Deadlock (1)

Deadlock

- Deadlock mungkin terjadi pada lingkungan multiprogramming, dimana sejumlah proses berkompetisi untuk sejumlah resource yang terbatas.
- Dalam hal ini, proses mungkin meminta beberapa resource. Jika resource tidak tersedia saat itu, proses harus menunggu (masuk ke wait state)
- Pada saat proses menunggu tidak pernah berubah state, karena resource yang diminta digunakan oleh proses yang menunggu lainnya.
- Situasi ini disebut “Deadlock”

Model Sistem (1)

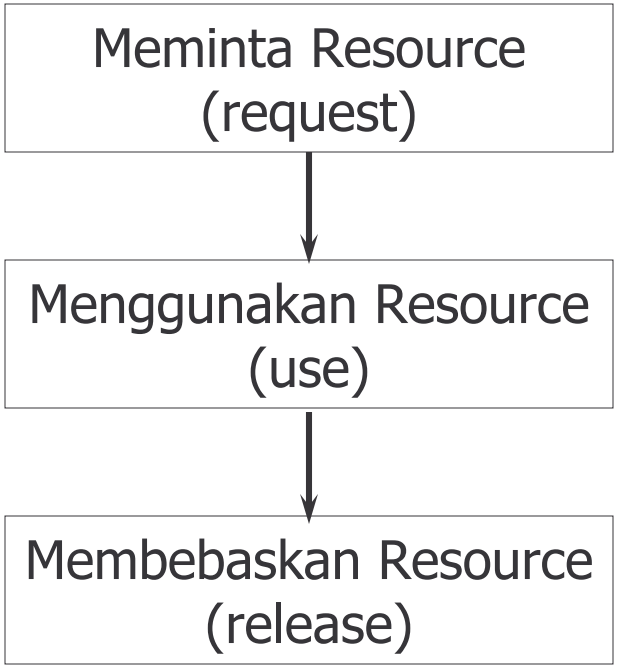
- Sebuah sistem terdiri dari sejumlah resource terbatas yang didistribusikan diantara proses yang saling berkompetisi
- Resource dipartisi dalam beberapa tipe, masing-masing terdiri dari sejumlah *identical instance*
- Proses harus meminta resource sebelum menggunakannya dan harus melepaskan resource setelah menggunakannya.
  - Sebuah proses mungkin meminta beberapa resource sesuai kebutuhan untuk menyelesaikan pekerjaannya
  - Sejumlah resource yang diminta mungkin tidak memenuhi jumlah resource total yang tersedia pada sistem
- Sekumpulan proses dikatakan berada pada "*deadlock state*" bila setiap proses menunggu event yang disebabkan oleh proses lain dalam kumpulan tersebut
  - Event utama berupa *resource acquisition* dan *resource release*
  - Resource bisa berupa resource fisik (printer, tape drive, memory space dan CPU cycle) atau resource logika (file dan semaphore)

Model Sistem (2)

- Deadlock terjadi pada saat proses akan mengakses obyek
- Obyek tsb disebut resource
- Berupa hardware device (tape drive, memori, printer) atau informasi (record dalam suatu basis data, variabel global)
- Jenis resource : preemptable dan nonpreemptable
- Preemptable jika resource tsb dapat diambil (dilepas) dari proses yang memakainya tanpa memberi efek apapun pada proses tsb, contoh : memori
- Nonpreemptable jika resource tidak dapat diambil dari proses yang sedang membawanya karena akan menimbulkan kegagalan komputasi, contoh : printer

Model Sistem (3)

- Urutan untuk mengakses resource



Model sistem (4)

- Deadlock mungkin melibatkan proses yang berkompetisi untuk tipe resource yang sama, misalnya
  - Sistem mempunyai 3 tape drive, dan terdapat 3 proses masing-masing membawa tape drive
  - Bila setiap proses meminta tape drive tambahan, ketiga proses akan berada pada *deadlock state*
  - Setiap proses menunggu event "tape drive is released" yang disebabkan oleh satu dari waiting process.
- Deadlock mungkin melibatkan tipe resource yang berbeda, misalnya
  - Sistem dengan satu printer dan satu tape drive. Misalnya proses *P* membawa tape drive dan proses *Q* membawa printer. Jika *P* meminta printer dan *Q* meminta tape drive maka terjadi deadlock

