Sistem Operasi

Catatan Kuliah #6

Alauddin Maulana Hirzan, M. Kom
0607069401

Manajemen Proses - *Konkurensi: Deadlock dan Starvation*

Deadlock

Deadlock

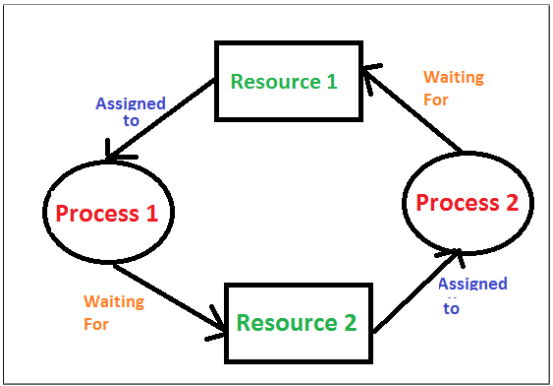
Apa itu *Deadlock*?

Dalam sistem operasi, **Deadlock** terjadi ketika dua atau lebih proses diblokir, menunggu satu sama lain untuk melepaskan sumber daya yang mereka pegang, yang mengakibatkan keadaan pemblokiran permanen. Hal ini dapat terjadi ketika beberapa proses bersaing untuk mendapatkan sumber daya yang sama, seperti memori atau perangkat I/O, dan mereka mendapatkan sumber daya tersebut dengan cara yang mencegah proses lain untuk mengaksesnya.

Deadlock

Ilustrasi Deadlock

Misalkan proses A telah mendapatkan sumber daya X dan menunggu sumber daya Y dilepaskan oleh proses B, sementara proses B telah mendapatkan sumber daya Y dan menunggu sumber daya X dilepaskan oleh proses A.



Deadlock

Penyebab Deadlock #1

Deadlock disebabkan oleh 4 hal yang terjadi bersamaan:

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

Deadlock

Penyebab Deadlock #2 - Mutual Exclusion

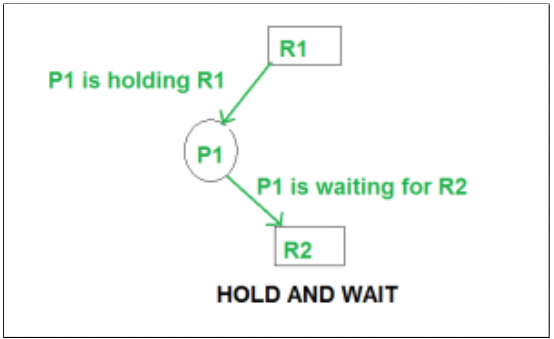
Mutual exclusion dapat menjadi penyebab kebuntuan dalam sistem operasi jika tidak dikelola dengan benar. Mutual exclusion adalah mekanisme yang memastikan bahwa hanya satu proses yang dapat mengakses sumber daya bersama pada satu waktu

Deadlock dapat terjadi ketika beberapa proses bersaing untuk mendapatkan sumber daya yang sama, dan setiap proses memegang beberapa sumber daya yang dibutuhkan oleh proses lain.

Deadlock

Penyebab Deadlock #3 - Hold and Wait

Hold and wait adalah kondisi di mana sebuah proses memegang satu sumber daya sementara secara bersamaan menunggu sumber daya lain yang sedang dipegang oleh proses lain.

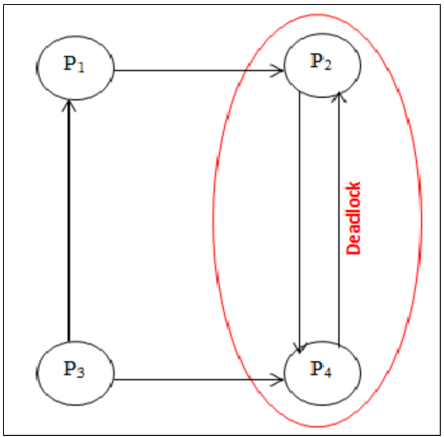


Deadlock

Penyebab Deadlock #4 - No Preemption

Preemption adalah menghentikan sementara sebuah tugas yang sedang dijalankan dan kemudian melanjutkannya.

