Laporan Materi Deadlock

Pengajaran Jarak Jauh (PJJ)

Kelompok X

NIM	Nama	Email	Kontribusi
11319031	Feby Sitinjak		Mengerjakan soal nomor 11-15
11319042	Hotma Aruan		Mengerjakan soal nomor 1-5
11319052	Irma Gracia Siagian		Mengerjakan soal nomor 6-10,21-23
11319053	Dian Sitanggang		Mengerjakan soal nomor 16-20
Tanggal Pengiriman: 11-04-2020			

Institut Teknologi Del

Minggu/Sesi	:	IX/3
Kode Matakuliah	:	10S2203
Nama Matakuliah	:	Sistem Operasi
PanduanKuliah	:	Panduan ini dibuat untuk mengarahkan mahasiswa memahami mengenai Deadlock untuk membantu mereka melaksanakan Pembelajaran Jarak Jauh (PJJ).
Setoran	:	LaporanMateri Deadlockdikirimkandalambentuk PDF denganaturanpenamaan file adalahNo_Kelompok_Laporan_Materi_Deadlock.
Batas Waktu Setoran	:	06April 2020
Tujuan	:	1. Mampu mendefinisikan model sistem deadlock 2. Mampu menjelaskan karakterisasi kondisi-kondisi yang menyebabkan deadlock 3. Mampu menjelaskan penggunaan resource allocation graph untuk pengenalan deadlock 4. Mampu mengelompokkan metode penanganan deadlock ke dalam metode pencegahan (prevention) dan penghindaran (avoidance) 5. Mampu menjabarkan algorithma-algorithma untuk penghindaran deadlock 6. Mampu menjelaskan perbedaan 2 teknik pemulihan deadlock yakni: termination dan preemption. 7. Mampu menjelaskan algorithma-algorithma untuk deteksi deadlock 8. Mampu menjelaskan perbedaan 2 teknik pemulihan deadlock yakni: termination dan preemption

Petunjuk

1. Anda dapat mengerjakan tugas ini secara berkelompok sebanyak 4 orang. Tuliskankontribusidarisetiap orang. Buktidiskusiharusdisertakansepertihasil*capturethread chat/video call***(diisidihalamanterakhirlaporanini).**
2. Anda diperbolehkan untuk diskusi dengan mahasiswa lainnya, asisten dan dosen dengan cara memberikan pertanyaan melalui e-course matakuliah Sistem Operasi. Pada e-course matakuliah Sistem Operasi akan dibuka forum diskusi dengan judul Deadlock.
3. Setiap kelompok harus berkontribusi untuk memberikan pertanyaan di forum dalam bentuk bentuk konfirmasi terhadap jawaban Anda, dan harus menyertakan nomor pertanyaan.Sebelum memberikan pertanyaan silahkan baca buku yang ada pada referensi dan juga slide mengenai Deadlock.
4. 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.
5. Jawaban diketikkan dalam bentuk laporan mengikuti template yang telah disediakan di-ecourse dan setiap soal harus ditulis secara berurutan.
6. Keterlambatan menyerahkan laporan tidak ditolerir dengan alasan apapun. Oleh karena itu, laporan harus dikumpul tepat waktu.

