

LAPORAN PRAKTIKUM SISTEM OPERASI

DEADLOCK



Agus Pranata Marpaung

13323033

DIII TEKNOLOGI KOMPUTER

**INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI**

Judul Praktikum

Minggu/Sesi	:	X/2 dan 3
Kode Mata Kuliah	:	1031202
Nama Mata Kuliah	:	SISTEM OPERASI
Setoran	:	Laporan Materi Deadlock dikirimkan dalam bentuk PDF dengan aturan penamaan file adalah NIM_Laporan_Materi_Deadlock.
Batas Waktu Setoran	:	?????
Tujuan	:	<ol style="list-style-type: none">1. Mampu mendefinisikan model sistem deadlock2. Mampu menjelaskan karakterisasi kondisi-kondisi yang menyebabkan deadlock3. Mampu menjelaskan penggunaan resource allocation graph untuk pengenalan deadlock4. Mampu mengelompokkan metode penanganan deadlock ke dalam metode pencegahan (prevention) dan penghindaran (avoidance)5. Mampu menjabarkan algoritma-algoritma untuk penghindaran deadlock6. Mampu menjelaskan perbedaan 2 teknik pemulihan deadlock yakni: termination dan preemption.7. Mampu menjelaskan algoritma-algoritma untuk deteksi deadlock8. Mampu menjelaskan perbedaan 2 teknik pemulihan deadlock yakni: termination dan preemption

Petunjuk

1. Anda dapat mengerjakan tugas ini secara individu.
2. Mencontoh pekerjaan dari orang lain akan dianggap plagiarisme dan anda akan ditindak sesuai dengan sanksi akademik yang berlaku di IT Del atau sesuai dengan kebijakan saya dengan memberikan nilai 0.
3. Jawaban diketikkan dalam bentuk laporan mengikuti template dan setiap soal harus ditulis secara berurutan.
4. Keterlambatan menyerahkan laporan tidak ditolerir dengan alasan apapun. Oleh karena itu, laporan harus dikumpul tepat waktu.

Referensi

- A. Silberschatz, P.B. Galvin, and G. Gagne, Operating System Concepts, 9th edition, Chapter 9, John Wiley & Sons, Inc., 2013.

Deadlock

1. **[Hal. 306]** Jelaskan mengapa manajemen memori diperlukan?

Jawab:

Karena Manajemen memori merupakan suatu proses yang penting dalam sistem komputer yang mengatur dan mengendalikan penggunaan memori yang efisien, melindungi data dan kode program, membagi memori antara proses, dan memungkinkan penggantian dan pengguliran proses serta virtualiasi memori sehingga sistem dapat berjalan lancar dan efisien.

2. **[Hal. 307]** Jelaskan istilah berikut:

- a. Frame

Jawab:

Frame merupakan suatu unit alokasi memori fisik tetap yang digunakan dalam manajemen memori dengan paging.

- b. Page

Jawab:

Page merupakan suatu unit alokasi memori logis yang menunjukkan bagian dari memori virtual.

- c. Segment

Jawab:

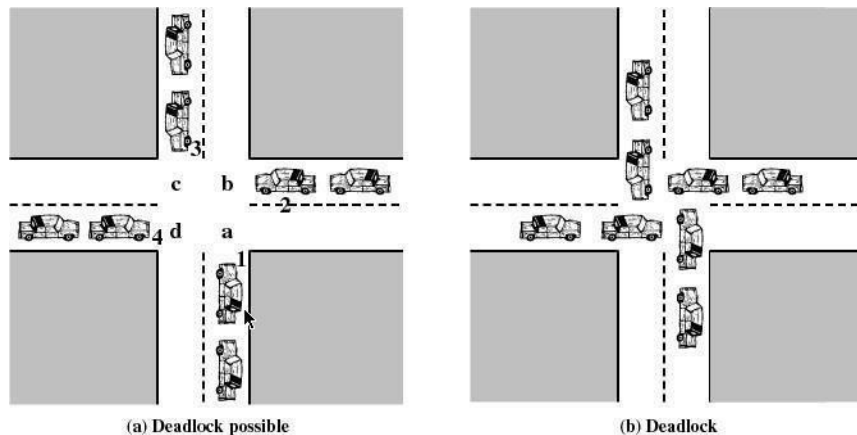
Segment merupakan suatu unit alokasi memori yang logis dalam manajemen memori dengan segmentasi yang memungkinkan pembagian ruang alamat proses yang dapat disesuaikan.

3. **[Hal. 307]** Jelaskan pada situasi bagaimanakah relocate (relokasi) dilakukan?