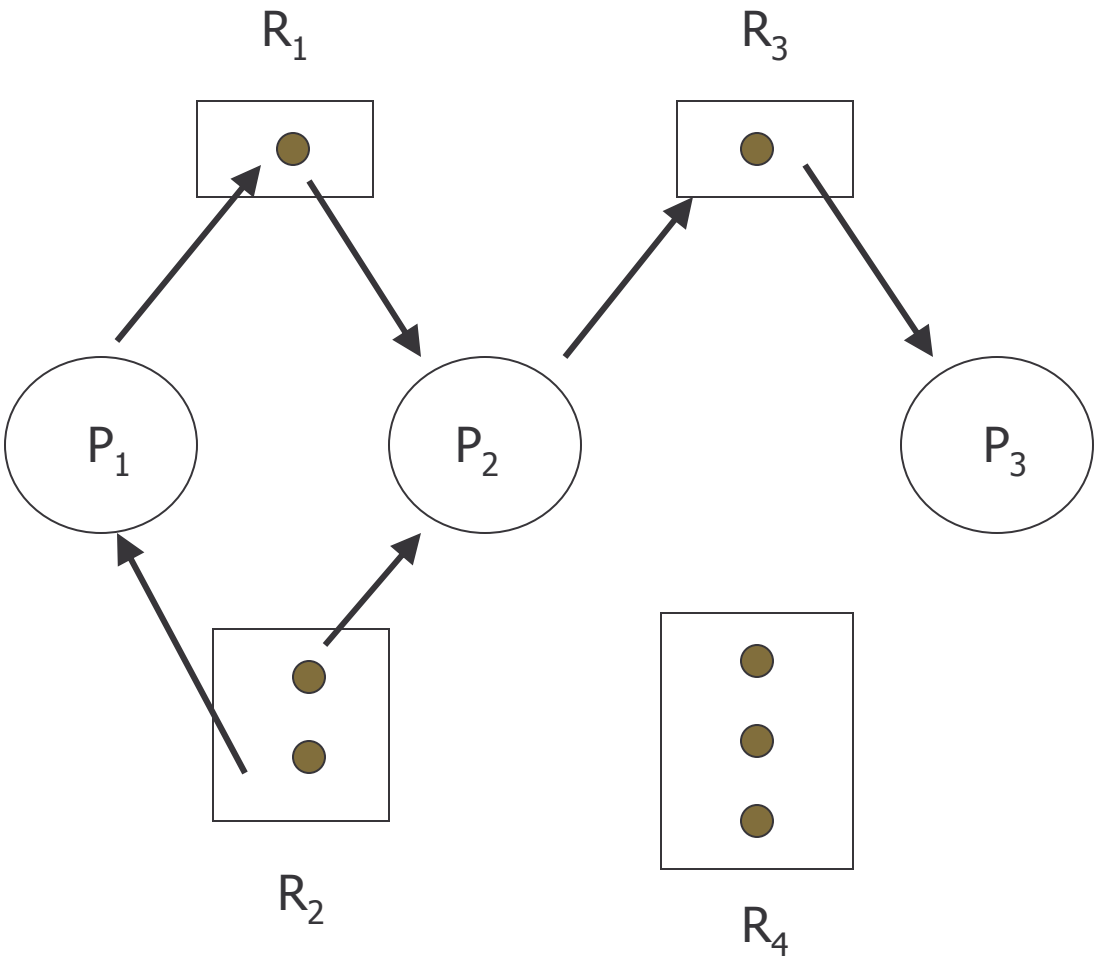
Karakteristik Deadlock

- Situasi deadlock dapat terjadi bila terdapat 4 kondisi yang berjalan serentak pada sistem :
  - **Mutual Exclusion** : setidaknya satu resource harus diadakan mode non-sharable, sehingga hanya satu proses pada satu waktu yang dapat menggunakan resource
  - **Hold & wait** : harus terdapat satu proses yang membawa setidaknya satu resource dan menunggu resource tambahan yang digunakan proses lain
  - **No preemption** : resource tidak bersifat preemptive, sehingga resource dapat dilepas hanya oleh proses yang membawanya, setelah proses menyelesaikan task
  - **Circular wait** : terdiri dari sekulmpulan proses yang menunggu $\{P_0, P_1, \ldots, P_n\}$, jika proses P_0 menunggu resource yang dibawa P_1, P_1 menunggu resource yang dibawa P_2 , P_{n-1} menunggu resource yang dibawa P_n dan P_n menunggu resource yang dibawa P_0

Resource Allocation Graph (RAG)