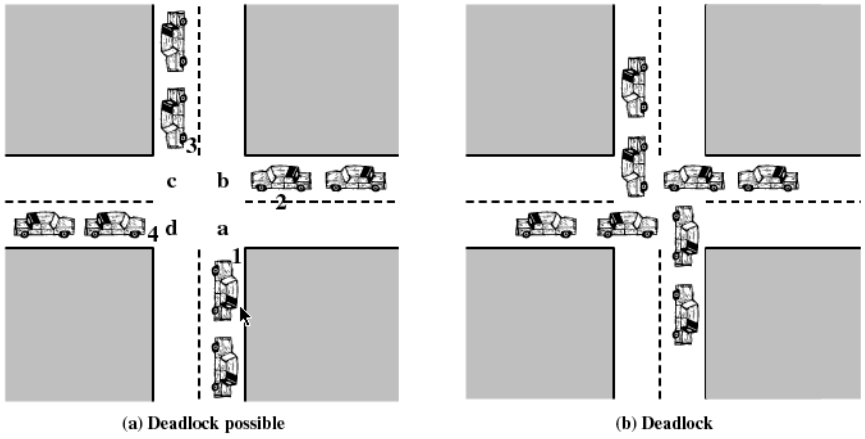
Referensi

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

Deadlock

1. [**Hal.315]** Jelaskan definisi *Deadlock.*
   Jawab:
   Deadlock adalah suatu kondisi dimana dua proses atau lebih saling menunggu proses yang lain untuk melepaskan *resource* yang sedang dipakai, karena adanya proses yang saling menunggu maka tidak akan ada kemajuan dalam proses tersebut. *Deadlock* merupakan masalah yang terjadi ketika banyak proses membagi *resource* yang hanya boleh diubah oleh satu proses saja dalam satu waktu.

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



   Jawab:
   Gambar A berpotensi *deadlock* karena pada gambar tersebut mobil sudah bergerak menuju arah yang bertabrakan dengan mobil lain dalam artian akan terjadi proses yang saling tunggu menunggu dan menimbulkan antrian apabila mobil tersebut berjalan terus dengan arah yang sama.
   Gambar B telah terjadi *Deadlock* karena pada gambar mobil-mobil tersebut sudah saling menunggu (antri) untuk mobil lain mengosongkan jalanan sehingga tidak ada kemajuan dalam mobil – mobil tersebut.

3. **[Hal. 316]**Sebutkan contoh sumberdaya yang dapat dikonsumsi oleh proses, dan mengapa *deadlock* dapat terjadi pada saat proses-proses akan mengkonsumsi sumberdaya tersebut.
   Jawab:
   Contoh sumber daya yang dapat dikonsumsi oleh proses yaitu satu printer dengan satu *drive* DVD apabila proses Pi memegang DVD dan proses Pj memegang printer dan apabila Pi meminta printer dan Pj meminta *drive* maka akan terjadi proses saling tunggu menunggu karena pada proses Pi memegang seluruh proses pada DVD apabila Pj memerlukan *drive* maka proses Pi harus menunggu proses sampai proses Pi dan begitu juga sebaliknya apabila proses Pi memerlukan printer maka dia harus menunggu proses Pj maka akan terjadi proses saling tunggu menunggu (*deadlock*).

4. **[Hal. 316]** Jelaskan 3(tiga) operasi yang harus dilakukan oleh proses ketika mengkonsumsi sumber daya
    a. *Request*
    b. *Use*
    c. *Release*

    Jawab :

    a. *Request* : Proses meminta sumber daya apabila permintaan tidak dapat diberikan dengan secepatnya maka proses tersebut harus menunggu sampai sumber daya yang diminta telah tersedia.
    b. *Use* : Dimana proses dapat menggunakan sumberdaya, contohnya *printer* untuk mencetak, *disk drive* untuk melakukan operasi I/O.
    c. *Release* :Proses melepaskan sumber daya setelah proses menyelesaikan penggunaan sumber daya maka sumber daya harus dilepaskan sehingga dapat digunakan oleh proses lain.

5. **[Hal. 316]** Sebutkan contoh *system call* yang dapat diterapkan untuk ketiga operasi yang disebutkan pada No.4.
    Jawab:
    System call yang dapat diterapkan untuk ketiga operasi pada soal no 4 yaitu :
    a. *Request : wait ( ), signal ( )*
    b. *Use :*
    c. *Release : free ( ), wait ( ), signal ( )*

6. **[Hal. 316]**Sebuah sistem memiliki sumber daya 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 danP_j meminta untuk menggunakan DVD drive, apakah akan terjadi *deadlock*? Jelaskanmengapademikian?

