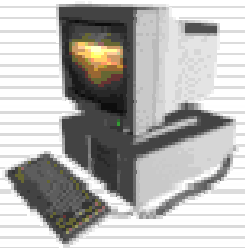
Deadlock

Deadly Embrace

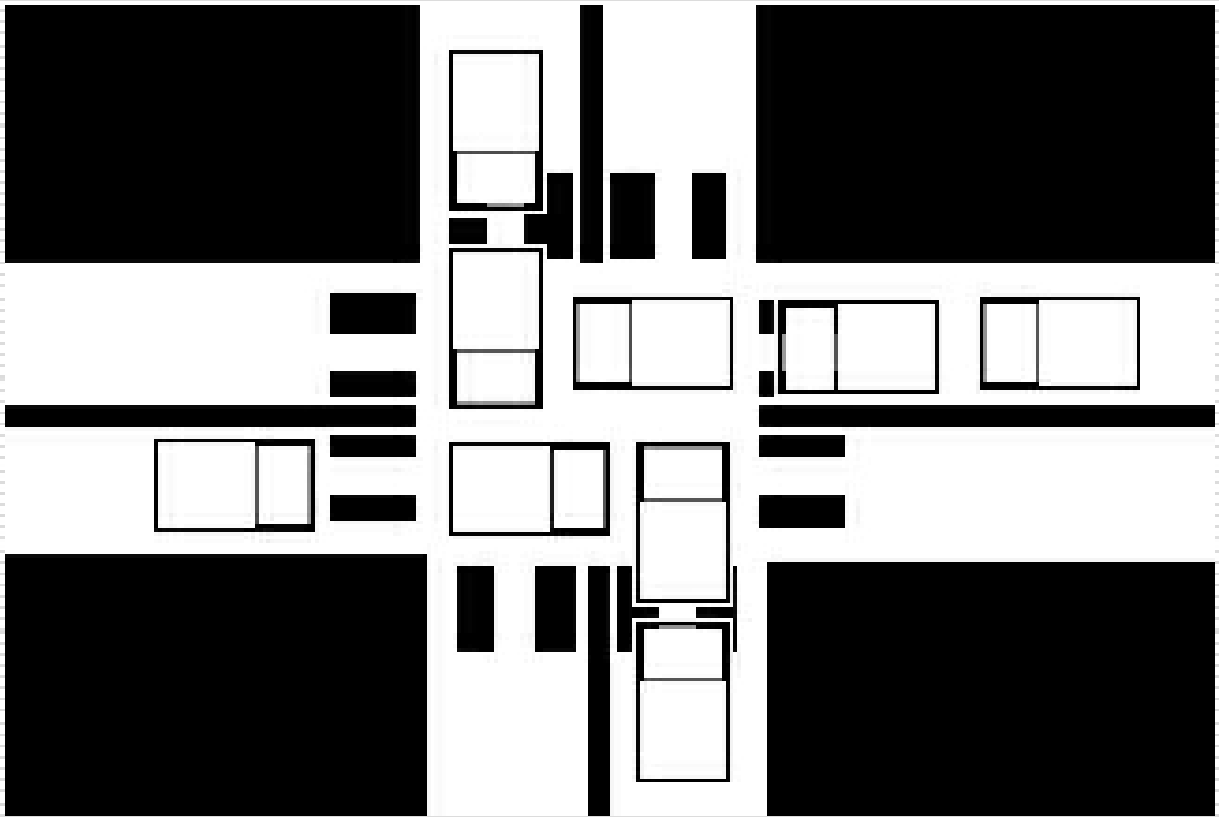


Deskripsi:

Dua atau lebih proses dikatakan berada dalam kondisi *deadlock*, bila setiap proses yang ada menunggu suatu kejadian yang hanya dapat dilakukan oleh proses lain dalam himpunan tersebut.

Contoh : (ilustrasi dipersimpangan jalan)

- Terdapat satu jalur pada jalan.
- 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).

