PEMAKAIAN GRAF UNTUK PENDETEKSIAN DAN PENCEGAHAN *DEADLOCK* PADA SISTEM OPERASI

Mira Muliati – NIM : 13505110

Program Studi Teknik Informatika
Sekolah Teknik Elektro Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if 15110@students.if.itb.ac.id

Abstrak

Teori graf merupakan topik yang banyak mendapat perhatian, karena model-modelnya sangat berguna untuk aplikasi yang luas, seperti masalah dalam jaringan komunikasi, transportasi, ilmu komputer, dan lain sebagainya. Salah satu aplikasi graf pada sistem operasi adalah graf alokasi sumber daya yang digunakan untuk melakukan pendeteksian dan pencegahan *deadlock*. *Deadlock* adalah suatu kondisi dimana sistem tidak berjalan lagi karena adanya proses yang saling menunggu.
Graf alokasi sumber daya memiliki dua jenis simpul, yaitu simpul sumber daya dan simpul proses. Sisi graf alokasi sumber daya juga dibedakan menjadi dua, yaitu sisi permintaan dan sisi alokasi. Sisi permintaan merepresentasikan daya yang diminta oleh suatu proses terhadap sumber daya tertentu, sedangkan sisi alokasi menunjukkan alokasi daya yang dilakukan oleh suatu sumber daya terhadap proses tertentu.
Setelah suatu sistem direpresentasikan dalam graf alokasi sumber daya, dapat dilakukan pendeteksian *deadlock* dengan memperhatikan *cycle* pada graf tersebut. Graf yang tidak memiliki *cycle* dipastikan tidak akan mengalami *deadlock*. Sebaliknya, graf yang mengandung *cycle* berpotensi mengalami *deadlock*. Suatu graf alokasi sumber daya akan mengalami *deadlock* jika graf tersebut mengandung *cycle* dan memiliki simpul sumber daya dengan satu instant.

Pencegahan *deadlock* dapat dilakukan dengan menambahkan *claimed edge* pada graf. *Claimed edge* Pi → Rj berarti bahwa proses Pi akan meminta sumber daya Rj pada suatu waktu. Ketika proses Pi memerlukan sumber daya Rj, *claimed edge* diubah menjadi sisi permintaan. Dan setelah proses Pi selesai menggunakan Rj, sisi alokasi diubah kembali menjadi *claimed edge.*

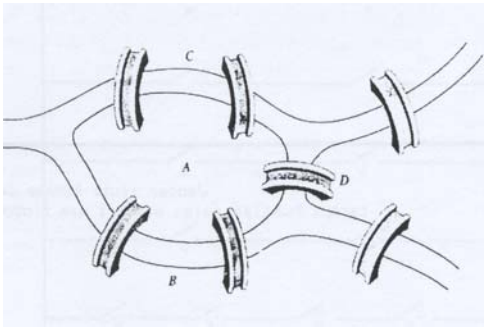
Kata kunci: graf alokasi sumber daya, cycle,deadlock.

1. Pendahuluan

1.1 Graf

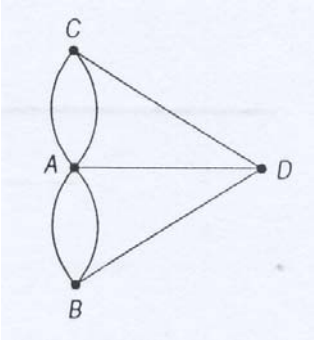
1.1.1 Sejarah Graf

Graf dipakai pertama kali oleh seorang matematikawan Swis pada tahun 1763 untuk memecahkan teka-teki jembatan Konigsberg. Di kota Konigsberg –Jerman Timur– terdapat sungai Pregal yang dibelah dua oleh Pulau Kneipof. Daratan yang dipisahkan oleh sungai tersebut dihubungkan oleh tujuh buah jembatan. Teka-tekinya adalah : apakah mungkin melalui ketujuh jembatan tersebut dan kembali ke tempat semula dengan masing-masing jembatan dilalui tepat satu kali ?