Jawab:

Relokasi (Relocate) adalah proses memindahkan program atau data dari satu lokasi memori ke lokasi lainnya. Relokasi ini juga sering dibutuhkan dalam beberapa situasi, termasuk saat program atau proses berpindah dari satu bagian memori ke bagian yang lainnya atau ketika sistem operasi melakukan alokasi memori baru untuk program yang baru dijalankan.

4. Dari gambar ilustrasi di bawah, jelaskan mengapa gambar A disebut berpotensi terjadi *deadlock* sedangkan gambar B telah terjadi *deadlock*.



Jawab:

- **Gambar A (Deadlock Possible)**

Pada gambar A, Mobil-mobil berada di posisi di mana mereka dapat melanjutkan perjalanan tanpa menghalangi mobil lain asalkan mereka bergerak secara bergantian. Namun, jika semua mobil mencoba untuk bergerak secara bersamaan, akan terjadi *Deadlock* (buntu).

- **Gambar B (Deadlock)**

Pada gambar B, Mobil-mobil tersebut berada di dalam persimpangan dan berposisi kepala dengan ekor dalam formasi persegi. Setiap mobil menghalangi jalur mobil lain yang ada di depannya dan membuat mobil tidak bisa bergerak ke arah manapun.

5. [Hal. 316] Sebutkan contoh sumber daya yang dapat dikonsumsi oleh proses, dan mengapa *deadlock* dapat terjadi pada saat proses-proses akan mengkonsumsi sumber daya tersebut.

Jawab:

Contoh sumber daya yang dapat dikonsumsi oleh proses:

1. Memori
2. CPU
3. Perangkat I/O

Deadlock dapat terjadi ketika proses-proses saling menunggu untuk mendapatkan sumber daya yang sedang digunakan oleh proses lainnya yang mengakibatkan situasi dimana tidak ada proses yang dapat melanjutkan eksekusi mereka. Hal ini bisa terjadi karna penggunaan sumber daya yang tidak efisien atau penjadwalan buruk pada sistem komputer.

6. [Hal. 316]Jelaskan 3 (tiga) operasi yang harus dilakukan oleh proses ketika mengkonsumsi sumber daya!

- a. Request

Jawab:

- 1) Proses melakukan operasi pengajuan untuk mendapatkan atau memanfaatkan sumber daya yang diperlukan.
- 2) Pengajuan ini dapat terjadi saat proses memerlukan akses ke sumber daya tertentu untuk melanjutkan pelaksanaan tugasnya.

- 3) Sistem akan merespon pengajuan dengan memberikan atau menolak akses ke sumber daya yang diajukan, tergantung pada ketersediaan dan kondisi sumber daya tersebut.

b. Use

Jawab:

- 1) Setelah permintaan diterima dan izin yang diberikan, proses menggunakan atau mengakses sumber daya yang diminta.
- 2) Operasi ini mencakup penggunaan sumber daya tersebut untuk melaksanakan tugas-tugas yang diperlukan oleh proses.
- 3) Proses akan menggunakan sumber daya tersebut sebelum akhirnya dilepaskan.

c. Release

Jawab:

- 1) Setelah selesai menggunakan sumber daya, proses melakukan operasi pelepasan untuk melepaskan sumber daya yang sudah tidak diperlukan lagi.
- 2) Dengan melepaskan sumber daya, sumber daya tersebut akan tersedia kembali untuk digunakan oleh proses lainnya.
- 3) Operasi pelepasan ini penting untuk mencegah pemborosan sumber daya dan memastikan efisiensi dalam penggunaan sumber daya pada sistem.

7. [Hal. 316] Sebutkan contoh *system call* yang dapat diterapkan untuk ketiga operasi yang disebutkan pada No.4

Jawab:

1. **Permintaan (Acquire):** lock()
System call ini bisa digunakan untuk mendapatkan kunci guna memasuki bagian yang kritis.
2. **Penggunaan (Use)**
Tidak ada *system call* yang spesifik untuk penggunaan sumber daya. Penggunaan sumber daya biasanya melibatkan rangkaian *system call* yang sesuai dengan tipe sumber daya tersebut.
3. **Pelepasan (Release):** unlock()
System call ini bisa digunakan untuk melepaskan kunci setelah selesai dari bagian yang kritis.

8. [Hal. 316] Sebuah sistem memiliki sumberdaya sebuah printer dan sebuah DVD drive. Semisalnya, proses P_i menggunakan DVD dan proses P_j menggunakan printer. Jika P_i meminta untuk mendapatkan printer dan P_j meminta untuk menggunakan DVD drive, apakah akan terjadi *deadlock*? Jelaskan mengapa demikian?

Jawab:

Ya, *deadlock* dapat terjadi. Karena Proses P_i telah memegang DVD drive dan sedang menunggu printer yang saat ini dipakai oleh proses P_j . Di sisi lain, proses P_j yang telah memiliki printer, sedang menunggu DVD drive yang saat ini dipakai oleh proses P_i .