- Adalah graph berarah yang digunakan untuk menggambarkan deadlock lebih presisi. RAG terdiri dari himpunan vertek dan edge
- Himpunan vertek *V* dibagi dua tipe
  - $P = \{P_1, P_2, \ldots, P_n\}$ himpunan semua proses dalam sistem
  - $R = \{R_1, R_2, \ldots, R_n\}$ himpunan tipe resource dalam sistem
- Himpunan edge dikelompokkan dalam 2 tipe
  - **Request Edge** : menghubungkan proses P_i dengan resource R_j dengan $P_i \rightarrow R_j$ artinya proses P_i meminta R_j dan proses P_i sedang menunggu R_j
  - **Assignment Edge** : menghubungkan resource R_j dengan proses P_i dengan $R_j \rightarrow P_i$ artinya anggota R_j sedang dibawa proses P_i

Contoh RAG (1)



Contoh RAG (2)

- Himpunan P, R dan E :

  $P = \{P_1, P_2, P_3\}$;

  $R = \{R_1/1, R_2/2, R_3/1, R_4/3\}$;

  $E = \{ P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$;
- Anggota resource :

  Satu anggota resource R_1;

  Dua anggota resource R_2;

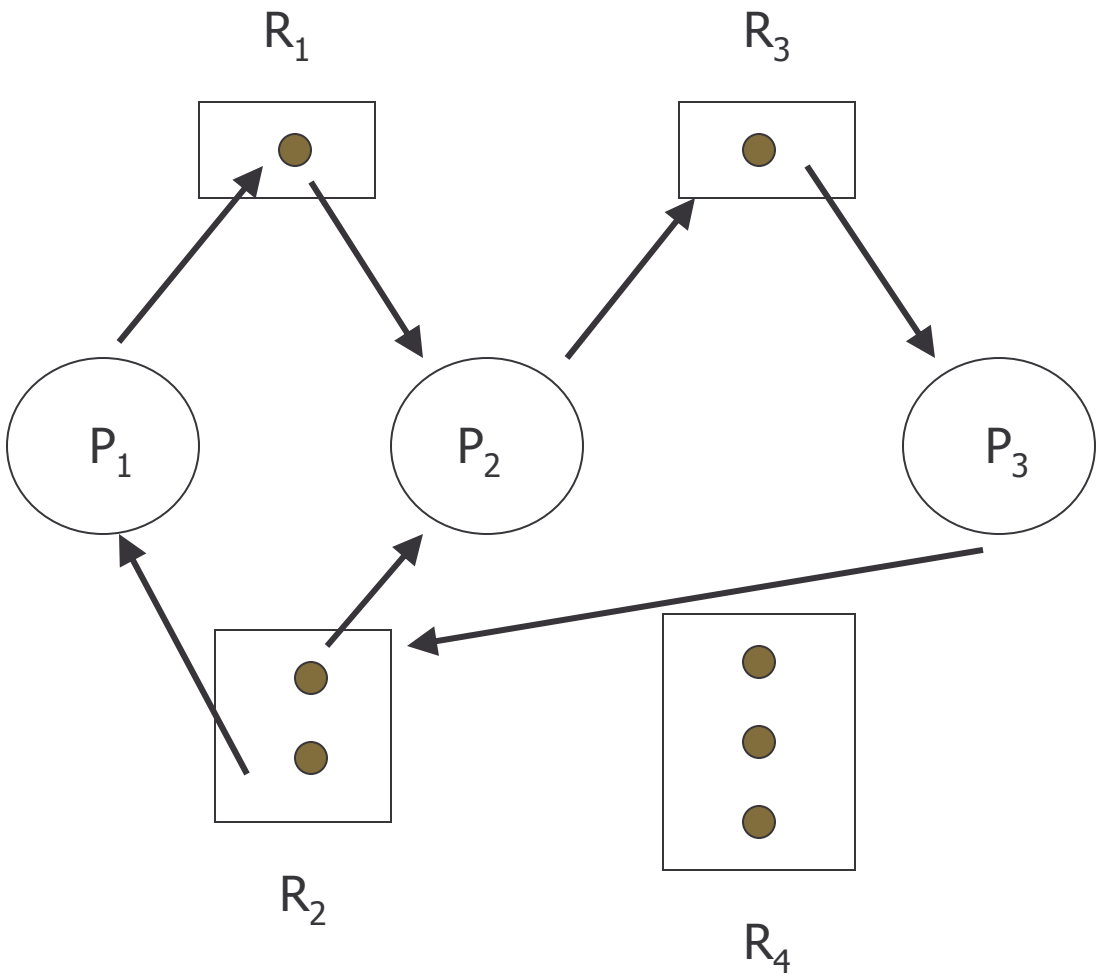
  Satu anggota resource R_3;