Sebelum Euler memodelkan masalah ini ke dalam graf dan mengemukakan solusinya, kebanyakan orang sepakat bahwa tidak mungkin kembali ke tempat semula namun mereka tidak mampu menjelaskan mengapa. Euler memodelkan daratan dengan titik yang disebut sebagai simpul dan jembatan yang

menghubungkannya sebagai garis yang disebut sebagai sisi.



Jawaban Euler adalah : orang tidak mungkin melalui ketujuh jembatan tersebut masing-masing satu kali dan kembali lagi ke tempat semula jika jumlah sisi dari masing-masing simpul tidak seluruhnya genap. Atau dengan kata lain, jika masing-masing simpul memiliki jumlah sisi genap maka dengan melalui masing-masing sisi satu kali kita dapat kembali ke tempat semula. Dari gambar di atas tampak bahwa simpul-simpul dari graf pemodelan jembatan Konigsberg memiliki sisi berjumlah ganjil, jadi orang tidak mungkin kembali ke tempat semula.

1.1.2 Definisi Graf

Graf adalah struktur diskrit yang terdiri dari simpul dan sisi dimana sisi-sisi menghubungkan simpul yang ada [1] . Graf juga didefinisikan sebagai pasangan himpunan (V,E) yang dalam hal ini :

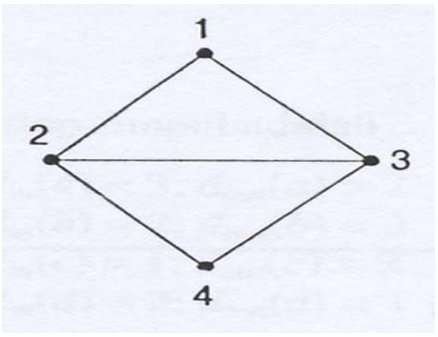
V = himpunan tidak kosong dari simpul-simpul (*vertex* atau *node*) = $\{v_1,v_2,...v_n\}$ dan

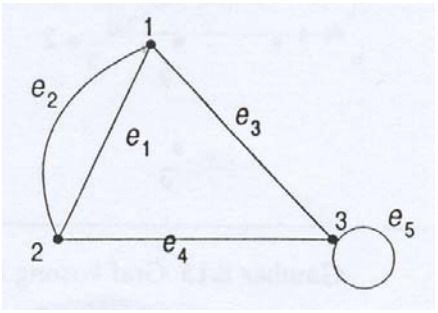
E = himpunan sisi-sisi (*edges* atau *arcs*) yang menguhubungkan sepasang simpul = $\{e_1,e_2,...e_n\}$

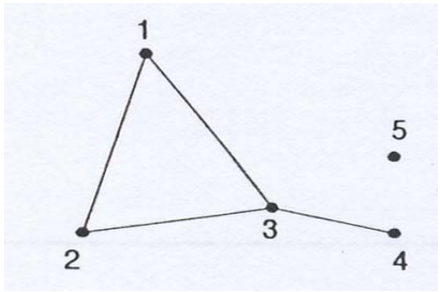
Atau dapat ditulis singkat notasi G = (V,E) [2] .

Simpul pada graf dinomori dengan huruf atau bilangan asli atau gabungan keduanya. Sedangkan sisi yang menghubungkan simpul vi dengan simpul vj dinyatakan dengan pasangan (vi,vj) [2] atau dengan lambang e_i dimana i = himpunan bilangan asli = {1,2,3,...} .

Secara geometri graf direpresentasikan sebagai himpunan titik yang dihubungkan dengan himpunan garis dalam ruang dua dimensi.



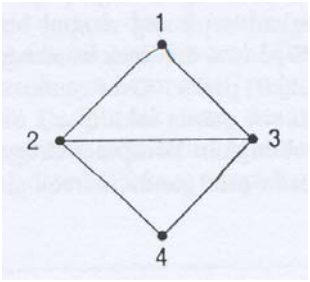




1.1.3 Jenis-jenis Graf

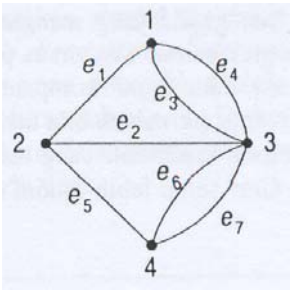
Berdasarkan jenis sisinya graf digolongkan menjadi dua jenis :