    Jawab:

    Jika kondisi yang diberikan adalah P_i sumber daya yang menggunakan DVD sementara P_i meminta untuk menggunakan printer tanpa melepaskan DVD maka akan terjadi proses wait. Begitu juga sebaliknya jika proses P_j menggunakan printer dan meminta untuk meggunakan DVD drive tanpa melepaskan printer maka akan terjadi proses wait. Jika proses wait terjadi terus menerus maka akan terjadi *deadlock.*

7. **[Hal. 319]** Jelaskan kondisi yang dapat menyebabkan *deadlock* berikut:
    a. *Mutualexclusion*
    b. *Hold and wait*
    c. *No preemption*
    d. *Circular Wait*

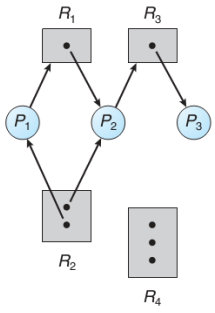
    Jawab:

a. ***Mutualexclusion*** adalah kondisi atau situasi dimana setidaknya 1 resource harus disimpan artinya 1 proses hanya dapat menggunakan 1 resource. Jika proses lain meminta sumber daya tersebut, maka proses permntaan tersebut harus ditunda sampai sumberdaya telah dirlis kembali.
b. ***Hold and wait*** adalah keadaan dimana sebuah proses harus ditahan pada setidaknya 1 resource atau sumber daya dan melakukan proses *wait* atau tunggu untuk memperoleh sumber daya lain yang saat ini sedang ditahan oleh proses lain yang menggunakan resource tersebut.
c. ***No preemption*** adalah kondisi dimana sumberdaya atau resource tidak dapat di dahului, artinya sumber daya hanya dapat dilepaskan secara sukarela oleh proses yang menahannya setelah tugas dari proses tersebut sudah selesai.
d. ***Circular wait*** kondisi dimana satu set (P0,P1,….,Pn) dari proses menunggu harus ada, sehingga proses P0 sedang menunggu sumber daya yang dimiliki oleh P1, P1 mengunggu proses yang dimiliki oleh P2, … , Pn-1 sedang menuggu sumber daya untuk dipasok oleh Pn, dan Pn menunggu sumber daya yang dimiliki oleh P0.

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

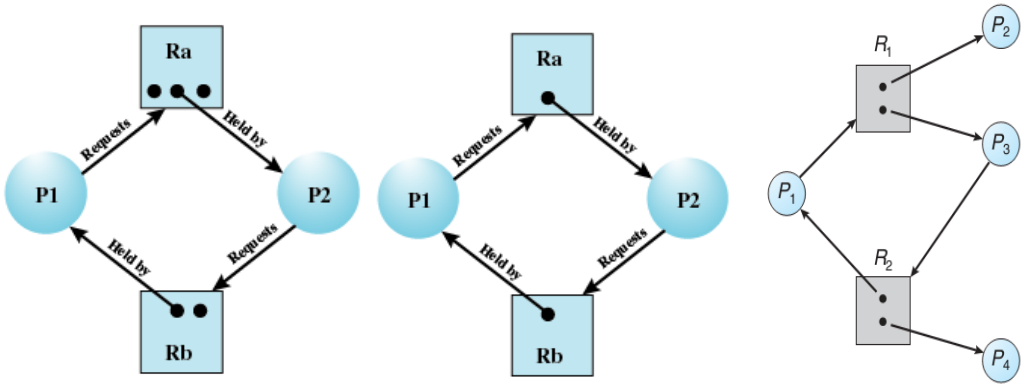
Request Edge	*Assignment Edge*
$P_1 R_a$ →	$P_1 R_a$ ←
	
(a) Resouce is requested	(b) Resource is held

Dari penjelasan diatas jawablah pertanyaan berikut:



Vertices	
• Resources	$\{R_1, \boldsymbol{R_2}, \boldsymbol{R_3}, \boldsymbol{R_4}\}$
• Proses	$\{P_1, \boldsymbol{P_2}, \boldsymbol{P_3}\}$
Edge	$\{P_1 \rightarrow R_1, \boldsymbol{P_2} \rightarrow \boldsymbol{R_3}, \boldsymbol{R_1} \rightarrow \boldsymbol{P_2}, \boldsymbol{R_2} \rightarrow \boldsymbol{P_2}, \boldsymbol{R_2} \rightarrow \boldsymbol{P_1}, \boldsymbol{R_3} \rightarrow \boldsymbol{P_3}\}$

Identifikasilah gambar dibawah ini, manakah yang menyebabkan *deadlock* dan mana yang tidak. Berikan penjelasan.



