printer memiliki nomor 1 sedangkan CD-ROM memiliki nomor 3. Proses boleh melakukan permintaan terhadap printer dan kemudian CD-ROM, namun tidak boleh sebaliknya.

c. Penghindaran Deadlock

Penghindaran terhadap deadlock adalah cara penanganan yang selanjutnya. Inti dari penghindaran adalah jangan sembarangan membolehkan proses untuk memulai atau meminta lagi. Maksudnya jangan pernah memulai suatu proses apabila nantinya akan menuju ke keadaan deadlock. Kedua, jangan memberikan kesempatan pada proses untuk meminta sumber daya tambahan jika penambahan tersebut akan membawa sistem pada keadaan deadlock. Tidak mungkin akan terjadi deadlock apabila sebelum terjadi sudah kita hindari.

Langkah lain untuk menghindari adalah dengan cara tiap proses memberitahu jumlah kebutuhan maksimum untuk setiap tipe sumber daya yang ada. Selanjutnya terdapat deadlock-avoidance algorithm yang secara rutin memeriksa state dari sistem untuk memastikan tidak adanya kondisi circular wait serta sistem berada pada kondisi safe state. Safe state adalah suatu kondisi dimana semua proses mendapatkan sumber daya yang dimintanya dengan sumber daya yang tersedia. Apabila tidak bisa langsung, ia harus menunggu selama waktu tertentu, kemudian mendapatkan sumber daya yang diinginkan, melakukan eksekusi, dan terakhir melepas kembali sumber daya tersebut. Terdapat dua jenis algoritma penghindaran yaitu resource-allocation graph untuk single instances resources serta banker's algorithm untuk multiple instances resources.

Dalam banker's algorithm, terdapat beberapa struktur data yang digunakan, yaitu:

Available . Jumlah sumber daya yang tersedia.

Max . Jumlah sumber daya maksimum yang diminta oleh tiap proses.
Allocation . Jumlah sumber daya yang sedang dimiliki oleh tiap proses.
Need . Sisa sumber daya yang masih dibutuhkan oleh proses, didapat dari max- allocation.

Kemudian terdapat safety algorithm untuk menentukan apakah sistem berada pada safe state atau tidak.

d. Pendeteksian Deadlock

Pada dasarnya kejadian deadlock sangatlah jarang terjadi. Apabila kondisi tersebut terjadi, masing-masing sistem operasi mempunyai mekanisme penanganan yang berbeda. Ada sistem operasi yang ketika terdapat kondisi deadlock dapat langsung mendeteksinya. Namun, ada pula sistem operasi yang bahkan tidak menyadari kalau dirinya sedang mengalami deadlock. Untuk sistem operasi yang dapat mendeteksi deadlock, digunakan algoritma pendeteksi. Secara lebih mendalam, pendeteksian kondisi deadlock adalah cara penanganan deadlock yang dilaksanakan apabila sistem telah berada pada kondisi deadlock. Sistem akan mendeteksi proses mana saja yang terlibat dalam kondisi deadlock. Setelah diketahui proses mana saja yang mengalami kondisi deadlock, maka diadakan mekanisme untuk memulihkan sistem dan menjadikan sistem berjalan kembali dengan normal.
Mekanisme pendeteksian adalah dengan menggunakan detection algorithm yang akan memberitahu sistem mengenai proses mana saja yang terkena deadlock. Setelah diketahui proses mana saja yang terlibat dalam deadlock, selanjutnya adalah dengan menjalankan mekanisme pemulihan sistem yang akan dibahas pada bagian selanjutnya. Berikut ini adalah algoritma pendeteksian deadlock.

e. Pemulihan Deadlock

Pemulihan kondisi sistem terkait dengan pendeteksian terhadap deadlock. Apabila menurut algoritma pendeteksian deadlock sistem berada pada keadaan deadlock, maka harus segera dilakukan mekanisme pemulihan sistem. Berbahaya apabila sistem tidak segera dipulihkan dari deadlock, karena sistem dapat mengalami penurunan performance dan akhirnya terhenti.
Cara-cara yang ditempuh untuk memulihkan sistem dari deadlock adalah sebagai berikut:

- Terminasi proses.