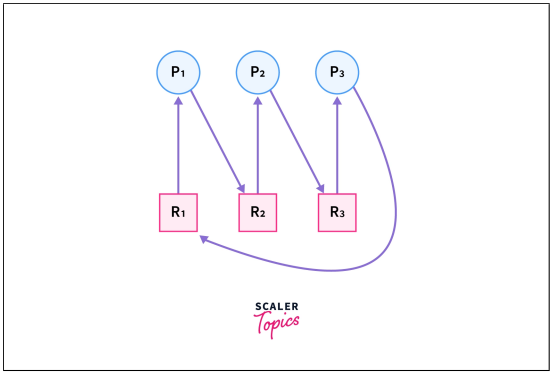
Misalnya, jika proses P1 menggunakan sumber daya dan proses P2 yang memiliki prioritas tinggi meminta sumber daya tersebut, proses P1 dihentikan dan sumber daya dialokasikan ke P2.



Deadlock

Penyebab Deadlock #5 - Circular Wait

Sesuai dengan namanya, **Circular Wait** terjadi karena adanya saling menunggu yang melingkar satu sama lainnya.



Deadlock

Jenis-jenis Deadlock #1

Deadlock memiliki berbagai macam jenis:

- Resource Deadlock
- Communication Deadlock
- Livelock
- Self-Deadlock
- Deadly Embrace

Deadlock

Jenis-jenis Deadlock #2

Resource Deadlock : jenis ini yang paling umum, dan ini terjadi ketika dua atau lebih proses menunggu sumber daya yang sama tersedia, yang dipegang oleh proses lain. Dalam skenario ini, semua proses yang terlibat tidak dapat dilanjutkan, dan sistem memasuki kondisi deadlock.

Communication Deadlock : Jenis ini terjadi ketika dua atau lebih proses menunggu pesan dari proses lain, dan pesan tersebut tidak pernah sampai. Dalam skenario ini, proses yang terlibat mungkin menunggu pesan tanpa batas waktu, yang dapat mengakibatkan situasi kebuntuan.

Deadlock

Jenis-jenis Deadlock #3

Livelock: Hal ini terjadi ketika dua atau lebih proses secara aktif mencoba menyelesaikan situasi kebuntuan, tetapi upaya mereka hanya mengakibatkan mereka berulang kali memblokir satu sama lain.

Self-Deadlock: Hal ini terjadi ketika satu proses mencoba mendapatkan beberapa sumber daya dengan urutan yang salah, menyebabkan proses menjadi macet dan tidak dapat dilanjutkan.

Deadly Embrace: Hal ini terjadi ketika dua atau lebih proses memegang sumber daya dan menunggu sumber daya tambahan yang dipegang oleh proses lain.

Deadlock

Mitigasi Deadlock

Mitigasi deadlock mengacu pada teknik dan strategi yang digunakan untuk mencegah, mendeteksi, dan menyelesaikan deadlock dalam sistem operasi. Ada beberapa pendekatan untuk mitigasi kebuntuan, termasuk:

- **Deadlock Prevention**
- **Deadlock Avoidance**
- **Dadlock Detection**
- **Deadlock Recovery**

Deadlock

Mitigasi Deadlock - **Prevention**

Salah satu cara untuk mencegah deadlock adalah dengan menghindari kondisi-kondisi yang dapat menyebabkan deadlock. Hal ini dapat dilakukan dengan memberlakukan urutan permintaan sumber daya yang ketat, membatasi jumlah maksimum sumber daya yang dapat dipegang oleh setiap proses, dan menggunakan mekanisme batas waktu untuk memastikan bahwa proses yang menunggu sumber daya tidak menunggu tanpa batas waktu.