1. Graf sederhana yaitu graf yang tidak memiliki sisi ganda maupun sisi loop.

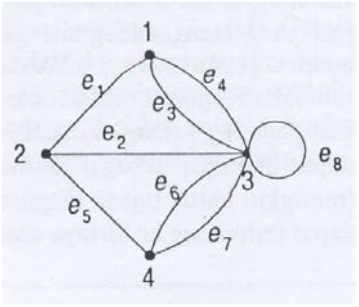


2. Graf tidak sederhana yaitu graf yang memiliki sisi ganda atau sisi loop. Graf tidak sederhana dibedakan menjadi dua :

   a) Graf ganda yaitu graf yang memiliki sisi ganda. Sisi ganda adalah sekumpulan sisi yang menghubungkan sepasang simpul yang sama. Dengan kata lain, bila sepasang simpul dihubungkan oleh lebih dari satu sisi, maka sisi-sisi itu disebut sisi ganda.

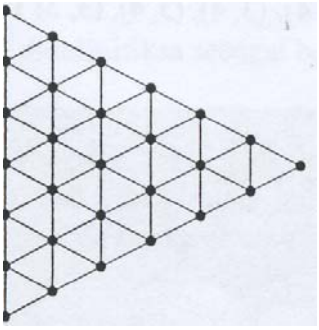


   b) Graf semu yaitu graf yang memiliki sisi loop. Loop adalah sisi yang menghubungkan sebuah simpul dengan dirinya sendiri.



Berdasarkan jumlah simpul yang dimilikinya, graf juga digolongkan menjadi dua jenis :

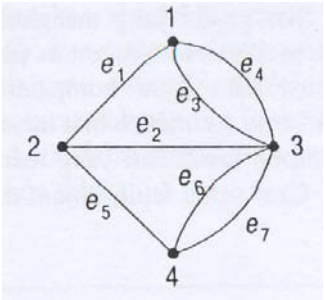
1. Graf berhingga (*limited graph*)
   Graph berhingga adalah graf yang jumlah simpulnya berhingga.



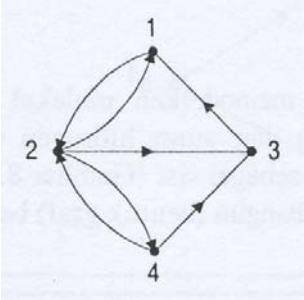
2. Graf tak berhingga (*unlimited graph*)
   Graf tak berhingga adalah graf yang jumlah simpulnya tak berhingga. Secara geometris graf tak berhingga digambarkan dengan sisi-sisi yang hanya memiliki satu simpul untuk setiap simpul luarnya. Sekilas nampak seperti graf yang belum selesai digambar.

Penggolongan lain dilakukan berdasarkan orientasi arah dari sisi-sisi graf. Ada dua jenis graf dalam penggolongan ini :

1. Graf tak berarah, yaitu graf yang sisi-sisinya tidak memiliki arah.



2. Graf Berarah, yaitu graf yang sisi-sisinya memiliki orientasi arah.

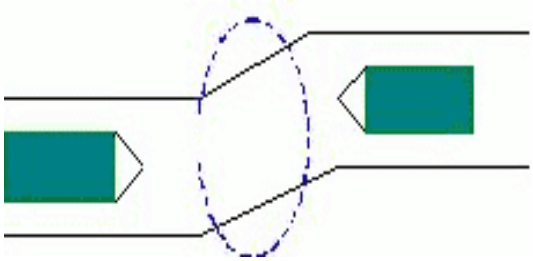


Pada graf jenis ini, sisi yang menghubungkan simpul 3 ke simpul 1 tidak sama dengan sisi yang menghubungkan simpul 1 ke simpul 3 karena orientasi arahnya berbeda.

