Deadlock (2)

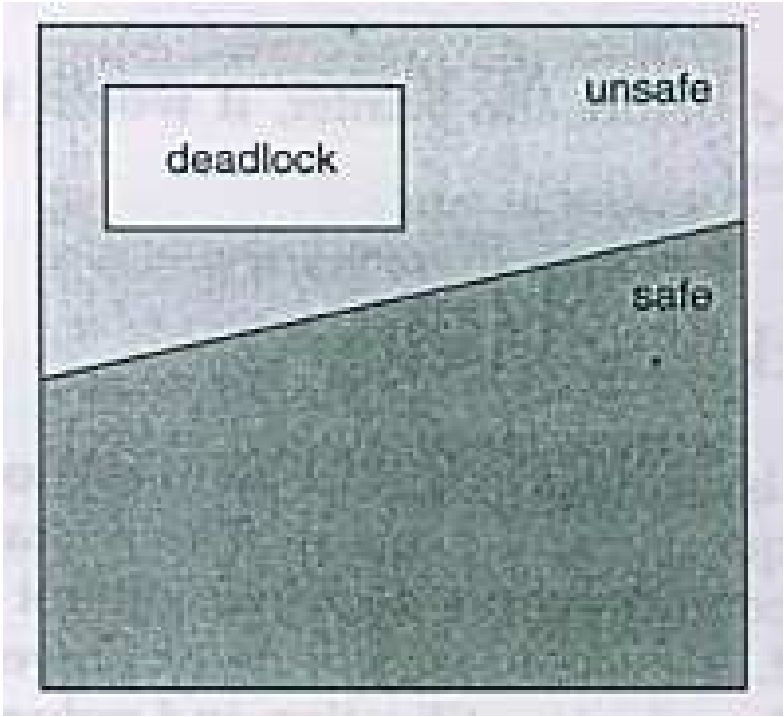
Menghindari Deadlock

- Algoritma untuk menghindari deadlock membutuhkan informasi tambahan mengenai aliran resource yang diminta. Model yang sederhana dan sangat berguna membutuhkan setiap proses menentukan resource maksimum dari setiap tipe yang diinginkan
- Menggunakan jumlah resource maksimum yang diminta, bersama-sama dengan jumlah resource saat ini yang dibawa oleh setiap proses, memungkinkan untuk membentuk algoritma yang menjamin sistem tidak pernah memasuki state deadlock.
- Algoritma deadlock-avoidance secara dinamis mengetes state resource-allocation untuk meyakinkan tidak pernah terjadi kondisi circular-wait
  - Alokasi resource "state" didefinisikan dengan jumlah resource yang tersedia dan resource yang dialokasikan dan kebutuhan maksimum dari proses

Safe State (1)

- Status dikatakan safe apabila sistem dapat mengalokasikan resource untuk tiap-tiap proses (sampai maksimumnya) dalam beberapa urutan dan masih dapat mencegah terjadinya deadlock
- Safe state bukan deadlock state. Sebaliknya deadlock state berada pada unsafe state.
  - Tidak semua unsafe state terjadi deadlock, tetapi unsafe state menuntun terjadinya deadlock
  - Selama safe state, OS dapat menghindari unsafe state
  - Pada unsafe state, OS tidak mencegah proses dari permintaan resource sehingga terjadi deadlock. Dalam hal ini kelakuan proses menyebabkan unsafe state

Safe State (2)



Safe State (3)

- Misalnya sistem mempunyai 12 magnetic tape drive dan 3 proses, P_0, P_1,P_2

Proses	Max need	Current need
P_0	10	5
P_1	4	2
P_2	9	2

- Pada saat t_0, sistem dalam safe state dengan urutan proses $<P_1,P_0,P_2>$
- Perubahan kondisi dari safe state ke unsafe safe, misalnya pada t_1 proses P_2 meminta dan dialokasikan tambahan 1 tape drive, hanya P_1 yang dapat dipenuhi