Informasi
Tidak mungkin untuk menghilangkan **Mutual Exclusion**, karena beberapa sumber daya pada dasarnya tidak dapat dibagikan,

Deadlock

Mitigasi Deadlock - **Avoidance**

Cara lain untuk mengurangi kebuntuan adalah dengan menggunakan algoritme alokasi sumber daya yang dapat memprediksi dan menghindari potensi kebuntuan. Salah satu algoritme tersebut adalah algoritme Banker, yang menentukan apakah permintaan sumber daya akan menghasilkan kondisi yang aman (yaitu kondisi di mana tidak ada kebuntuan yang bisa terjadi) sebelum mengabulkan permintaan tersebut.

Deadlock

Mitigasi Deadlock - **Detection**

Deteksi kebuntuan melibatkan pemeriksaan keadaan sistem secara berkala untuk menentukan apakah kebuntuan telah terjadi. Setelah kebuntuan terdeteksi, tindakan yang tepat dapat diambil untuk mengatasinya, seperti mematikan satu atau beberapa proses, melepaskan sumber daya, atau memutar kembali transaksi.

Deadlock

Mitigasi Deadlock - **Recovery**

Pemulihan kebuntuan melibatkan pengambilan tindakan korektif setelah kebuntuan terdeteksi. Salah satu pendekatannya adalah menghentikan satu atau lebih proses yang terlibat dalam kebuntuan dan melepaskan sumber daya yang mereka pegang. Pendekatan lainnya adalah mengembalikan transaksi dari satu atau beberapa proses ke titik waktu sebelumnya ketika proses tersebut tidak memiliki sumber daya apa pun.

Deadlock

Mitigasi Deadlock - Penanganan **Hold and Wait**

Dengan menghilangkan *Hold*

Proses menentukan sumber daya yang dibutuhkan sebelumnya sehingga tidak perlu menunggu alokasi setelah eksekusi dimulai.

Dengan menghilangkan *Wait*

Proses harus melepaskan semua sumber daya yang sedang dipegangnya sebelum membuat permintaan baru.

Deadlock

Mitigasi Deadlock - Penanganan **No Preemption**

Ada dua cara untuk menghilangkan kondisi ini dengan melakukan preemption:

Jika sebuah proses menahan beberapa sumber daya dan menunggu sumber daya lainnya, maka proses tersebut harus melepaskan semua sumber daya yang ditahan sebelumnya dan mengajukan permintaan baru untuk sumber daya yang dibutuhkan lagi. Proses dapat dilanjutkan kembali setelah memiliki semua sumber daya yang dibutuhkan.

Jika sebuah proses P1 sedang menunggu beberapa sumber daya, dan ada proses lain P2 yang memegang sumber daya tersebut dan diblokir karena menunggu sumber daya lain. Maka sumber daya diambil dari P2 dan dialokasikan ke P1.

Deadlock

Mitigasi Deadlock - Penanganan **Circular Wait**

Untuk menghilangkan **Circular Wait**, programmer menetapkan prioritas untuk setiap sumber daya. Sebuah proses hanya dapat meminta sumber daya dalam urutan prioritas yang meningkat.

Pada contoh di atas, proses P3 meminta sumber daya R1, yang memiliki nomor lebih rendah dari sumber daya R3 yang sudah dialokasikan untuk proses P3. Jadi permintaan ini tidak valid dan tidak dapat dilakukan, karena R1 sudah dialokasikan untuk proses P1.

Starvation

Starvation

Apa itu **Starvation**?

Starvation adalah situasi yang dapat terjadi pada sistem operasi ketika sebuah proses atau sekelompok proses dicegah untuk mengakses sumber daya bersama atau layanan sistem, meskipun mereka siap dan menunggu untuk melakukannya.

Hal ini dapat terjadi jika proses lain diberi prioritas akses ke sumber daya atau layanan, menyebabkan beberapa proses dibiarkan menunggu tanpa batas waktu. Dengan kata lain, starvation terjadi ketika sebuah proses tidak dapat berjalan karena terus menerus ditolak untuk mengakses sumber daya yang dibutuhkannya.