9. Jelaskan kondisi yang dapat menyebabkan *deadlock* berikut:

a. *Mutual exclusion*

Jawab:

Kondisi ini terjadi ketika setidaknya satu sumber daya tidak dapat dibagikan atau hanya dapat digunakan oleh satu proses dalam satu waktu.

b. *Hold and wait*

Jawab:

Kondisi ini terjadi ketika suatu proses memegang setidaknya satu sumber daya dan menunggu sumber daya lain digunakan oleh proses lain.

c. *No preemption*

Jawab:


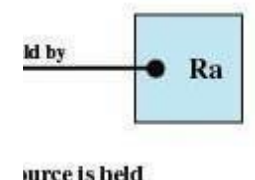
Kondisi ini terjadi ketika suatu sumber daya tidak dapat dipaksa untuk dilepaskan dari proses yang menampungnya.

d. *Circular wait*

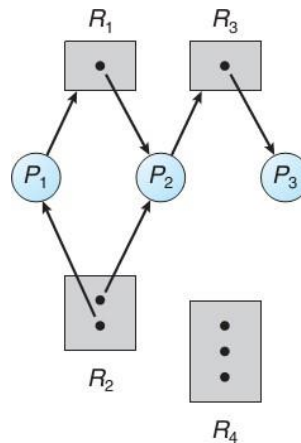
Jawab:

Keadaan ini terjadi ketika terdapat rangkaian proses $\{P_0, P_1, \dots, P_n\}$ yang mana setiap proses P_i menunggu sumber daya yang digunakan oleh proses $P_{(i+1)}$.

10. [Hal. 319-322] Deadlock dapat digambarkan dengan menggunakan *resource-allocation graph*. Pada graf ini proses dan sumberdaya digambarkan sebagai *vertices* V sedangkan direksi atau arah antara proses dan sumberdaya ditunjukkan sebagai *edge* E . Arah atau direksi ketika sebuah proses meminta untuk konsumsi sumberdaya digambarkan pada bagian (a) disebut juga dengan istilah *request edge*, sedangkan arah ketika sebuah sumberdaya sudah dialokasikan kepada proses yang meminta digambarkan pada bagian (b) disebut juga dengan istilah *assignment edge*.

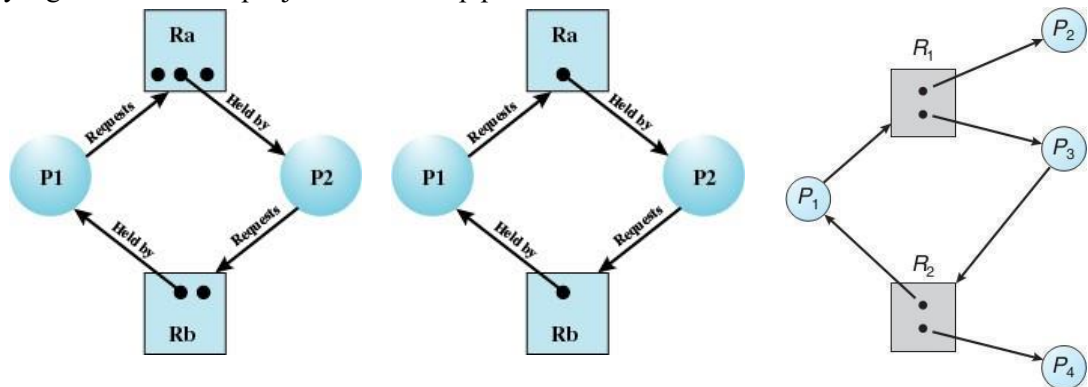
<i>Request Edge</i>	<i>Assignment Edge</i>
$P_1 \longrightarrow R_a$	$P_1 \longleftarrow R_a$
 <p>(a) Resource is requested</p>	 <p>Resource is held</p>

Dari penjelasan diatas jawablah pertanyaan berikut.



Vertices	
• Resources	$\{R_i, \dots \dots \dots\}$
• Proses	$\{P_i, \dots \dots \dots\}$
Edge	$\{P_1 \rightarrow R_1, \dots \dots \dots\}$

Identifikasilah gambar di bawah ini, manakah yang menyebabkan *deadlock* dan mana yang tidak. Berikan penjelasan terhadap pilihan Anda.



Jawab:

Pada gambar satu dan dua, setiap proses saling menunggu sumber daya yang dipegang oleh proses lain. Namun, pada gambar ketiga, deadlock tidak terjadi karena proses P4 dapat melepaskan sumber daya yang sedang digunakan dan memberikan sumber daya tersebut kepada proses P3. Akibatnya, siklus dapat terputus. Oleh karena itu, dapat disimpulkan bahwa sistem tidak akan mengalami deadlock jika graf alokasi sumber daya tidak membentuk siklus. Namun, jika terbentuk siklus, sistem mungkin mengalami deadlock atau tidak tergantung pada apa yang dilakukan untuk menghentikannya.