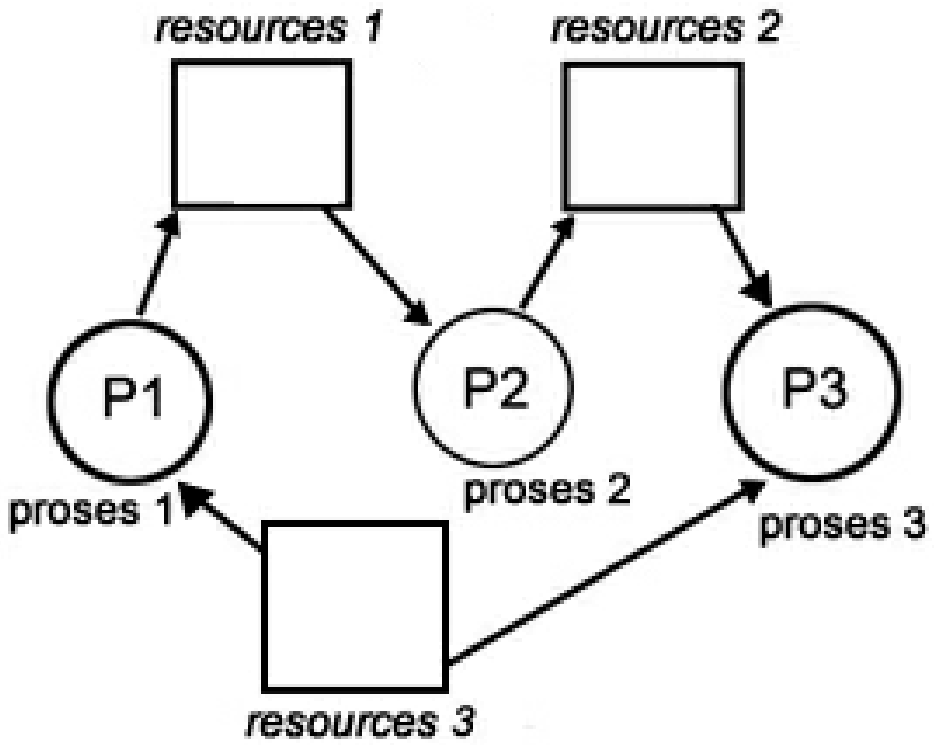


Ilustrasi terjadinya Deadlock

Resources-Allocation Graph :

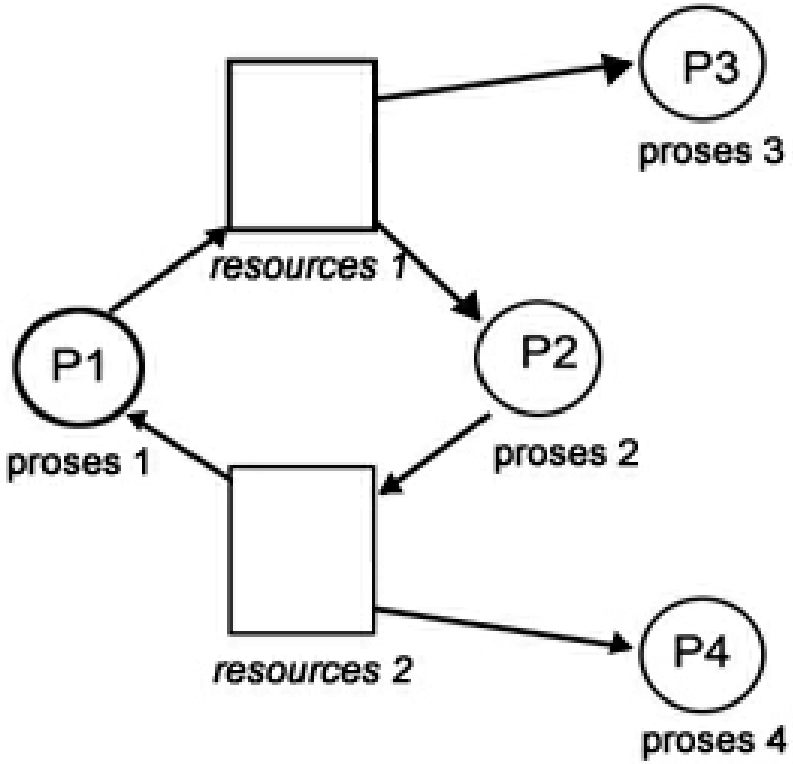
- Sebuah cara visual (matematika) untuk menentukan apakah ada *deadlock*, atau kemungkinan terjadinya.
- G = (V, E) Graf berisi *node and edge*. *Node* V terdiri dari proses-proses = {P1, P2, P3, ...} dan jenis *resource*. {R1, R2, ...} *Edge* E adalah (Pi, Rj) atau (Ri, Pj)
- Sebuah panah dari *process* ke *resource* menandakan proses meminta *resource*. Sebuah panah dari *resource* ke *process* menunjukkan sebuah *instance* dari *resource* telah ditempatkan ke proses.
- *Process* adalah lingkaran, *resource* adalah kotak; titik-titik merepresentasikan jumlah *instance* dari *resource* Dalam tipe. Meminta poin-poin ke kotak, perintah datang dari titik.