Jawab:

- Pada proses yang pertama kemungkinan akan terjadi *deadlock,* karena proses ini berbentuk siklus atau *cycle* namun memiliki multi instances. Artinya karena *resources* Ra dan Rb memiliki multi instances maka proses P1 dapat meminta sumberdaya Rb dan melepaskan sumberdaya Ra begitu juga dengan P2 dapat meminta sumberdaya dari Ra dan melepaskan Rb sehingga tidak terjadi *deadlock.*
- Pada proses yang kedua dapat terjadi *deadlock* karena proses ini berbentuk siklus atau *cycle* dan memiliki single instances. Artinya *resources* Ra dan Rb hanya memiliki single instances dimana proses tidak dapat melepaskan sumberdaya yang di pegang sehingga tidak ada jalan maka terjadilah *deadlock* atau kebuntuan.
- Proses yang ketiga adalah proses yang tidak menimbulkan *deadlock* karena tidak berbentuk siklus atau *cycle.* Jika diamati bahwa proses P4 dapat melepaskan R2, dan R2 dapat dialokasikan ke P3.

9.**[Hal. 323-327]** Jelaskanlah dua skema dibawah yang dapat digunakan untuk menjamin *deadlock* tidak terjadi :

a. *Deadlock prevention*
b. *Deadlock avoidance*

Jawab:

a. Deadlock prevention

Deadlock dapat dicegah agar tidak terjadi dengan menggunakan metode *deadlock prevention* yaitu membuat batasan permintaan dengan cara:

- *Mutual exclution* harus ditahan, yang berarti setidaknya ada 1 *resource* yang tidak dapat dibagikan. *Resource* yang dapat dibagi tidak memerlukan akses yang ekslusif sehingga tidak terjadi *deadlock* atau kebuntuan.
- Untuk memastikan agar tidak terjadi kondisi *hold-and-wait* dalam suatu sistem maka kita harus memastikan bahwa setiap kali proses meminta satu sumber daya, proses tersebut tidak memiliki sumber daya yang sedang ditahan. Jadi jika suatu proses ingin memnita sumber daya lain maka sumber daya yang sedang dimiliki atau sedang ditahan harus dilepas.

- Untuk mencegah agar *deadlock* tidak terjadi maka kondisi non-preemptive harus dihilangkan dengan protocol berikut: jika suatu proses menahan beberapa sumber daya dan meminta sumber daya lain yang tidak dapat dialokasikan (yaitu proses harus menunggu), maka semua sumber daya yang sedang dipegang oleh proses saat ini harus ditolak. Sehingga, sumber – sumber lain dapat dilepaskan dengan mudah sehingga tidak terjadi *deadlock.*

- Untuk menghindari terjadinya *circular wait* adalah dengan melakukan pemesanan total semua jenis sumber daya dengan paksa dan untuk meminta sumber daya dalam urutan peningkatan jumlah enumerasi.

b. *Deadlock avoidance*
   Menghindari *deadlock* dengan cara hanya memberi akses kepada permintaan sumber daya yang memiliki kemungkinan tidak menimbukan *deadlock*.
   - Jika pemberian akses sumber daya memiliki kemungkinan tidak mengalami *deadlock,* maka sumber daya diberikan ke proses yang meminta.
   - Jika sumber daya memungkinkan timbulnya *deadlock,* maka proses yang meminta disuspend sampai waktu permintaannya aman diberikan.

10. **[Hal. 327]** Deadlock example.
    Perhatikan fungsi dibawah 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.
   - Thread-1 : transaction (checking_acount, savings_account, 25);
   - Thread-2 : transaction (savings_account, checking_account,50);

   Apakah akan terjadi *deadlock*? Buktikan pada bagian mana *deadlock* terjadi

b. Jika *deadlock* terjadi bagaimana cara mengatasinya? **[hint : Hal 337]**
   Jawab:

   a. *Deadlock* kemungkinan dapat terjadi jika dua thread tersebut menjalankan fungsi transaksi () secara bersamaan, namun mentransposisi akun yang berbeda. Artinya satu proses meminta thread Thread-1 : transaction (checking_acount, savings_account, 25) dan proses lain juga dapat meminta thread Thread-2 : transaction (savings_account, checking_account,50).
   b. Untuk mengatasinya dilakukan recovery yang akan menghentikan siklus wait.