11. [Hal. 323-327] Jelaskanlah dua skema di bawah yang dapat digunakan untuk menjamin *deadlock* tidak terjadi:
 - a. *Deadlock prevention*

Jawab:

Skema *Deadlock prevention* ini berupaya mencegah kebuntuan dengan menghilangkan setidaknya satu dari empat kondisi Coffman yang terkait dengan *deadlock*, yang meliputi Mutual Exclusion, Hold and Wait, No Preemption, dan Circular Wait.

b. *Deadlock avoidance*

Skema *Deadlock avoidance* ini mengharuskan sistem memiliki informasi tambahan tentang berapa banyak sumber daya yang mungkin dibutuhkan setiap proses. Hal ini memungkinkan sistem untuk memutuskan apakah permintaan sumber daya dapat dipenuhi atau tidak berdasarkan algoritma penghindar kebuntuan seperti Algoritma Bankir.

12. [Hal. 327]Deadlock example

Perhatikan fungsi di bawah yang telah menggunakan lock ordering.

```
void transaction(Account from, Account to, double amount)
{
    mutex lock1, lock2;
    lock1 = get_lock(from);
    lock2 = get_lock(to);

    acquire(lock1);
    acquire(lock2);

    withdraw(from, amount);
    deposit(to, amount);

    release(lock2);
    release(lock1);
}
```

a. Jika terdapat dua thread menjalankan fungsi transaction() diatas secara simultan.

1) Thread-1: transaction(checking_account, savings_account, 25);

2) Thread-2: transaction(savings_account, checking_account, 50);

Apakah akan terjadi deadlock? Buktikan pada bagian mana deadlock terjadi (ikuti alur kode program di atas).

Jawab:

Ya, *deadlock* dapat terjadi.

Pembuktian bagian *deadlock* yang terjadi:

```
// Thread-1
acquire(lock1);
acquire(lock2);
```

```
// Thread-2
acquire(lock1);
acquire(lock2);
```

Thread-1 memulai dan mengunci checking_account (lock1) menggunakan fungsi acquire(lock1); .

Thread-2 memulai dan mengunci savings_account (lock2) menggunakan fungsi acquire(lock2); .

Thread-1 mencoba mengunci savings_account (lock2) menggunakan fungsi acquire(lock2); , tetapi tidak bisa karena lock2 sudah dipegang oleh Thread-2. Akibatnya, Thread-1 menunggu.

Thread-2 mencoba menggunakan fungsi `acquire (lock1)`; untuk mengunci `checking_account (lock1)`, tetapi tidak bisa karena `lock1` sudah dipegang oleh Thread-1. Oleh karena itu, Thread-2 menunggu.

Kondisi deadlock sekarang terjadi karena kedua thread menunggu satu sama lain untuk melepaskan kunci.

- b. Jika deadlock terjadi bagaimana cara mengatasinya? Berikan pembuktian [hint: Hal 337].

Jawab:

Untuk cara mengatasinya yaitu menggunakan deteksi *deadlock*, timeout, atau mekanisme lainnya untuk mengunci akun dalam urutan yang sama.

Berikut pembuktian *deadlock*:

```
// Thread-1
transaction(checking_account, savings_account, 25);

// Thread-2
transaction(savings_account, checking_account, 50);
```

13. [Hal. 328] Jelaskan dengan memberi contoh pada kondisi apakah *safe state* dan *unsafe state* terjadi?

Jawab:

a) Safe State

Safe state merupakan suatu sistem yang berada dalam keadaan aman ketika terdapat periode alokasi sumber daya yang memungkinkan setiap proses memperoleh sumber daya yang diperlukan tanpa menimbulkan *deadlock*.

Contoh:

Misalkan ada tiga proses P1, P2, dan P3, serta tiga sumber daya R1, R2, dan R3. Jika P1 membutuhkan R1 dan R2 untuk menyelesaikan eksekusinya, P2 membutuhkan R2 dan R3, dan P3 membutuhkan R1 dan R3, maka urutan P1 → P3 → P2 adalah urutan yang aman. P1 dapat menyelesaikan eksekusinya dan melepaskan R1 dan R2, kemudian P3 dapat menyelesaikan eksekusinya

b) Unsafe State

Unsafe state merupakan suatu sistem dianggap tidak aman jika tidak ada urutan alokasi sumber daya yang memungkinkan setiap proses memperoleh sumber daya yang dibutuhkannya tanpa menimbulkan *deadlock*. Namun perlu diperhatikan bahwa kondisi tidak aman tidak serta merta berarti sistem akan *deadlock*.