  Tiga anggota resource R_4;
- Status proses :
  - Proses P_1 membawa satu anggota R_2 dan menunggu mendapatkan satu anggota R_1
  - Proses P_2 membawa satu anggota R_1 dan R_2 dan menunggu mendapatkan satu anggota R_3
  - Proses P_3 membawa satu anggota R_3

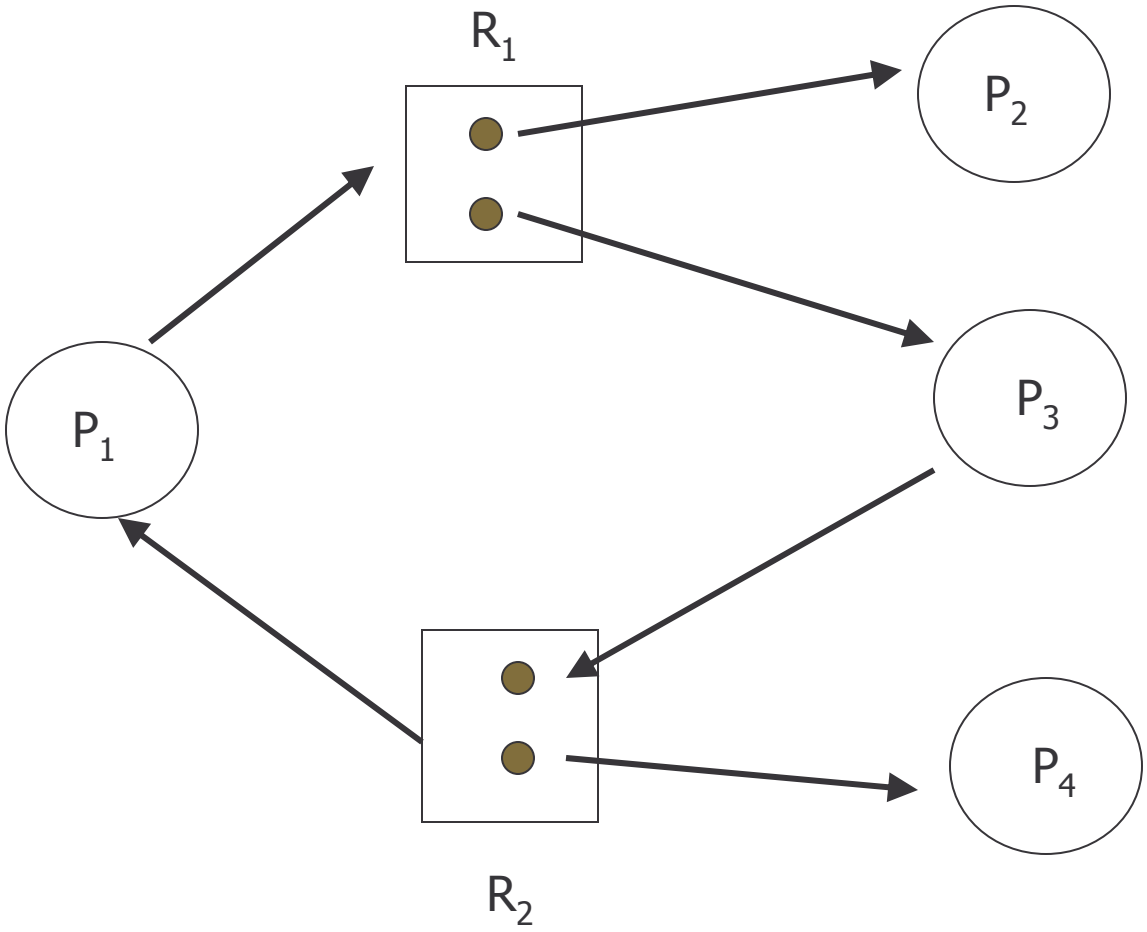
Contoh RAG (3)