11. Jelaskan dengan memberi contoh pada kondisi apakah *safe state* dan *unsafe state* terjadi?
    Jawab:
    Safe state adalah banyaknya sumber daya yang sudah dialokasikan terhadap masing-masing proses.Unsafe state adalah banyaknya sumber daya yang belum dialokasikan kepada masing – masing proses, sehingga masih ada sumbe rdaya yang digunakan oleh beberapa proses. Kondisi safe state dan unsafe state terjadi untuk menghindari deadlock.
    Contoh:

Proses	R Alokasi	Max
A	2+4	10
B	1	3
C	3	7
Jumlah	10	

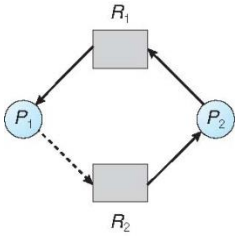
    Sisa 10-10=0(tidak ada sisa sumber daya yang dapat digunakan)
    Dari pengalokasian diatas, ternyata mendapatkan hasil unsafe, jadi artinya terjadi deadlock apabila proses A dialokasikan pertama. Dimana proses A tidak pernah selesai untuk diproses dan proses B&C menunggu terus menerus. Maka dengan itu, kita akan mencari proses lain, yang akan di proses lebih awal sehingga tidak terjadi deadlock.

12. Jelaskan kedua algoritma di bawah yang digunakan pada *deadlock avoidance*:
    a. *Resourceallocation-graph*
    b. *Banker'salgorithm*

Jawab:

a. *Resource allocation-graph* untuk menghindari deadlock pada sistem yang hanya mempunyai satu instance saja pada tiap resource.
b. *Banker's algorithm* untuk menghindari deadlock pada sistem yang mempunyai banyak instance untuk tiap resource.

13. Jelaskan mengapa gambar di bawah merupakan *unsafe state* pada Resource *Allocation-Graph.*



Jawab:
Karena jika kita ibaratkan bahwa jika sistem mengalokasikan R2 pada P2 maka akan memungkinkan terjadinya cycle yang menyebabkan deadlock. Sehingga sistem tidak dapat mengalokasikan R2 pada P2, bila sistem memaksa mengalokasikan maka akan terjadi keadaan unsafe state.

14. 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:

Tabel available

Available		
R1	R2	R3
9	3	6

Tabel Need

	Need		
	R1	**R2**	**R3**
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

Urutan proses yang mencapai safe state:

<P2,P3,P4,P1>

15. 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 danurutan 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						

Jawab :

	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	1	1	1	2	2	2	9	2	6
P_2	6	1	2	0	0	2			
P_3	2	1	1	1	0	3			
P_4	0	0	2	4	2	0			

Urutan proses yang mencapai safe state:

< P2, P3, P4, P1>

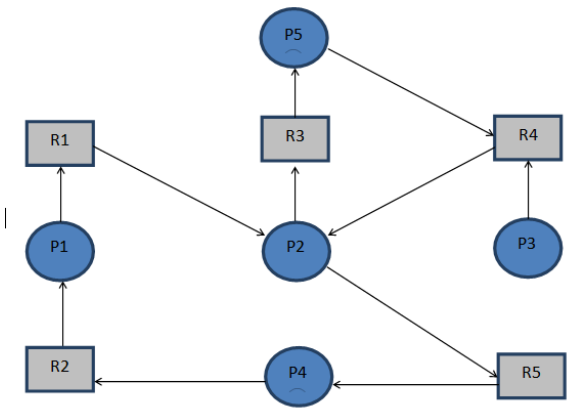
16. **[Hal. 329-332]**Merujukpadasoal No. 4, jikaP_4 meminta resource dengandetail R_1, R_2, R_3 = (4, 3, 5), apakah masih mencapai *safestate*? Jika, tidakapakah proses lainnyamasihdapatdilanjutkan?Jika p4 meminta resource menjadi (4, 3, 5) proses tersebuttidakmencapai safe state karenatidakmemenuhisyaratdimana request <= need dansemua proses tidakdapatdijalankankarnasyarattersebutharusterpenuhidulubarudapatmenjalankanp1, p2, p3, p4.

