Deadlock

I Made Andhika

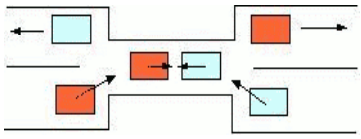
Gambaran Umum Deadlock

- Misalkan pada suatu komputer terdapat dua buah program, sebuah *tape drive* dan sebuah *printer*. Program A mengontrol *tape* drive, sementara program B mengontrol *printer*. Setelah beberapa saat, program A meminta *printer*, tapi *printer* masih digunakan. Berikutnya, B meminta *tape drive*, sedangkan A masih mengontrol *tape drive*. Dua program tersebut memegang kontrol terhadap sumber daya yang dibutuhkan oleh program yang lain. Tidak ada yang dapat melanjutkan proses masing-masing sampai program yang lain memberikan sumber dayanya, tetapi tidak ada yang mengalah. Kondisi inilah yang disebut *Deadlock* atau pada beberapa buku disebut *Deadly Embrace*

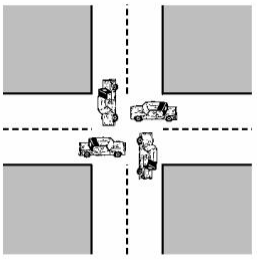
Definisi Umum Deadlock

- Suatu kondisi dimana proses tidak berjalan lagi atau pun tidak ada komunikasi lagi antar proses.
- *Deadlock* disebabkan karena proses yang satu menunggu sumber daya yang sedang dipegang oleh proses lain yang sedang menunggu sumber daya yang dipegang oleh proses tersebut.

Contoh Deadlock



Deadlock pada jembatan



Deadlock pada persimpangan

Model Sistem Deadlock

- Menurut *Coffman* dalam bukunya *"Operating System"* menyebutkan empat syarat bagi terjadinya *deadlock*, yaitu:
  - *Mutual Exclusion*
    - Suatu kondisi dimana setiap sumber daya diberikan tepat pada satu proses pada suatu waktu.
  - *Hold and Wait*
    - Kondisi yang menyatakan proses-proses yang sedang memakai suatu sumber daya dapat meminta sumber daya yang lain.
  - *Non-pre-emptive*
    - Kondisi dimana suatu sumber daya yang sedang berada pada suatu proses tidak dapat diambil secara paksa dari proses tersebut,sampai proses itu melepaskannya.
  - *Circular Wait*
    - Kondisi yang menyatakan bahwa adanya rantai saling meminta sumber daya yang dimiliki oleh suatu proses oleh proses lainnya.

Strategi Mengahadapi Deadlock

- Strategi untuk menghadapi *deadlock* dapat dibagi menjadi tiga pendekatan, yaitu:
  - Mengabaikan adanya *deadlock*.
  - Memastikan bahwa *deadlock* tidak akan pernah ada, baik dengan metode Pencegahan, dengan mencegah empat kondisi *deadlock* agar tidak akan pernah terjadi. Metode Menghindari *deadlock*, yaitu mengizinkan empat kondisi *deadlock*, tetapi menghentikan setiap proses yang kemungkinan mencapai *deadlock*.
  - Membiarkan *deadlock* untuk terjadi, pendekatan ini membutuhkan dua metode yang saling mendukung, yaitu:
    - Pendeteksian *deadlock*, untuk mengidentifikasi ketika *deadlock* terjadi.
    - Pemulihan *deadlock*, mengembalikan kembali sumber daya yang dibutuhkan pada proses yang memintanya.

Metode Mengahadapi Deadlock

- **Strategi *Ostrich***
  - Pendekatan yang paling sederhana adalah dengan menggunakan strategi burung unta: masukkan kepala dalam pasir dan seolah-olah tidak pernah ada masalah sama sekali. Beragam pendapat muncul berkaitan dengan strategi ini. Menurut para ahli Matematika, cara ini sama sekali tidak dapat diterima dan semua keadaan *deadlock* harus ditangani. Sementara menurut para ahli Teknik, jika komputer lebih sering mengalami kerusakkan disebabkan oleh kegagalan *hardware*, *error* pada kompilator atau *bugs* pada sistem operasi. Maka ongkos yang dibayar untuk melakukan penanganan *deadlock* sangatlah besar dan lebih baik mengabaikan keadaan *deadlock* tersebut. Metode ini diterapkan pada sistem operasi UNIX dan MINIX.