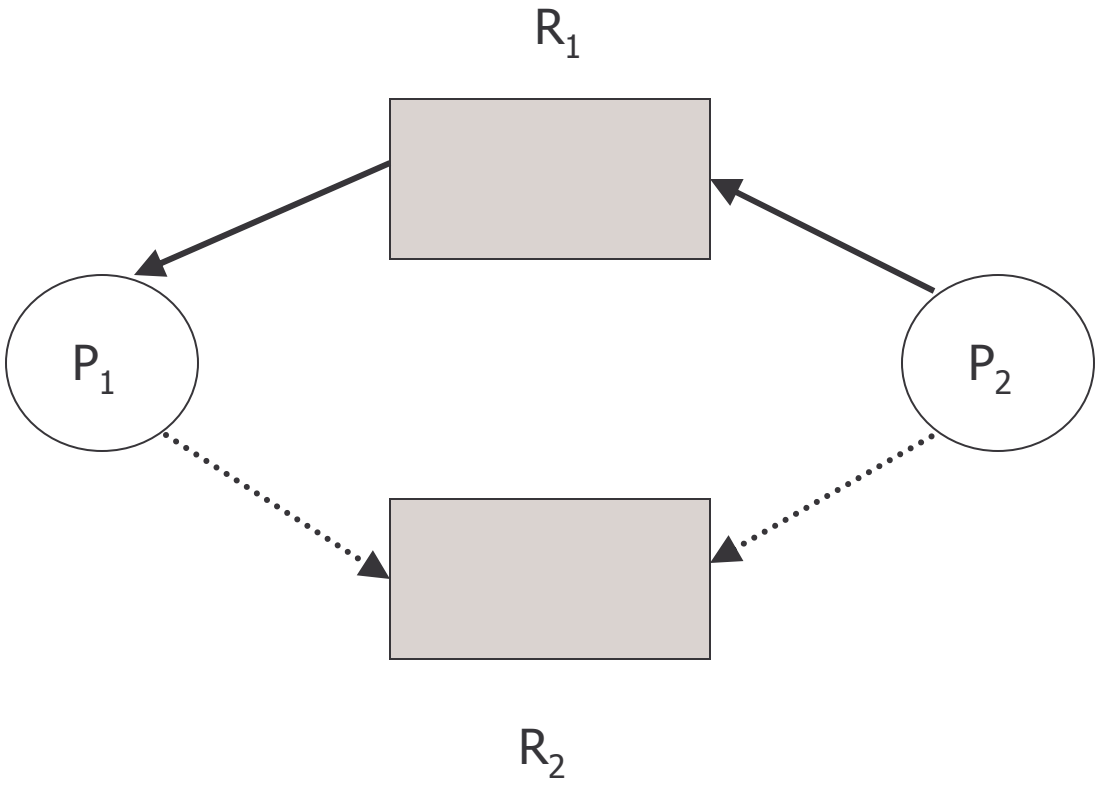
Algoritma Deadlock-Avoidance

- **Algoritma Resource-allocation graph**
  - Algoritma ini dapat diaplikasikan pada sistem yang terdiri dari beberapa proses dan sejumlah tipe resource, masing-masing mempunyai single instance
- **Algoritma Banker**
  - Algoritma ini diaplikasikan untuk sembarang sistem
  - Nama algoritma dipilih karena dapat digunakan pada sistem banking untuk menjamin bahwa bank tidak pernah mengalokasikan uang kontan yang tersedia jika tidak memenuhi kebutuhan semua customer