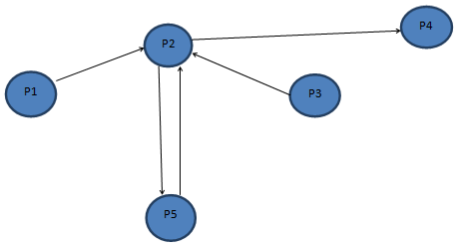
17. **[Hal. 333-336]**Jika sebuah sistem tidak menggunakan algoritma *deadlock-prevention* atau *deadlockavoidance*, kemungkinan akan terjadi *deadlock*. Jika demikian, langkah yang dilakukan adalah?

    Jawab:

    Maka deadlock terjadi karena mutual exclusion, non-preemtion, hold and wait, dan sirkuler. Pada masalah dalam proses ini dapat diatasi dengan cara pencegahan, ostrich, dan menghindari. Jika sistem yang digunakan tidak menyediakan algorithm tersebut maka deadlock harus dideteksi dahulu dan diperbaiki dengan menggunakan mekanisme detection algorithm dan cara rollback dan restart sistem ke safe state.

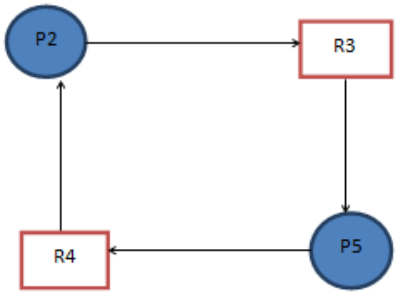
18. **[Hal.333-336]**Pada *deadlockdetection*, jika setiap sumber daya (*resource*) hanya memiliki satu *instance*, maka *deadlock* dapat dideteksi dengan menggunakan algoritm agrar wait-for. Graf ini diperoleh dari algoritma graf *resource-allocation* dengan menghilangkan *node* sumber daya dan menggabungkan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





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

    Ya , deadlock terjadi dimana antara P2 dan P5 yaitu P2 akan melepaskan sumber daya yang dibutuhkan oleh P5 dan P2 sedang dalam keadaan waiting. Dan P5 juga akan melepaskan sumberdaya yang dibutuhkan oleh P2 sedangkan P2 masih dalam keadaan menunggu dan P5 juga akan menunggu jadi pada bagian tersebut terjadi deadlock.



20. **[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:
    Jawab:

    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:

<…, …., …., ….>

21. **[Hal. 333-336]** Merujuk pada soal No. 19, jikaP_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:

Jika P2 meminta *resource* dengan detail R1,R2,R3 (0,0,1) maka akan terjadi deadlock dan proses lainnya tidak dapat dilanjut karena meskipun sumber daya yang dimiliki oleh proses P0 dapat diklaim, namun jumlah sumber daya yang tersedia tidak cukup untuk memenuhi *request* dari prosess. Sehingga terjadiah *deadlock* yang terdiri dari proses P1,

P2,P3,P4

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

    Jawab:

    a. Terminasi Proses

       Metode ini akan mematahkan atau memutusakan *deadlock cycle,* tetapi bisa saja proses-proses yang *deadlock* telah dikomputasi dalam waktu yang lama dan hasil – hasil komputasi parsial harus dibuang, sehingga ada kemungkinan harus dikomputasi ulang. Abort suatu proses pada satu waktu sampai tereliminir. Metode ini memungkinkan adanya *overhead.* Proses-proses yang harus di*abort* adalah proses yang terminasinya minimum cost. Faktor – faktor yang yang mempengaruhi pemilihan proses adalah :