Metode Mengahadapi Deadlock

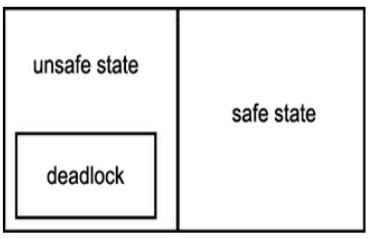
- **Menghindari *Deadlock***
  - Pendekatan metode ini adalah dengan hanya memberi kesempatan ke permintaan sumber daya yang tidak mungkin akan menyebabkan *deadlock*. Metode ini memeriksa dampak pemberian akses pada suatu proses, jika pemberian akses tidak mungkin menuju kepada *deadlock*, maka sumber daya akan diberikan pada proses yang meminta. Jika tidak aman, proses yang meminta akan di-*suspend* sampai suatu waktu permintaannya aman untuk diberikan. Kondisi ini terjadi ketika setelah sumber daya yang sebelumnya dipegang oleh proses lain telah dilepaskan.
  - Kondisi aman yang dimaksudkan selanjutnya disebut sebagai *safe-state*, sedangkan keadaan yang tidak memungkinkan untuk diberikan sumber daya yang diminta disebut *unsafe-state*.

Metode Mengahadapi Deadlock

- **Kondisi Aman (*Safe state*)**
  - Suatu keadaan dapat dinyatakan sebagai *safe state* jika tidak terjadi *deadlock* dan terdapat cara untuk memenuhi semua permintaan sumber daya yang ditunda tanpa menghasilkan *deadlock*. Dengan cara mengikuti urutan tertentu.

Metode Mengahadapi Deadlock

- **Kondisi Tak Aman (*Unsafe state*)**
  - Suatu *state* dinyatakan sebagai *state* tak selamat (*unsafe state*) jika tidak terdapat cara untuk memenuhi semua permintaaan yang saat ini ditunda dengan menjalankan proses-proses dengan suatu urutan.



Metode Mengahadapi Deadlock

- **Algoritma *Bankir***
  - Algoritma penjadualan ini diungkapkan oleh Dijkstra (1965) lebih dikenal dengan nama Algoritma Bankir. Model ini menggunakan suatu kota kecil sebagai percontohan dengan suatu bank sebagai sistem operasi, pinjaman sebagai sumber daya dan peminjam sebagai proses yang membutuhkan sumber daya. Deadlock akan terjadi apabila terdapat seorang peminjam yang belum mengembalikan uangnya dan ingin meminjam kembali, padahal uang yang belum dikembalikan tadi dibutuhkan oleh peminjam lain yang juga belum mengembalikan uang pinjamannya.

Algoritma Bankir (Lanjutan)

- Beberapa kelemahan algoritma Bankir Tanenbaum (1992), Stallings (1995) dan Deitel (1990) adalah sebagai berikut:
  - Sulit untuk mengetahui seluruh sumber daya yang dibutuhkan proses pada awal eksekusi.
  - Jumlah proses yang tidak tetap dan berubah-ubah.
  - Sumber daya yang tadinya tersedia dapat saja menjadi tidak tersedia kembali.
  - Proses-proses yang dieksekusi haruslah tidak dibatasi oleh kebutuhan sinkronisasi antar proses.
  - Algoritma ini menghendaki memberikan semua permintaan selama waktu yang berhingga.

Metode Mengahadapi Deadlock

- **Metode Pencegahan**
  - Metode ini merupakan metode yang paling sering digunakan. Metode Pencegahan dianggap sebagai solusi yang bersih dipandang dari sudut tercegahnya *deadlock*. Tetapi pencegahan akan mengakibatkan kinerja utilisasi sumber daya yang buruk.
  - Metode pencegahan menggunakan pendekatan dengan cara meniadakan empat syarat yang dapat menyebabkan deadlock terjadi pada saat eksekusi Coffman (1971).