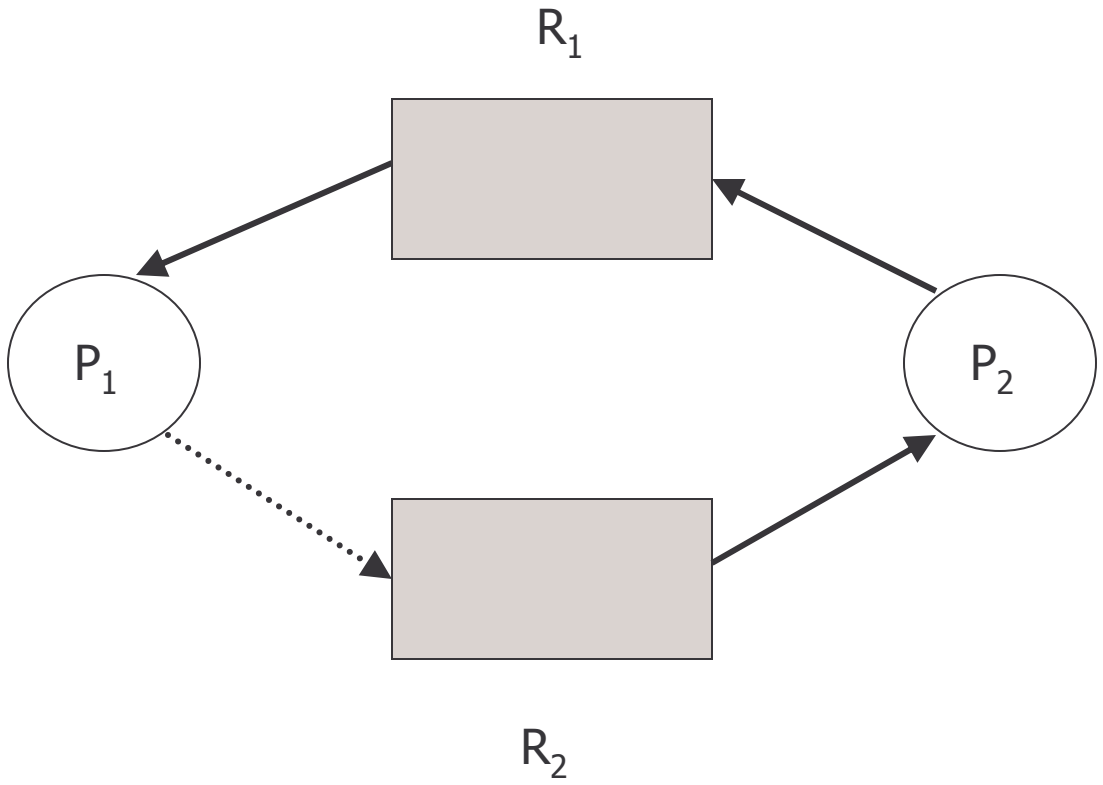
Algoritma Resource-Allocation Graph (1)

- Ditunjukkan dengan resource-allocation graph
- Claim edge $P_i \rightarrow R_j$ menunjukkan bahwa proses P_i mungkin membutuhkan R_j suatu saat nanti
  - Sama dengan request edge
  - Dibuat dengan garis putus-putus
  - Bila R_j dilepas oleh P_i, assignment edge $R_j \rightarrow P_i$ dikonversikan kembali ke claim edge

Algoritma Resource-Allocation Graph (2)



Unsafe State pada Resource-Allocation Graph (3)



Algoritma Banker

- Misalkan ada n proses dalam sistem dan m tipe resource, maka struktur data yang dibutuhkan :
  - *Available* : suatu vektor dengan panjang m yang menunjukkan resource-resource yang tersedia untuk setiap tipe
  - *Max* : matriks n X m yang mendefinisikan maksimum permintaan untuk tiap-tiap proses
  - *Allocation* : matriks n X m yang mendefinisikan jumlah resource yang sedang dialokasikan untuk setiap proses
  - *Need* : matriks n X m yang menunjukkan sisa resource yang dibutuhkan untuk tiap-tiap proses
- Bila X dan Y adalah vektor dengan panjang i. $X \leq Y$ jika dan hanya jika $X[i] \leq Y[i]$ untuk setiap $i=1,2,\ldots,n$
- Contoh jika $X=(1,7,3,2)$ dan $Y=(0,3,2,1)$ maka $Y \leq X$. $Y < X$ jika $Y \leq X$ dan $Y \neq X$

Algoritma Safety

- Algoritma untuk menentukan apakah sistem dalam safe state :
  1. Vektor *Work* dan *Finish* dengan panjang *m* dan *n*. Inisialisasi *Work :=Available* dan *Finish[i] := false* for *i=1, 2, …, n*
  2. Temukan i sehingga
     a. *Finish[I] := false*
     b. $Need_i \leq Work$