Contoh:

Dalam contoh sebelumnya, jika P1 memiliki R1 dan R2 dan P2 dan P3 menunggu R1 dan R2, sistem berada dalam keadaan tidak aman. Jika P1 tidak melepaskan sumber daya yang dibutuhkan oleh P2 dan P3, sistem akan masuk ke dalam keadaan *deadlock*.

14. [Hal. 329-332] Jelaskan kedua algoritma di bawah yang digunakan pada *deadlock avoidance*:

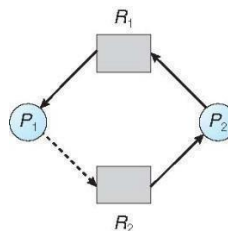
a. *Resource allocation-graph*

Resource allocation-graph (RAG) adalah grafik yang menunjukkan keadaan sistem sehubungan dengan proses dan sumber daya. Proses mana yang memegang sumber daya dan proses mana yang menunggu sumber daya tertentu. RAG dapat digunakan untuk mengetahui apakah sistem dalam keadaan *deadlock*.

b. *Banker's algorithm*

Banker's algorithm adalah suatu algoritma untuk mencegah deadlock dan alokasi sumber daya. Ini melakukan pengujian keamanan dengan mensimulasikan alokasi sumber daya maksimum yang mungkin, lalu melakukan pengecekan s-state untuk menguji aktivitas yang mungkin sebelum memutuskan apakah alokasi harus diteruskan atau tidak.

15. [Hal. 329-332] Jelaskan mengapa gambar di bawah merupakan *unsafe state* pada Resource Allocation-Graph.



Jawab:

Karena kedua proses menunggu sumber daya yang dimiliki oleh proses lain. Tidak ada urutan alokasi sumber daya yang memungkinkan setiap proses memperoleh sumber daya yang dibutuhkannya tanpa menghentikan proses.

16. [Hal. 329-332] Dengan menggunakan Banker's Algorithm, tentukan urutan proses yang akan dieksekusi sehingga tetap mencapai *safe state* (tuliskan caranya dengan mengikuti algoritma *Safety* pada slide halaman 7.29). Carilah nilai dari *available*, *need*, dan *urutan proses*. Spesifikasi dari keempat proses sebagai berikut:

Resource vector:

R_1	R_2	R_3
9	3	6

	Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	1	0	0	3	2	2
P_2	6	1	2	6	1	3			
P_3	2	1	1	3	1	4			
P_4	0	0	2	4	2	2			

	Need		
	R_1	R_2	R_3
P_1
P_2
P_3
P_4

Urutan proses yang mencapai safe state:

<..., ..., ..., ...>

Jawab:

	Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	1	0	0	3	2	2011
P_2	6	1	2	6	1	3			
P_3	2	1	1	3	1	4			
P_4	0	0	2	4	2	2			

- Available R_1 : $1 + 6 + 2 - 9 = 0$
- Available R_2 : $1 + 1 - 3 = -1$
- Available R_3 : $2 + 1 + 2 - 6 = -1$

	Need		
	R_1	R_2	R_3
P_1	2	2	2
P_2	0	0	1
P_3	1	0	3
P_4	4	2	0

Urutan Proses yang mencapai safe state:

< P_2 , P_3 , P_4 , P_1 >

17. [Hal. 329-332] Merujuk pada soal No.13, semisalnya P_1 meminta *resource* dengan detail $R_1, R_2, R_3 = (0, 1, 0)$. Dengan menggunakan Banker's Algorithm, tunjukkan apakah permintaan dari P_1 dapat diberikan atau tidak (tuliskan caranya dengan mengikuti algoritma *Resource-Request* pada slide halaman 7.30). Kemudian tentukan urutan proses yang akan dieksekusi sehingga tetap mencapai *safe state* (tuliskan caranya dengan mengikuti algoritma *Safety* pada slide halaman 7.29). Carilah nilai dari *available*, *need*, *available* dan urutan proses.

	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1
P_2	6	1	2			
P_3	2	1	1			
P_4	0	0	2			

Urutan proses yang mencapai safe state:

<..., ..., ..., ...>

Jawab:

	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	0	3	1	2	1	0	1
P_2	6	1	2	0	0	1			
P_3	2	1	1	1	0	3			
P_4	0	0	2	4	2	0			

Urutan proses yang mencapai safe state:

< P_2, P_3, P_4, P_1 >

18. [Hal. 329-332] Merujuk pada soal No. 4, jika P_4 meminta resource dengan detail $R_1, R_2, R_3 = (4, 3, 5)$ apakah masih mencapai *safe state*? Jika tidak apakah proses lainnya masih dapat dilanjutkan?