Metode Pencegahan (Lanjutan)

- Syarat pertama yang akan dapat ditiadakan adalah *Mutual Exclusion*, jika tidak ada sumber daya yang secara khusus diperuntukkan bagi suatu proses maka tidak akan pernah terjadi *deadlock*. Namun jika membiarkan ada dua atau lebih proses mengakses sebuah sumber daya yang sama akan menyebabkan *chaos*. Langkah yang digunakan adalah dengan *spooling* sumber daya, yaitu dengan mengantrikan *job-job* pada antrian dan akan dilayani satu-satu.
- Beberapa masalah yang mungkin terjadi adalah:
  - Tidak semua dapat di-*spool*, tabel proses sendiri tidak mungkin untuk di-*spool*
  - Kompetisi pada ruang disk untuk *spooling* sendiri dapat mengarah pada *deadlock*
- Hal inilah yang menyebabkan mengapa syarat pertama tidak dapat ditiadakan, jadi *mutual exclusion* benar-benar tidak dapat dihilangkan.

Metode Pencegahan (Lanjutan)

- Cara kedua dengan meniadakan kondisi *hold and wait* terlihat lebih menjanjikan. Jika suatu proses yang sedang menggunakan sumber daya dapat dicegah agar tidak dapat menunggu sumber daya yang lain, maka *deadlock* dapat dicegah. Langkah yang digunakan adalah dengan membuat proses agar meminta sumber daya yang mereka butuhkan pada awal proses sehingga dapat dialokasikan sumber daya yang dibutuhkan. Namun jika terdapat sumber daya yang sedang terpakai maka proses tersebut tidak dapat memulai prosesnya.
- Masalah yang mungkin terjadi:
  - Sulitnya mengetahui berapa sumber daya yang dibutuhkan pada awal proses
  - Tidak optimalnya pengunaan sumber daya jika ada sumber daya yang digunakan hanya beberapa waktu dan tidak digunakan tapi tetap dimiliki oleh suatu proses yang telah memintanya dari awal.

Metode Pencegahan (Lanjutan)

- Meniadakan syarat ketiga *non preemptive* ternyata tidak lebih menjanjikan dari meniadakan syarat kedua, karena dengan meniadakan syarat ketiga maka suatu proses dapat dihentikan ditengah jalan. Hal ini tidak dimungkinkan karena hasil dari suatu proses yang dihentikan menjadi tidak baik.
- Cara terakhir adalah dengan meniadakan syarat keempat *circular wait*. Terdapat dua pendekatan, yaitu:
  - Mengatur agar setiap proses hanya dapat menggunakan sebuah sumber daya pada suatu waktu, jika menginginkan sumber daya lain maka sumber daya yang dimiliki harus dilepas.
  - Membuat penomoran pada proses-proses yang mengakses sumber daya. Suatu proses dimungkinkan untuk dapat meminta sumber daya kapan pun, tetapi permintaannya harus dibuat terurut.
- Masalah yang mungkin terjadi dengan mengatur bahwa setiap proses hanya dapat memiliki satu proses adalah bahwa tidak semua proses hanya membutuhkan satu sumber daya, untuk suatu proses yang kompleks dibutuhkan banyak sumber daya pada saat yang bersamaan. Sedangkan dengan penomoran masalah yang dihadapi adalah tidak terdapatnya suatu penomoran yang dapat memuaskan semua pihak.

Metode Pencegahan (Lanjutan)

- Secara ringkas pendekatan yang digunakan pada metode pencegahan deadlock dan masalah-masalah yang menghambatnya, terangkum dalam tabel dibawah ini.