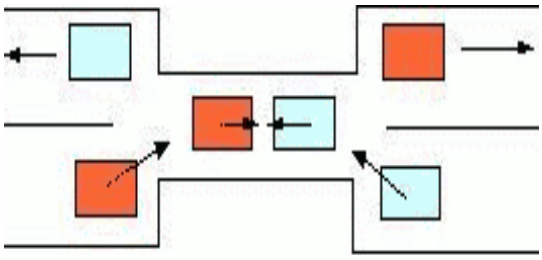
1.2 *Deadlock*

1.2.1 Definisi *Deadlock*

Menurut arti katanya *deadlock*adalah kebuntuan. Dalam sistem operasi kebuntuan yang dimaksud adalah kebuntuan proses sehingga *deadlock* digunakan untuk penyebutan terhadap suatu kondisi permanen dimana proses tidak dapat berjalan ataupun tidak ada komunikasi lagi antar proses.
Contoh *deadlock*pada rel kereta api :



Kedua kereta tidak dapat berjalan karena keduanya saling menunggu kereta yang lain agar bisa lewat sehingga terjadi *deadlock*.
Contoh lain ditunjukkan oleh gambar berikut :



Gambar tersebut menunjukkan *deadlock* yang terjadi pada sebuah jembatan. Kedua mobil dari sisi bawah gambar tidak dapat melaju karena terjadi *deadlock* di tengah jembatan. *Deadlock* tersebut hanya dapat diatasi bila salah satu atau bahkan beberapa mobil mundur.

1.2.2 Penyebab *Deadlock*

Deadlock disebabkan karena suatu proses menunggu sumber daya yang sedang digunakan oleh proses lain yang juga sedang menunggu sumber daya yang digunakan oleh proses sebelumnya. Atau dengan kata lain setiap proses dalam set saling menunggu untuk sumber daya.
Ada empat kondisi yang dapat mengakibatkan *deadlock*, yaitu :

1. Mutual Eksklusif

   Mutual eksklusif adalah suatu kondisi dimana hanya ada satu proses yang boleh memakai sumber daya. Proses lain yang ingin memakai sumber daya tersebut harus menunggu hingga sumber daya tadi dilepaskan atau tidak ada proses yang memakai sumber daya tersebut.

2. *Hold and wait*

   Proses yang sedang memakai sumber daya boleh meminta sumber daya lain. Maksudnya, proses tersebut menunggu hingga benar-benar sumber daya yang diminta tidak dipakai oleh proses lain. Hal ini bisa menyebabkan apa yang disebut sebagai kelaparan sumber daya, sebab bisa saja sebuah proses tidak mendapat sumber daya dalam waktu yang lama

3. *No Preemption*

   Sumber daya yang ada pada sebuah proses tidak boleh diambil begitu saja oleh proses lainnya. Sumber daya tersebut harus dilepaskan dengan sendirinya oleh proses yang memegangnya. Jadi, keseluruhan proses menunggu dan mempersilahkan proses yang memiliki sumber daya untuk menjalankan prosesnya.