       - Ada prioritas dari proses
       - Berapa lama proses telah mengkomputasi dan berapa lama lagi akan mengkomputasi sebelum menyelesaikan tugasnya.
       - Berapa banyak dan apa tipe sumberdaya yang digunakan oleh proses
       - Berapa sumber daya lagi yang dibutuhkan proses supaya selesai
       - Berapa banyak proses yang perlu diterminasi

    b. Prempt Sumber Daya

       Mengeliminasi *deadlock* menggunakan *preempt* sumber daya , artinya sumber daya dari suatu proses harus di *preempt* secara berturut dan memberikan sumber daya ke proses lain sampai *deadlock cycle* patah. Berikut beberapa langkah – langkahnya:

- Memlilih korban
  Seperti pada terminasi proses, kita harus menentukan sumber daya dan proses mana yang akan di preempt dengan minimum cost.

- *Rollback*
  Jika kita *preempt* sebuah sumber daya dari sebuah proses, proses tidak dapat berlanjut dengan eksekusi normal karena proses kehilangan beberapa sumber daya yang diperlukan. Kita harus *rollback* proses ke beberapa *safe state* dan *restart* dari *state* tersebut. Agar mudah untuk menentukan *safe state* adalah dengan total rollback, abort proses, dan restart.

- Starvation
  Dalam sebuah sistem dimana pemilihan korban berdasar primer pada faktor *cost,* dapat terjadi suatu proses tidak pernah menyelesaikan tugasnya karena suatu sumber daya selalu di*preempt* pada proses yang sama. Kita harus memastikan bahwa sebuah proses dapat dipilih sebagai korban hanya dengan batasan waktu tertentu. Solusi pada umumnya adalah dengan menambahkan jumlah *rollback* ke dalam factor *cost.*

23. **[Hal. 223]** Dining Philosophers
    Terdapat 5 (lima) filosof menghabiskan hidupnya untuk berfikir dan makan. Filosof tersebut membagi meja berlingkar dengan 5 kurisi yang dimiliki oleh setiap filosof. Menu mereka adalah spaghetti yang membutuhkan 2 (dua) alat makan yaitu 2 garpu. Ditengah meja terdapat semangkuk spaghettig, 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.

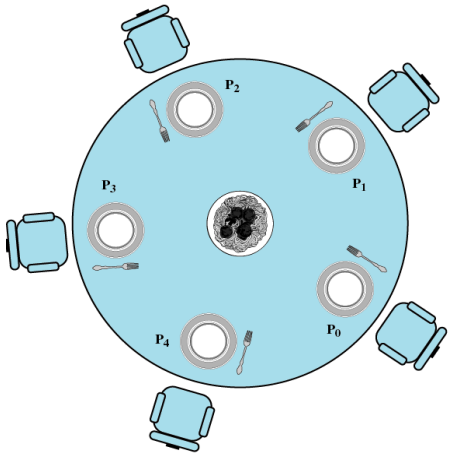


Figure 6.11 Dining Arrangement for Philosophers

a. Sesuai dengan ilustrasi *dining philosophers* diatas, jelaskan bagaimana *deadlock* dan *starvation* akan terjadi?
b. Jelaskan beberapa cara untuk mengatasi *deadlock* yang terjadi.

Jawab:

a. *Deadlock* akan terjadi ketika para filsof ingin makan dengan menggunakan 2 garpu sedangkan yang tersedia adalah 5 maka akan ada 2 filsof yang dapat makan dan 3 yang tidak. Filsof yang mendapatkan 2 garpu akan makan (dalam arti bahwa filsof itu akan menahan garpu tanpa me*release*nya) sementara filsof lain akan menunggu sampe filsof yang dapat makan me*release* garpu atau *resources.*
b. Beberapa cara untuk mengatasi *deadlock* yang terjad adalah :
    - Filsof yang duduk harus berjumlah paling banyak =4
    - Filosof dapat mengambil garpu sebanyak 2 jika diperbolehkan (pengambilan dilakukan dengan *critical section*)
    - Dengan menggunakan solusi asymmetric yaitu filosof yang bernomor ganjil mengambil garpu kiri terlebih dahulu kemudian garpu kanannya, sedangkan filosof genap mengambil garpu kanan terlebih dahulu kemudian garpu bagian kirinya.

Bukti pelaksanaan diskusi melalui chat/video call