Tabel 1 Deadlock

Syarat	Langkah	Kelemahan
Mutual Exclusion	*Spooling* sumber daya	Dapat menyebabkan *chaos*
Hold and Wait	Meminta sumber daya di awal	Sulit memperkirakan di awal dan tidak optimal
No Pre-emptive	Mengambil sumber daya di tengah proses	Hasil proses tidak akan baik
Circular Wait	Penomoran permintaan sumber daya	Tidak ada penomoran yang memuaskan semua pihak

Mendeteksi dan Memulihkan *Deadlock*

- Metode ini mengunakan pendekatan dengan teknik untuk menentukan apakah *deadlock* sedang terjadi serta proses-proses dan sumber daya yang terlibat dalam deadlock tersebut. Setelah kondisi *deadlock* dapat dideteksi, maka langkah pemulihan dari kondisi *deadlock* dapat segera dilakukan. Langkah pemulihan tersebut adalah dengan memperoleh sumber daya yang diperlukan oleh proses-proses yang membutuhkannya. Beberapa cara digunakan untuk mendapatkan sumber daya yang diperlukan, yaitu dengan terminasi proses dan *pre-emption* (mundur) suatu proses. Metode ini banyak digunakan pada komputer *mainframe* berukuran besar.

Mendeteksi dan Memulihkan *Deadlock*

- **Terminasi Proses**
  - Metode ini akan menghapus proses-proses yang terlibat pada kondisi *deadlock* dengan mengacu pada beberapa syarat. Beberapa syarat yang termasuk dalam metode ini adalah, sebagai berikut:
    - Menghapus semua proses yang terlibat dalam kondisi *deadlock* (solusi ini terlalu mahal).
    - Menghapus satu persatu proses yang terlibat, sampai kondisi *deadlock* dapat diatasi (memakan banyak waktu).
    - Menghapus proses berdasarkan prioritas, waktu eksekusi, waktu untuk selesai, dan kedalaman dari *rollback*.

Mendeteksi dan Memulihkan *Deadlock*

- ***Resources Preemption***
  - Metode ini lebih menekankan kepada bagaimana menghambat suatu proses dan sumber daya, agar tidak terjebak pada *unsafe condition*.
  - Beberapa langkahnya, yaitu:
    - Pilih salah satu - proses dan sumber daya yang akan di-*preempt*.
    - *Rollback* ke *safe state* yang sebelumnya telah terjadi.
    - Mencegah suatu proses agar tidak terjebak pada *starvation* karena metode ini.

Diagram Graf

- Sebuah sistem komputer terdiri dari berbagai macam sumber-daya (*resources*), seperti:

1. Fisik (Perangkat, Memori)
2. Logika (Lock, Database record)
3. Sistem Operasi (PCB Slots)
4. Aplikasi (Berkas)

- Mekanisme hubungan dari proses-proses dan sumber-daya yang dibutuhkan/digunakan dapat di diwakilkan dengan *graf.*

Diagram Graf

- Graf adalah suatu struktur diskrit yang terdiri dari vertex dan sisi, dimana sisi menghubungkan vertex-vertex yang ada.
- Graf dibagi menjadi dua bagian, yaitu
  simple graf dan multigraf.
- Salah satu contoh implementasi graf dalam sistem operasi adalah graf alokasi sumber daya
- Graf alokasi sumber daya merupakan graf sederhana dan graf berarah.
- Graf alokasi sumber daya adalah bentuk visualisasi dalam mendeteksi maupun menyelesaikan masalah *deadlock*.

Diagram Graf

- Komponen Graf Alokasi Sumber daya

  1. Proses P= {P0, P1, P2, P3,..., Pi,..., Pm}. Terdiri dari semua proses yang ada di sistem. Untuk proses, vertexnya digambarkan sebagai lingkaran dengan nama prosesnya.

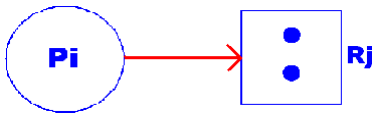
  2. Sumber daya R= {R0, R1, R2, R3,..., Rj,..., Rn}. Terdiri dari semua sumber daya yang ada disistem. Untuk sumber daya, vertexnya digambarkan sebagai segi empat dengan instans yang dapat dialokasikan serta nama sumber dayanya.

Diagram Graf

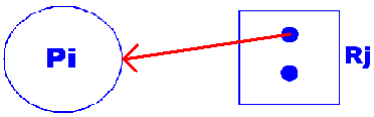
- Sisi, E={Pi-> Rj, , Rj-> Pi, } terdiri dari dua jenis, yaitu:

1. Sisi permintaan: Pi -> Rj Sisi permintaan menggambarkan adanya suatu proses Pi yang meminta sumber daya Rj.
2. Sisi alokasi: Rj -> Pi. Sisi alokasi menggambarkan adanya suatu sumber daya Rj yang mengalokasikan salah satu instansnya pada proses Pi.