4. *Circular Wait*

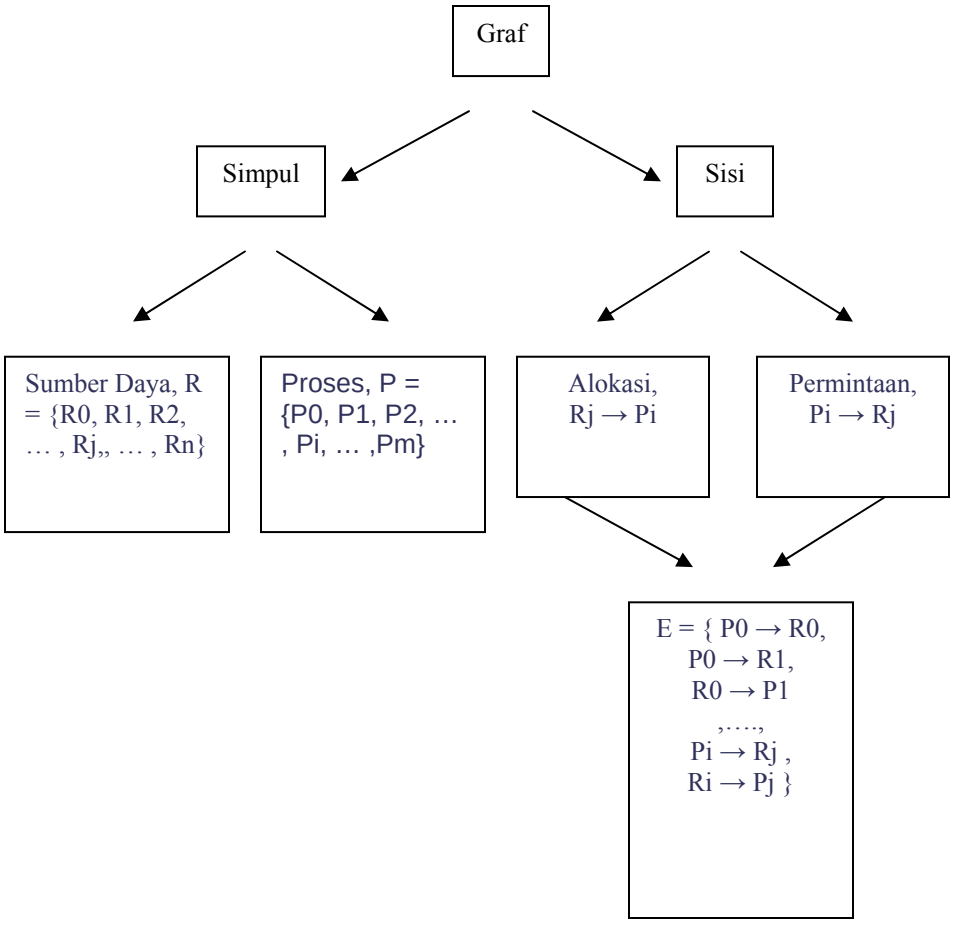
   Circular wait ditandai dengan adanya kondisi seperti rantai (*cycle* -untuk graf-), yaitu setiap proses menuggu sumber daya yang dipegang proses berikutnya.

2. Graf Alokasi Sumber Daya

Graf alokasi sumber daya adalah salah satu penerapan graf pada sistem operasi. Graf aloksi sumber daya merupakan graf sederhana dan graf berarah [3] . Graf ini merupakan visualisasi yang membantu proses pendeteksian dan penanganan masalah *deadlock*.

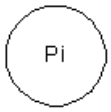
2.1 Komponen Graf Alokasi Sumber Daya

Pada graf alokasi sumber daya, simpul dan sisinya dibedakan menjadi dua.



2.1.1 Jenis-jenis Simpul Graf Alokasi Sumber Daya

1. Proses P= {P0, P1, P2, P3, , Pi, , Pm}. Terdiri dari semua proses yang ada di sistem. Simpulnya digambarkan sebagai lingkaran dengan nama prosesnya. Gambar berikut menunjukkan simpul untuk proses Pi

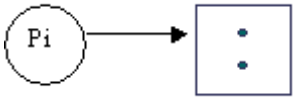


2. Sumber daya R= {R0, R1, R2, R3, , Rj, , Rn}. Terdiri dari semua sumber daya yang ada di sistem. Simpulnya digambarkan sebagai segi empat dengan

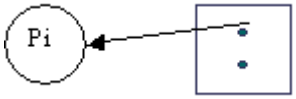
instans yang dapat dialokasikan serta nama sumber dayanya.

2.1.2 Jenis-jenis Sisi Graf Alokasi Sumber Daya

1. Sisi permintaan: Pi -> Rj Sisi permintaan menggambarkan adanya suatu proses Pi yang meminta sumber daya Rj.

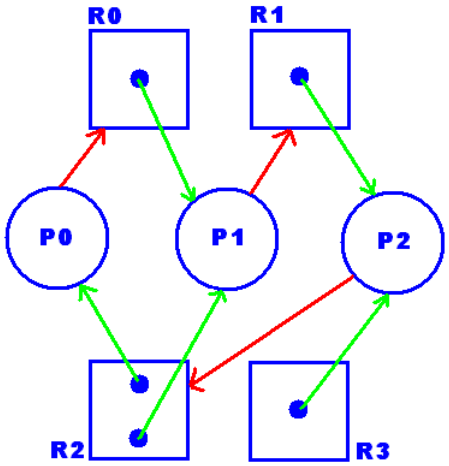


2. Sisi alokasi: Rj -> Pi. Sisi alokasi menggambarkan adanya suatu sumber daya Rj yang mengalokasikan salah satu instansnya pada proses Pi.