Starvation

Kapan **Starvation** Terjadi? #1

Starvation dapat terjadi dalam berbagai skenario, seperti ketika proses dengan prioritas tinggi terus-menerus memperoleh sumber daya, mencegah proses dengan prioritas lebih rendah untuk mengaksesnya, atau ketika proses terjebak dalam satu lingkaran dan terus-menerus memperoleh sumber daya tanpa melepaskannya.

Starvation dapat menyebabkan kinerja sistem menurun dan dapat menyebabkan proses dengan prioritas lebih rendah diblokir tanpa batas waktu, yang mengakibatkan berkurangnya throughput sistem.

Starvation

Kapan **Starvation** Terjadi? #2

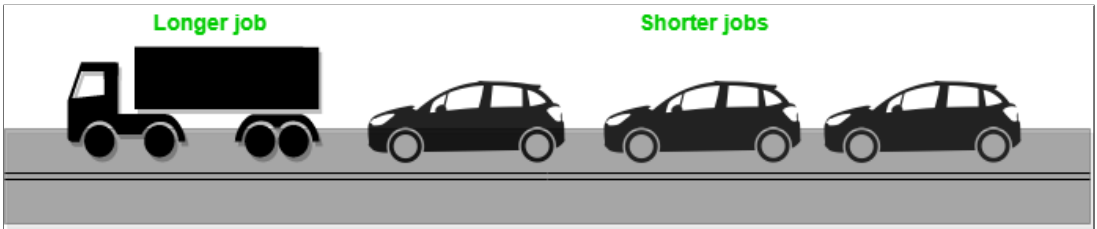
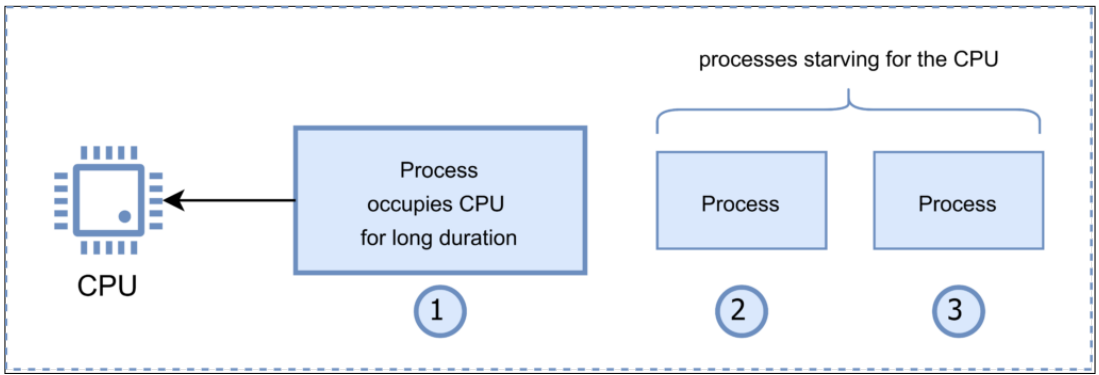


Figure - The Convey Effect, Visualized

Starvation

Kapan **Starvation** Terjadi? #3



Starvation

Mitigasi **Starvation**

Untuk mengurangi risiko kelaparan, sistem operasi sering kali menggunakan berbagai algoritma penjadwalan yang bertujuan untuk mengalokasikan sumber daya dengan cara yang adil dan efisien. Hal ini dapat melibatkan prioritas proses berdasarkan faktor-faktor seperti tingkat prioritas, waktu yang dihabiskan untuk menunggu sumber daya, atau jumlah sumber daya yang dimiliki saat ini.

Dengan menggunakan algoritme tersebut, sistem operasi dapat memastikan bahwa proses diberikan akses yang adil ke sumber daya bersama, meminimalkan risiko kelaparan dan memastikan pengoperasian sistem yang efisien.


THANK YOU