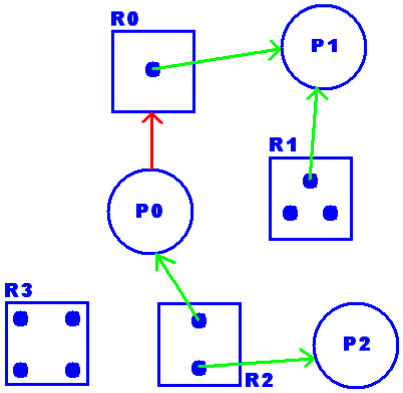
Diagram Graf



Proses Pi meminta sumber daya Rj



Sumber daya Rj yang mengalokasikan salah satu

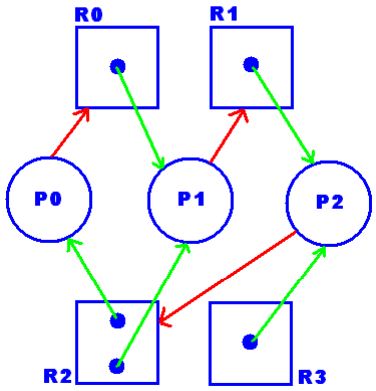


Graf Alokasi Sumber daya

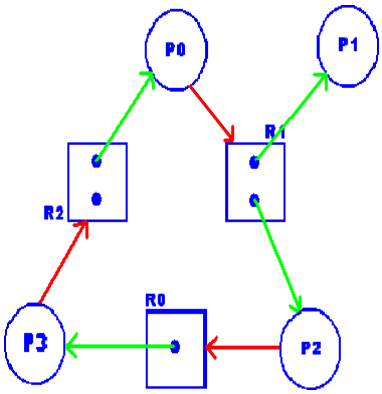
Diagram Graf

- Untuk mengetahui ada atau tidaknya deadlock (Pendeteksian) dalam suatu graf dapat dilihat dari perputaran dan resource yang dimilikinya, yaitu:
  - ❑Jika tidak ada perputaran berarti tidak deadlock.
  - ❑Jika ada perputaran, ada potensi terjadi deadlock.
  - ❑Resource dengan instan tunggal dan perputaran mengakibatkan deadlock.

Diagram Graf



Graf dengan deadlock

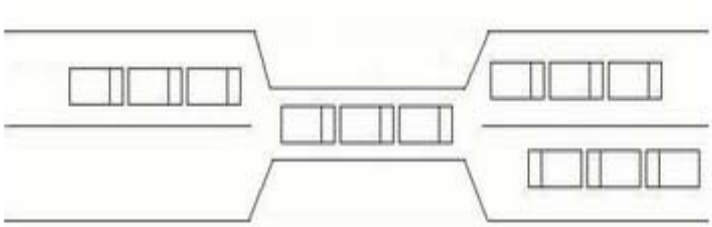


Graf tanpa deadlock

Starvation

- *Starvation* adalah kondisi yang biasanya terjadi setelah *deadlock*. Proses yang kekurangan *resource* (karena terjadi *deadlock*) tidak akan pernah mendapat *resource* yang dibutuhkan sehingga mengalami *starvation* (kelaparan).
- Namun, *starvation* juga bisa terjadi tanpa *deadlock*. Hal ini ketika terdapat kesalahan dalam sistem sehingga terjadi ketimpangan dalam pembagian *resouce*. Satu proses selalu mendapat *resource*, sedangkan proses yang lain tidak pernah mendapatkannya.
- Ilustrasi *starvation* tanpa *deadlock* di dunia nyata dapat dilihat di bawah ini.

Starvation



- Pada gambar diatas, pada antrian kanan terjadi *starvation* karena *resource* (jembatan) selalu dipakai oleh antrian kiri, dan antrian kanan tidak mendapatkan giliran.