3. Pendeteksian *Deadlock*

Setelah memodelkan sistem operasi ke dalam bentuk graf alokasi sumber daya, dapat dilakukan pengecekan *deadlock* dengan memperhatikan *cycle* pada graf tersebut. Jika tidak ditemukan *cycle* maka dapat dipastikan sistem berada dalam kondisi aman, tidak akan terjadi *deadlock* pada saat running. Namun jika ditemukan *cycle* pada graf alokasi sumber daya belum tentu terjadi *deadlock*, sistem dikatakan berpotensi mengalami *deadlock*. Graf alokasi sumber daya dengan simpul sumber daya R_i hanya memiliki satu instan **dan** ditemukan *cycle* maka dapat dipastikan sistem akan mengalami *deadlock*.



Pada gambar di atas ditemukan *cycle* yang berpotensi menimbulkan *deadlock*, *cycle* yang pertama R2 → P0 → R0 → P1 → R1 → P2 → R2 dan *cycle* kedua R2 → P1 → R1 → P2 → R2. Pada graf alokasi sumber daya di atas semua sumber daya memiliki satu instans kecuali sumber daya R2. dari graf dapat ditarik beberapa hal tentang system :

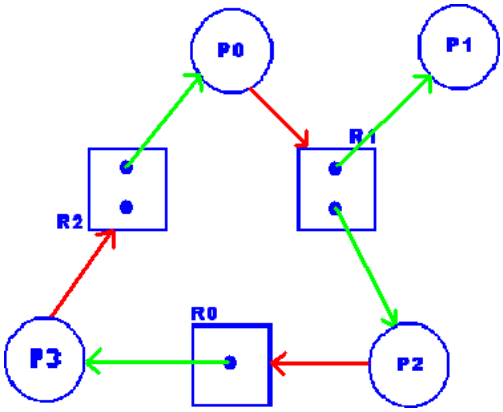
1. P0 meminta sumber daya R0.
2. R0 mengalokasikan sumber dayanya pada P1.
3. P1 meminta sumber daya R1.
4. R1 mengalokasikan sumber dayanya pada P2.
5. P2 meminta sumber daya R2.
6. R2 mengalokasikan sumber dayanya pada P0 dan P1.
7. R3 mengalokasikan sumber dayanya pada P2.

Hal-hal tersebut dapat mengakibatkan *deadlock* sebab :

1. P0 memerlukan sumber daya R0 untuk menyelesaikan prosesnya.
2. R0 dialokasikan untuk P1.
3. P1 memerlukan sumber daya R1.
4. R1 dialokasikan untuk P2.
5. P2 memerlukan sumber daya R2.
6. R2 mengalokasikan sumber dayanya pada P0 dan P1.

Semua sumber daya yang diperlukan oleh suatu proses tidak dapat dipenuhi sehingga proses tersebut tidak dapat melepaskan sumber daya yang telah dialokasikan kepadanya. Akibatnya terjadi proses tunggu-menunggu antar proses yang tidak dapat berakhir.

Selanjutnya akan dibahas graf alokasi sumber daya dengan *cycle* namun *cycle* tersebut tidak mengakibatkan *deadlock*.



Graf alokasi sumber daya tersebut memiliki *cycle* tetapi tidak terjadi *deadlock*. *Cycle* graf tersebut adalah **P0 –> R1 –> P2 –> R0 –> P3 –> R2 –> P0.**

Graf di atas menunjukkan beberapa hal:

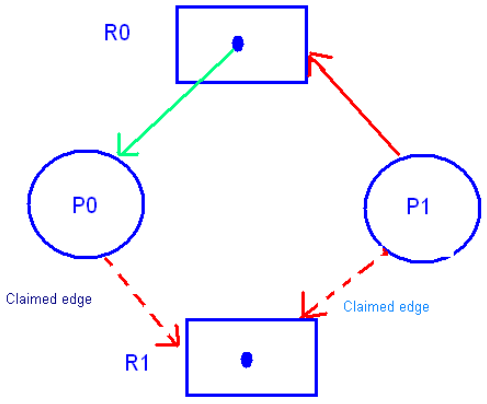
1. P0 meminta sumber daya R1.
2. R1 mengalokasikan sumber dayanya pada P2.
3. P2 meminta sumber daya R0.
4. R0 mengalokasikan sumber dayanya pada P3.
5. P3 meminta sumber daya R2.
6. R0 mengalokasikan sumber dayanya pada P3.
7. R1 mengalokasikan sumber dayanya pada P1.