1. Jika graph tidak mempunyai siklus, tidak ada proses pada sistem yang deadlock. Sebaliknya, jika graph terdapat siklus deadlock mungkin terjadi
2. Jika setiap tipe resource mempunyai tepat satu instance, maka siklus menunjukkan terjadi deadlock. Dengan kata lain, jika siklus melibatkan hanya satu tipe resource masing-masing mempunyai satu instance, maka terjadi deadlock. Setiap proses yang dilibatkan pada siklus dalam keadaan deadlock.
   - Dalam hal ini siklus pada graph merupakan kondisi perlu dan cukup untuk adanya deadlock
3. Jika setiap tipe resource mempunyai beberapa instance, maka siklus tidak "perlu" menandakan terjadi deadlock
   - Dalam hal ini siklus pada graph merupakan kondisi perlu tapi tidak cukup untuk adanya deadlock

Contoh RAG dengan Deadlock



Contoh RAG dg siklus tetapi tidak terjadi deadlock



Metode Mengendalikan Deadlock (1)

- Mengadopsi metode dibawah ini :
  1. Menggunakan suatu protokol untuk meyakinkan bahwa sistem tidak akan pernah mengalami deadlock
  2. Mengijinkan sistem mengalami deadlock, tapi kemudian segera memperbaikinya
  3. Mengabaikan semua permasalahan bersama-sama dan menganggap bahwa deadlock tidak akan pernah terjadi
- Menggunakan skema "deadlock prevention" atau "deadlock avoidance" untuk meyakinkan deadlock tidak pernah terjadi
  - *Deadlock prevention* adalah sekumpulan method untuk meyakinkan setidaknya tidak terjadi satu dari kondisi yang menyebabkan deadlock
  - *Deadlock avoidance* bahwa OS diberikan informasi tambahan yang menyatakan resource mana yang diminta proses dan digunakan selama proses berjalan. Dengan informasi tambahan, keputusan dapat dibuat untuk setiap permintaan perlu tidaknya proses menunggu

Metode Mengendalikan Deadlock (2)

- Pada sistem yang tidak menggunakan algoritma yang mencegah atau menghindari deadlock, maka deadlock mungkin terjadi. Pada lingkungan ini, sistem dapat menyediakan algoritma yang mengetes state dari sistem untuk menentukan apakah deadlock terjadi dan algoritma untuk memperbaiki dari deadlock
- Jika sistem tidak yakin deadlock tidak pernah terjadi dan juga tidak menyediakan mekanisme untuk mendeteksi deadlock dan memperbaikinya, maka situasi dimana sistem dalam state deadlock tetapi tidak ada jalan untuk mengenali apa yang mungkin terjadi