  Pemulihan sistem dapat dilakukan dengan cara melalukan terminasi terhadap semua proses yang terlibat dalam deadlock. Dapat pula dilakukan terminasi terhadap proses yang terlibat dalam deadlock secara satu per

satu sampai 'lingkaran setan' atau circular wait hilang. Seperti diketahui bahwa circular wait adalah salah satu karakteristik terjadinya deadlock dan merupakan kesatuan dengan tiga karakteristik yang lain. Untuk itu, dengan menghilangkan kondisi circular wait dapat memulihkan sistem dari deadlock.Dalam melakukan terminasi terhadap proses yang deadlock, terdapat beberapa faktor yang menentukan proses mana yang akan diterminasi. Faktor pertama adalah prioritas dari proses-proses yang terlibat deadlock. Faktor kedua adalah berapa lama waktu yang dibutuhkan untuk eksekusi dan waktu proses menunggu sumber daya. Faktor ketiga adalah berapa banyak sumber daya yang telah dihabiskan dan yang masih dibutuhkan. Terakhir, faktor utilitas dari proses pun menjadi pertimbangan sistem untuk melakukan terminasi pada suatu proses.

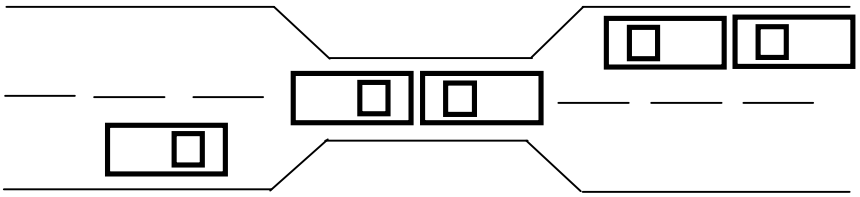
- Rollback and Restart.

  Dalam memulihkan keadaan sistem yang deadlock, dapat dilakukan dengan cara sistem melakukan preempt terhadap sebuah proses dan kembali ke state yang aman. Pada keadaan safe state tersebut, proses masih berjalan dengan normal, sehingga sistem dapat memulai proses dari posisi aman tersebut. Untuk menentukan pada saat apa proses akan rollback, tentunya ada faktor yang menentukan. Diusahakan untuk meminimalisasi kerugian yang timbul akibat memilih suatu proses menjadi korban. Harus pula dihindari keadaan dimana proses yang sama selalu menjadi korban, sehingga proses tersebut tidak akan pernah sukses menjalankan eksekusi.

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 akes 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).

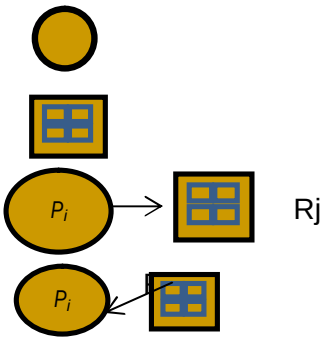
RESOURCE-ALLOCATION GRAPH

Sekumpulan vertex V dan sekumpulan edge E

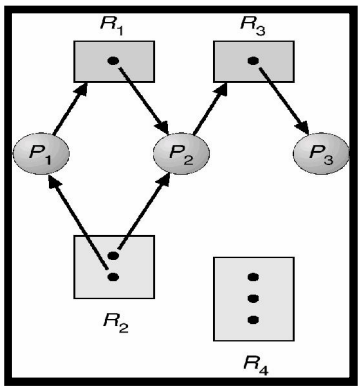
- V dipartisi ke dalam 2 tipe
  - $P = \{P_1, P_2, ..., P_n\}$, terdiri dari semua proses dalam sistem.
  - $R = \{R_1, R_2, ..., R_m\}$, terdiri dari semua sumberdaya dalam sistem
- request edge/permintaan edge : arah edge $P_1 \rightarrow R_j$
- assignment edge/penugasan edge – arah edge $R_j \rightarrow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

RESOURCE-ALLOCATION GRAPH (CONT.)