Jawab:

Tergantung pada status sistem saat ini, yaitu bagaimana sumber daya dialokasikan dan permintaan sumber daya oleh proses lain. Jika permintaan ini menyebabkan sistem tidak lagi dalam kondisi aman, proses lain mungkin dapat dilanjutkan atau tidak, tetapi jika sistem memilih untuk menghentikan proses, proses tersebut tidak akan dapat dilanjutkan.

19. [Hal. 333-336] Jika sebuah sistem tidak menggunakan algoritma *deadlock-prevention* atau *deadlock avoidance*, kemungkinan akan terjadi *deadlock*. Jika demikian, langkah yang dilakukan adalah?

Jawab:

Langkah yang dilakukan yaitu:

1) Deteksi *Deadlock*

Jalankan algoritma deteksi deadlock secara berkala untuk mengetahui apakah grafik alokasi sumber daya mengandung siklus.

2) Pemulihan dari *Deadlock*

Jika *deadlock* terdeteksi, lakukan salah satu dari Tindakan berikut:

1) Preemption

Ambil kembali sumber daya dari beberapa proses dan berikan kepada proses yang lain.

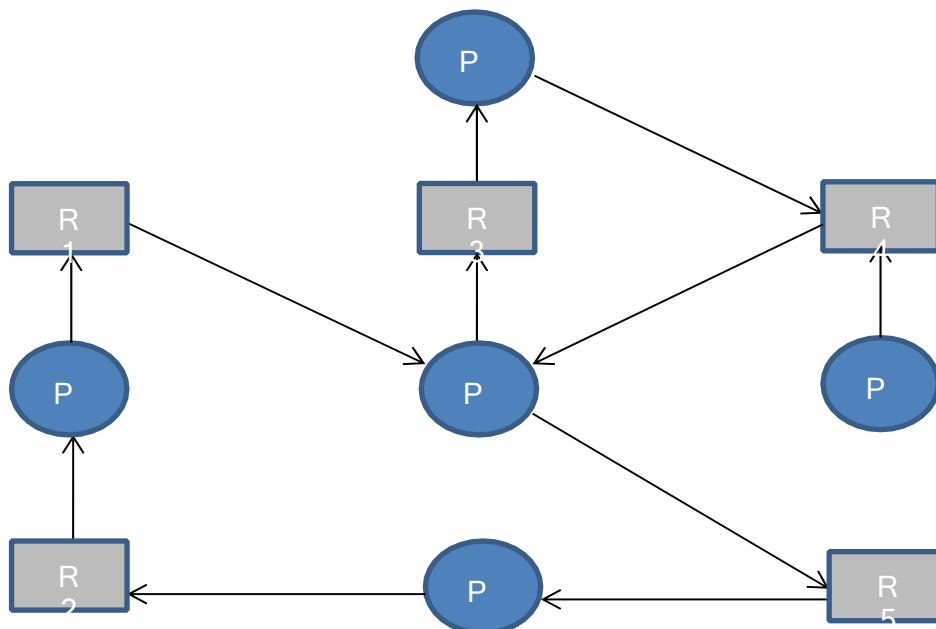
2) Abort Process

Hentikan satu atau lebih proses untuk membebaskan sumber daya.

3) Wait

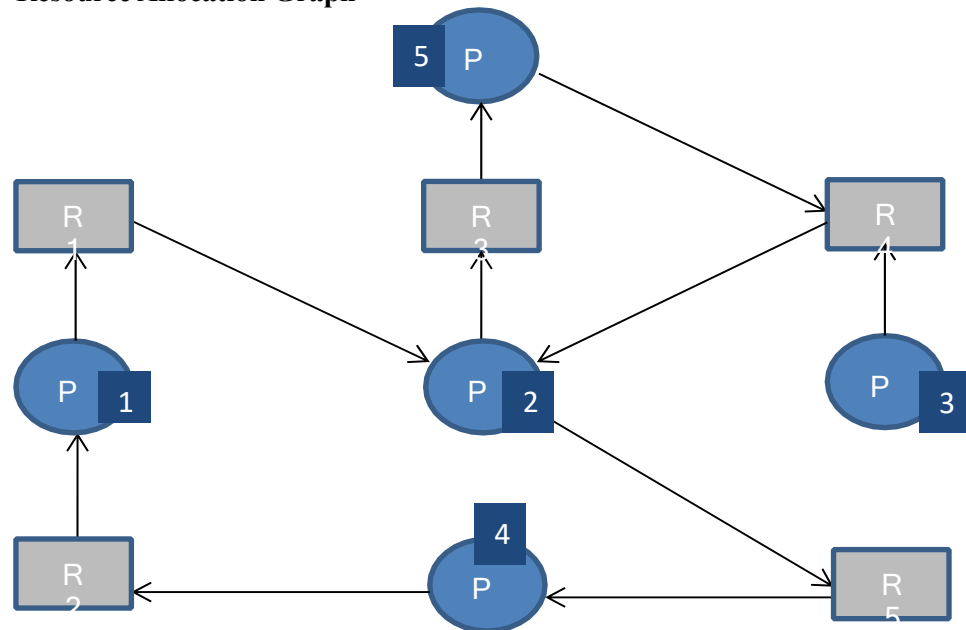
Biarkan proses menunggu sampai kondisi *deadlock* terselesaikan sendiri.