     Jika tidak ada i, ke langkah 4
  3. *Work := Work + Allocation;*

     Finish[i] := true

     ke langkah 2
  4. Jika *Finish[i] := true* untuk semua i, maka sistem dalam safe state

Algoritma Resource-Request

- $Request_i$ adalah vektor request untuk proses P_i. Jika $Request_i[j] = k$, maka proses P_i menginginkan k instance dari tipe resource R_j. Bila request untuk resource dibuat oleh proses P_i maka dilakukan aksi berikut :
  1. Jika $Request_i \leq Need_i$, ke langkah 2. Sebaliknya, terjadi kondisi error, karena proses melebihi maksimum resource
  2. Jika $Request_i \leq Available$, ke langkah 3. Sebaliknya P_i harus menunggu, karena resource tidak tersedia
  3. Apakah sistem dialokasikan requested resource untuk proses P_i dengan memodifikasi state berikut :

     $Available := Available - Request_i;$

     $Allocation_i := Allocation_i + Request_i;$

     $Need_i := Need_i - Request_i;$

     Jika hasil state resource-allocation safe, transaksi selesai dan proses P_i dialokasikan untuk resource tersebut. Tetapi jika state baru unsafe, maka P_i harus menunggu $Request_i$ dan state resource-allocation disimpan

Kelemahan Algoritma Banker

- Proses-proses kebanyakan belum dapat mengetahui berapa jumlah resource maksimum yang dibutuhkan
- Jumlah proses tidak tetap
- Beberapa resource kadang bisa diambil dari sistem sewaktu-waktu, sehingga meskipun secara teoritis ada, namun kenyataannya tidak tersedia
- Proses-proses seharusnya berjalan secara terpisah, sehingga urutan eksekusi proses tidak dibatasi oleh kebutuhan sinkronisasi antar proses
- Algoritma menghendaki untuk memberikan semua permintaan hingga waktu yang tidak terbatas
- Algoritma menghendaki client-server mengembalikan resource setelah batas waktu tertentu

Pendeteksian Deadlock

- Jika sistem tidak menggunakan baik algoritma deadlock-prevention atau deadlock-avoidance maka situasi deadlock dapat terjadi. Pada sebuah lingkungan sistem harus menyediakan
  - Algoritma yang mengetes state dari sistem untuk menentukan apakah terjadi deadlock
  - Algoritma untuk memperbaiki deadlock
- Terdapat dua algoritma untuk mendeteksi deadlock
  - Algoritma yang dapat diaplikasikan untuk sistem yang terdiri dari single instance untuk setiap tipe resource
  - Algoritma yang dapat diaplikasikan untk sembarang sistem (misalnya sistem yang terdiri dari beberapa instance dari tipe resource). Algoritma yang digunakan mirip dengan algoritma banker yang digunakan untuk menghindari deadlock.
- Algoritma deteksi :
  - Kapan algoritma deteksi digunakan ? Jawabannya tergantung pada dua faktor berikut :
    - Seberapa sering terjadi deadlock ?
    - Berapa banyak proses yang dilibatkan bila deadlock terjadi?

Perbaikan dari Deadlock

- Terminasi proses :
  - Abort semua proses deadlock
  - Abort satu proses pada satu waktu sampai cycle dieliminasi
- Resource preemption
  - Untuk memperkecil deadlock menggunakan resource preemption, beberapa resource harus preempted berurutan dari proses dan memberikan proses ke proses lain sampai cycle deadlock putus
  - Jika preemption dibutuhkan sehubungan dengan deadlock, maka dipertimbangkan isu berikut :
    - Memilih korban : resource dan proses mana yang di preempted
    - Rollback : jika resource preempted dari proses, apakah yang terjadi dengan proses tersebut ? Tentunya, tidak dapat melakukan eksekusi normal karena kehilangan resource yang dibutuhkan. Maka harus roll back proses ke safe state yang sama, dan restart dari state tersebut
    - Starvation : bagaimana meyakinkan starvation tidak terjadi ? Bagaimana menjamin resource tidak selalu preempted dari proses yang sama