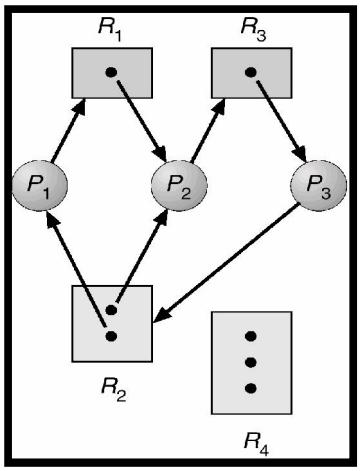
- Process
- Resource Type with 4 instances
- P_i requests instance of R_j
- P_i is holding an instance of R_j



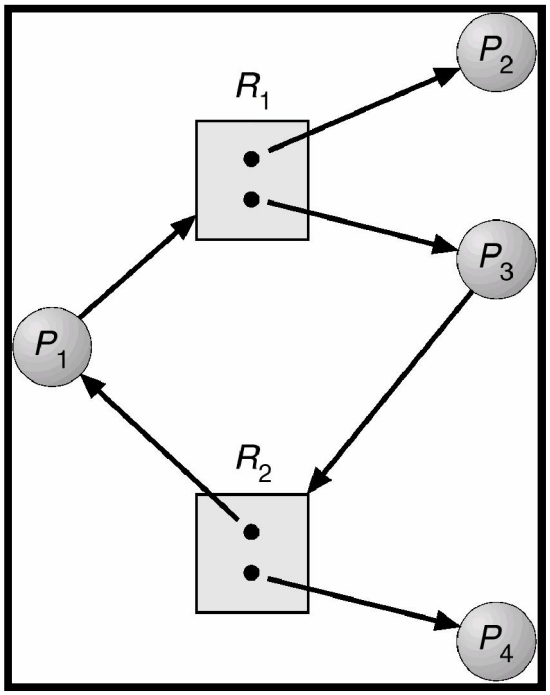
CONTOH RESOURCE ALLOCATION GRAPH



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)

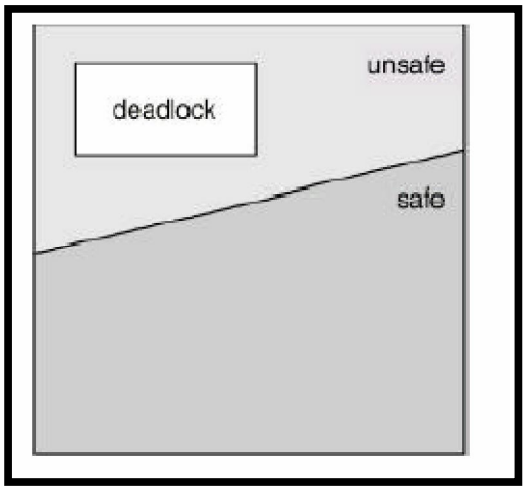
- Mencegah No-Preemption
  - Jika proses telah mengalokasi resource dan ingin mengalokasi resource lain – tapi tidak diperoleh (wait) : maka ia melepaskan semua resource yang telah dialokasi.
  - Proses akan di-restart kelak untuk mecoba kembali mengambil semua resources

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 menyebakan 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)

- Misakan 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 + Allocationi // 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 Æ 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

C. SOAL LATIHAN/TUGAS

1. Jelaskan pengertian dealock?
2. Jelaskan syarat terjadinya deadlock?
3. Jelaskan cara mengatasi deadlock?

D. DAFTAR PUSTAKA

Buku

Bambang Hariyanto. 1997. Sistem Operasi, Bandung:Informatika Bandung.

Dali S. Naga. 1992. Teori dan Soal Sistem Operasi Komputer,Jakarta: Gunadarma.

Operating System Concepts (6th or 7th Edition). Silberschatz, Galvin, Gagne, ISBN: 0-471-25060-0. Wiley

Silberschatz Galvin. 1995. 4 Edition Operating System Concepts: Addison Wesley.

Sri Kusumadewi. 2000. Sistem Operasi. Yogyakarta: J&J Learning.

Tanenbaum, A.1992. Modern Operating Systems.New York: Prentice Hall

6.

Link and Sites:

http://www.ilmukomputer.com
http://vlsm.bebas.org
http://www.wikipedia.com