Gambar Graph. Sumber:

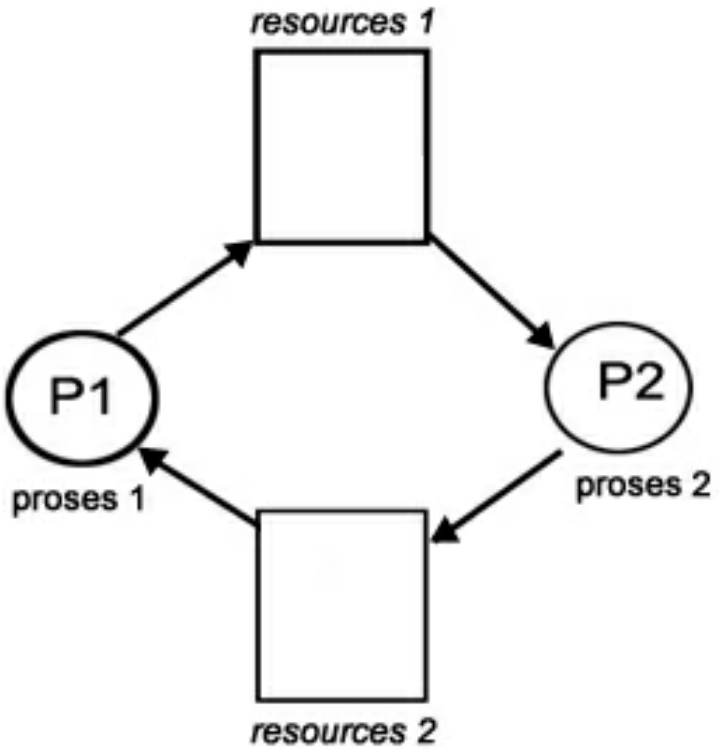


JIKA MEMBENTUK LINGKARAN, MAKA:

- jika tipe *resource* memiliki banyak *instance*, maka *deadlock* DAPAT ada.



- jika setiap tipe *resource* mempunyai satu *instance*, maka *deadlock* telah terjadi.



EMPAT PENYEBAB TERJADINYA *DEADLOCK*

(COFFMAN):

1. *Mutual Exclusion*

   Suatu kondisi dimana setiap sumber daya diberikan tepat pada satu proses pada suatu waktu.

2. *Hold and Wait*

   Kondisi yang menyatakan proses-proses yang sedang memakai suatu sumber daya dapat meminta sumber daya yang lain.

3. *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.

4. *Circular Wait*

   Kondisi yang menyatakan bahwa adanya rantai saling meminta sumber daya yang dimiliki oleh suatu proses oleh proses lainnya.

Metode untuk mengatasi *deadlock* , :

1. Strategi *Ostrich*
2. Mencegah *Deadlock*
3. Menghindari *Deadlock*
4. Mendeteksi *Deadlock* dan Memulihkan *Deadlock*

STRATEGI MENGHADAPI *DEADLOCK* , YAITU:

1. Mengabaikan adanya *deadlock (***Strategi *Ostrich)***.

2. Memastikan bahwa *deadlock* tidak akan pernah ada, baik dengan **metode Pencegahan**, dengan mencegah empat kondisi *deadlock* agar tidak akan pernah terjadi.

3. **Metode Menghindari *deadlock***, yaitu mengizinkan empat kondisi *deadlock*, tetapi menghentikan setiap proses yang kemungkinan mencapai *deadlock*.