20. [Hal. 333-336] Pada *deadlock detection*, jika setiap sumberdaya (*resource*) hanya memiliki satu *instance*, maka *deadlock* dapat dideteksi dengan menggunakan algoritma graf wait-for. Graf ini diperoleh dari algoritma graf *resource-allocation* dengan menghilangkan *node* sumberdaya dan menggabungkan *edge* yang bersesuaian. Berdasarkan graf *Resource-allocation* pada gambar bagian (a), hasilkanlah graf wait-for.

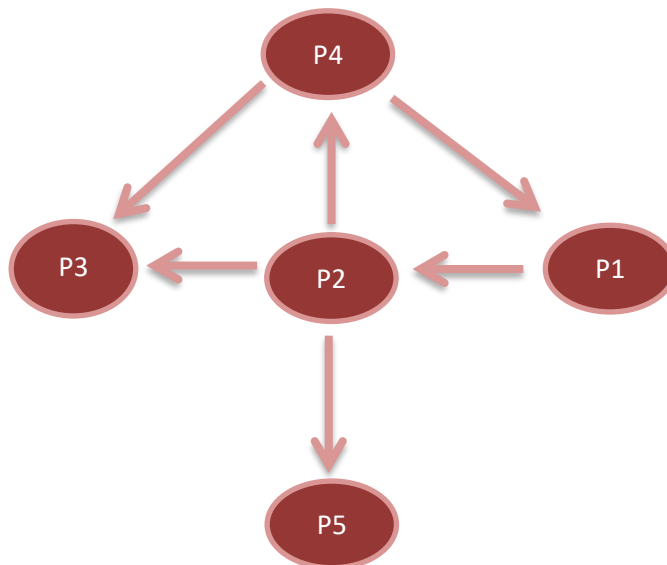


Jawab:

a) Resource Allocation Graph



b) Wait for Graph



21. [Hal. 333-336] Merujuk pada soal No.17, apakah terdeteksi *deadlock*? Jika ya, tunjukkan dan jelaskan pada bagian mana.

Jawab:

Ketika proses atau akses objek, seperti file atau perangkat, tidak dilakukan dengan benar atau secara bersamaan, *deadlock* terjadi. Jika suatu proses mengakses objek tertentu secara bersamaan, proses lain tidak dapat menggunakan objek tersebut sampai objek tersebut dilepaskan. Ini dapat menyebabkan sumber daya objek terkunci, yang membuat proses lain tidak dapat mengaksesnya untuk waktu yang lama. Akibatnya, sumber daya akan kelaparan.

22. [Hal. 333-336] Dengan menggunakan Banker's Algorithm untuk mendeteksi *deadlock* pada slide hal 7.38 dan 7.39. Tentukan urutan proses yang akan dieksekusi sehingga tetap mencapai tidak ada *deadlock*. Spesifikasi dari keempat proses sebagai berikut:

Resource vector:

R_1	R_2	R_3
7	2	6

	Allocation			Request			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	0	0	2	0	2
P_2	0	1	0	0	0	0			
P_3	3	0	3	0	0	0			
P_4	0	0	2	0	0	2			
P_5	2	1	1	1	0	0			

Urutan proses yang tidak deadlock:

<..., ..., ..., ...>

Jawab:

	Allocation			Request			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	0	0	2	0	2	0	0	0
P_2	0	1	0	0	0	0			
P_3	3	0	3	0	0	0			
P_4	0	0	2	0	0	2			
P_5	2	1	1	1	0	0			

Urutan proses yang tidak deadlock:

< P_2 , P_3 >

23. [Hal. 333-336] Merujuk pada soal No. 19, jika P_2 meminta resource dengan detail $R_1, R_2, R_3 = (0, 0, 1)$, apakah teridentifikasi deadlock? Jika, ya apakah proses lainnya masih dapat dilanjutkan?