Pencegahan Deadlock (1)
MUTUAL EXCLUSION

- Kondisi ini tergantung pada sifat alamiah resource, karena kondisi mutual-exclusion harus memegang resource non-sharable
  - Misalnya : sebuah printer tidak dapat secara serentak digunakan untuk beberapa proses
- Resource yang dapat digunakan bersama-sama (*sharable resource*) tidak membutuhkan akses mutual exclusion dan karenanya tidak dapat dilibatkan dalam deadlock. Sebuah proses tidak perlu menunggu untuk *sharable resource*.
  - Misalnya : file read-only dimana jika sejumlah proses akan membuka file read-only pada waktu yang bersamaan, dijamin akses ke file bisa serentak

Pencegahan Deadlock (2)
HOLD AND WAIT

- Kondisi hold an wait dapat dicegah terjadi dengan menjamin jika sebuah proses meminta sebuah resource, proses tersebut tidak boleh memegang resource. Protokol yang dapat diadopsi
  1. Setiap proses harus meminta semua resource yang dibutuhkan dan dialokasikan semua resource tersebut sebelum eksekusi dimulai
  2. Alternatif lain, sistem hanya mengijinkan sebuah proses untuk meminta resource jika proses tidak memegang resource. Dalam hal ini, sebuah proes mungkin meminta beberapa resource dan menggunakannya. Sebelum memintah resource tambahan, proses harus melepaskan semua resource yang dialokasikan untuknya
- Kedua protokol diatas mempunyai kelemahan :
  - *Resource utilization* menjadi rendah karena beberapa resource dialokasikan tetapi tidak digunakan untuk waktu yang lama
  - *Starvation* terjadi karena proses yang memerlukan beberapa resource populer harus menunggu, karena sedikitnya satu dari resource tsb dialokasikan untuk proses yang lain.

Pencegahan Deadlock (3)
NO PREEMPTION

- Agar tidak terjadi kondisi ini, dapat menggunakan satu dari dua protokol berikut :
  1. Jika proses memegang beberapa resource meminta resource lain yang tidak dapat segera dialokasikan, maka semua resource yang dipegang harus preemptive.
  2. Alternatif lain, jika proes meminta beberapa resource :
     - Pertama dicek apakah tersedia, jika tersedia maka dialokasikan
     - Jika tidak, cek apakah resource tsb dialokasikan untuk beberapa proses lain yang menunggu resource tambahan. Jika ya, preemptive-kan resource dari proses yang menunggu dan mengalokasikan untuk proses yang meminta
     - Jika resource tidak tersedia atau dipegang oleh proses yang menunggu resource, proses yang meminta harus menunggu. Sementara menunggu beberapa resource di-preemptive-kan, tetapi hanya jika proses lain memintanya.

Pencegahan Deadlock (4)
CIRCULAR WAIT

- Satu metode untuk meyakinkan kondisi ini tidak pernah terjadi adalah melakukan "total ordering" untuk semua tipe resource dan proses yang meminta resource harus menaikkan order
- Setiap tipe resource ditandai sebuah angka integer unik yang membandingkan dua resource dan menentukan satu resource mempunyai urutan terdahulu.
  - Contoh : tipe resource terdiri dari tape drive, disk drive dan printer, maka order didefinisikan sbb :
    F(tape drive) = 1,
    F(disk drive) = 5,
    F(printer) = 12.
- Protokol yang digunakan untuk mencegah deadlock :
  - Setiap proses dapat meminta resource hanya dg menaikkan order. Sebuah proses menginisialisasi permintaan misalnya R_i ,maka proses dapat meminta resource R_j jika dan hanya jika $F(R_j) > F(R_i)$
  - Alternatif lain, permintaan dari proses untuk suatu tipe resource R_j dipenuhi jika dan hanya jika proses melepas resource R_i sehingga $F(R_i) \geq F(R_j)$