4. Membiarkan *deadlock* untuk terjadi, pendekatan ini membutuhkan dua metode yang saling mendukung, yaitu:
   a. **Pendeteksian *deadlock*, untuk mengidentifikasi ketika *deadlock* terjadi.**
   b. **Pemulihan *deadlock*, mengembalikan kembali sumber daya yang dibutuhkan pada proses yang memintanya.**

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.
- Menurut para ahli Matematika, cara ini sama sekali tidak dapat diterima dan semua keadaan *deadlock* harus ditangani.
- 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.

MENCEGAH *DEADLOCK*

- Metode yang paling sering digunakan.
- 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).

Syarat pertama :

- Meniadakan *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.

syarat pertama tidak dapat ditiadakan, Karena :

- Tidak semua dapat di-*spool*, tabel proses sendiri tidak mungkin untuk di-*spool*
- Kompetisi pada ruang disk untuk *spooling* sendiri dapat mengarah pada *deadlock*

Syarat kedua :

- 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.

Syarat Ketiga :

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.

Syarat Keempat :

- meniadakan *circular wait*.

- Terdapat dua pendekatan, yaitu:
  1. 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.
  2. Membuat penomoran pada proses-proses yang mengakses sumber daya. Suatu proses dimungkinkan untuk dapat meminta sumber daya kapan pun, tetapi permintaannya harus dibuat terurut.

Deadlock Avoidence

Merupakan Pengujian Secara Dinamis Untuk Meyakinkan bahwa tidak ada kondisi *Circular Wait,* meliputi banyaknya *resource yang tersedia, banyaknya resource yang dialokasikan, terdapat maksimum 2 resource yang dibutuhkan oleh proses*

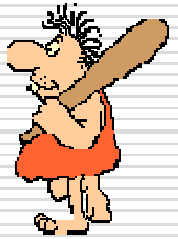
Masalah yang Dihadapi :

- 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.

Kesimpulan Mencegah 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



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*.

- ## 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.

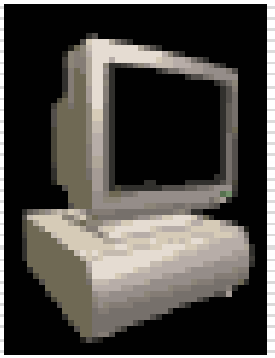
- ## Kondisi Tak Aman (*Unsafe state*)

  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.

Contoh :**Kondisi Aman (*Safe state*)**

- Pada suatu sistem dengan 10 sumber daya yg sama, dimana P1 memerlukan sd_max sebanyak 10 dan saat ini sedang menggenggam 2, P2 sd_max 3 dan menggenggam 1, P3 sd_max 7 dan menggenggam 3.

- Tentukan urutan eksekusi proses utk mencapai kondis aman ?



Kondisi awal Proses

Proses	Sumber Daya Yg di Genggam	Sumber Daya Maximum
P1	2	10
P2	1	3
P3	3	7
Sisa Sumber Daya	4	

Mendeteksi *Deadlock* 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.

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:
  1. Menghapus semua proses yang terlibat dalam kondisi *deadlock* (solusi ini terlalu mahal).
  2. Menghapus satu persatu proses yang terlibat, sampai kondisi *deadlock* dapat diatasi (memakan banyak waktu).
  3. Menghapus proses berdasarkan prioritas, waktu eksekusi, waktu untuk selesai, dan kedalaman dari *rollback*.

Resources Preemption

- Metode ini lebih menekankan kepada bagaimana menghambat suatu proses dan sumber daya, agar tidak terjebak pada *unsafe condition*.

- Beberapa langkahnya, yaitu:
  1. Pilih salah satu - proses dan sumber daya yang akan di-*preempt*.
  2. *Rollback* ke *safe state* yang sebelumnya telah terjadi.
  3. Mencegah suatu proses agar tidak terjebak pada *starvation* karena metode ini.

Cara mendeteksi Deadlock

- Menggagalkan / Membatalkan semua proses yang deadlock
- Mem-backup semua proses yang deadlock dan me-restract semua proses tersebut
- Menggagalkan semua proses yang deadlock secara berturut-turut hingga tidak ada deadlock
- Menggagalkan pengalokasian resource-resource berturut-turut hingga tidak ada deadlock