Jawab:

Tergantung pada status sistem saat ini, yaitu bagaimana sumber daya didistribusikan dan diminta oleh proses lain. Dalam kasus di mana P_2 meminta sumber daya dengan detail $R_1, R_2, R_3 = (0, 0, 1)$, kita perlu mengetahui status sistem saat ini untuk menentukan apakah permintaan ini akan menyebabkan deadlock. Jika terjadi deadlock, strategi pemulihan deadlock yang digunakan menentukan apakah proses lainnya dapat dilanjutkan atau tidak. Namun, proses tidak dapat dilanjutkan jika sistem memilih untuk menghentikannya.

24. [Hal. 336-338] Sebutkan dan jelaskan dua cara untuk memulihkan *deadlock* yang terjadi.

Jawab:

1) Preemption

Preemption adalah suatu Teknik yang memungkinkan sistem operasi mengambil kembali sumber daya yang telah diberikan kepada satu proses dan memberikan sumber daya tersebut kepada proses lain yang membutuhkannya. Proses yang diambil alih akan ditangguhkan sampai sumber daya tersebut dapat diberikan kembali kepadanya.

2) Abort Process

Pada teknik ini, sistem operasi mengakhiri satu atau lebih proses untuk membebaskan sumber daya dan memecahkan *deadlock*. Faktor-faktor seperti prioritas proses, waktu tunggu, jumlah dan jenis sumber daya yang digunakan, dll. dapat menentukan proses mana yang dapat dihentikan. Setelah proses dihentikan dan sumber daya dibebaskan, proses lain dapat melanjutkan eksekusi sampai sumber daya tersebut dibebaskan.

25. [Hal. 223] Dining Philosophers

Terdapat 5 (lima) filosof menghabiskan hidupnya untuk berfikir dan makan. Filosof tersebut membagi meja berlingkar dengan 5 kursi yang dimiliki oleh setiap filosof. Menu mereka adalah spaghetti yang membutuhkan 2 (dua) alat makan yaitu 2 garpu. Ditengah meja terdapat semangkuk spaghetti, 5 piring, dan 5 garpu. Bila filosofi lapar, maka akan mengambil 2 garpu terdekat (sebelah kanan dan kirinya). Filosofi berikutnya tidak dapat mengambil garpu tetangganya yang sedang digunakan, harus menunggu tetangganya selesai menggunakan.

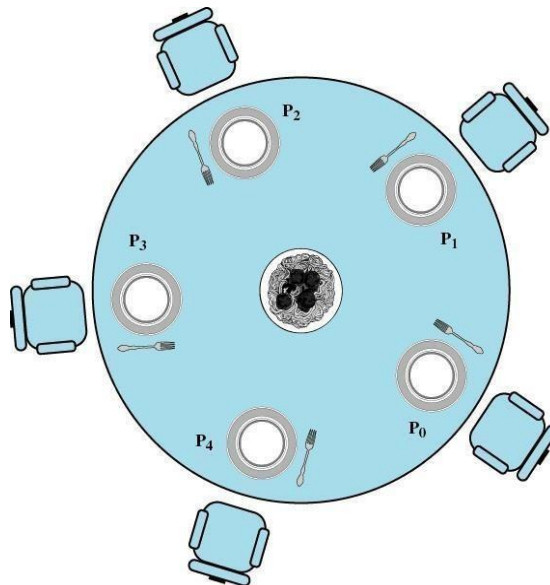


Figure 6.11 Dining Arrangement for Philosophers

- a. Sesuai dengan ilustrasi dining philosophers di atas, jelaskan bagaimana deadlock dan starvation akan terjadi?

Jawab:

Deadlock : Jika semua filosof mengambil garpu di sebelah kiri mereka secara bersamaan, maka semua garpu akan digunakan; tidak ada satu pun filosof yang dapat mengambil garpu di sebelah kanannya karena garpu itu sudah digunakan oleh filosof lain. Akibatnya, tidak ada filosof yang bisa makan.

Starvation : Jika filosof selalu menemukan bahwa tetangganya menggunakan salah satu dari dua garpu yang dia butuhkan, dia harus menunggu selamanya. Jika situasi ini berlanjut, filosof tersebut akan kehabisan makanan karena tidak pernah mendapatkan kesempatan untuk makan.

- b. Jelaskan beberapa cara untuk mengatasi deadlock yang terjadi.

Jawab:

1) Resource Hierarchy Solution

Setiap garpu diberi nomor, dan setiap filosof selalu mengambil garpu dengan nomor lebih rendah terlebih dahulu.

2) Wait-Die and Wound-Wait Schemes

Pada skema ini, jika sebuah proses (filosof) meminta sumber daya (garpu) yang sedang digunakan oleh proses lain, proses lain tersebut dapat memilih untuk menunggu atau "membunuh" sumber daya untuk dilepaskan.

3) Timeouts

Setiap filosof harus menetapkan batas waktu untuk menggunakan garpu. Jika waktu habis dan mereka belum selesai makan, mereka harus mengembalikan kedua garpunya.