Hal ini tidak menyebabkan *deadlock* walaupun ada *cycle* sebab semua sumber daya diperlukan sP1 dapat terpenuhi sehingga P1 dapat melepaskan semua sumber dayanya untuk digunakan oleh proses lain.

4. Pencegahan *Deadlock*

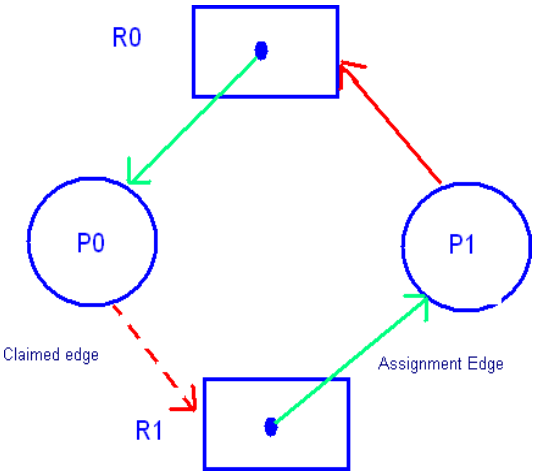
Berikut ini dipaparkan metode yang dapat dipakai untuk mencegah *deadlock* jika sumber daya hanya memiliki satu instans. Pada metode ini ada komponen tambahan pada sisi yaitu *claimed edge*. Sama halnya dengan sisi yang lain, *claimed edge* menghubungkan antara sumber daya dan proses.

Claimed edge Pi → Rj berarti bahwa proses Pi akan meminta sumber daya Rj pada suatu waktu. *Claimed edge* sebenarnya merupakan sisi yang digambarkan sebagai garis putus-putus. Ketika proses Pi memerlukan sumber daya Rj, *claimed edge* diubah menjadi sisi permintaan. Dan setelah proses Pi selesai menggunakan Rj, sisi alokasi diubah kembali menjadi *claimed edge.*



Jika tidak ada perputaran dalam graf, maka sistem berada dalam status aman. Tetapi jika perputaran ditemukan maka sistem berada dalam status tidak aman. Pada saat status tidak aman ini, proses Pi harus menunggu sampai

permintaan sumber dayanya dipenuhi. Gambar berikut menunjukkan bahwa sistem dalam keadaan tidak aman.



5. Kesimpulan

Deadlock ialah suatu kondisi permanen dimana proses tidak berjalan lagi ataupun tidak berkomunikasi lagi antar proses. Sebenarnya *deadlock* dapat disebabkan oleh empat hal yaitu :

1. Proses *Mutual Exclusion*

2. Proses memegang dan menunggu

3. Proses *Preemption*

4. Proses Menunggu dengan siklus *deadlock* tertentu

Untuk mendeteksi *deadlock* dan menyelesaikannya dapat digunakan graf alokasi sumber daya sebagai visualisasinya. Jika tidak ada *cycle*, berarti tidak ada *deadlock*. Jika ada *cycle*, berpotensi terjadi *deadlock*. Graf alokasi sumber daya dengan satu instan dan *cycle* mengakibatkan *deadlock*.

Pencegahan *deadlock* dapat dilakukan dengan menambahkan *claimed edge* pada graf. *Claimed edge* Pi → Rj berarti bahwa proses Pi akan meminta sumber daya Rj pada suatu waktu. Ketika proses Pi memerlukan sumber daya Rj, *claimed edge* diubah menjadi sisi permintaan. Dan setelah proses Pi selesai menggunakan Rj, sisi alokasi diubah kembali menjadi *claimed edge.*

DAFTAR PUSTAKA

[1] Kristiana. (2006). Bab 24 Diagram Graf. *http://www.unej.ac.id/fakultas/mipa/jid/vol5no1/kristiana.pdf*. Tanggal akses: 13 Desember 2006 pukul 16:00.

[2] Munir, Rinaldi. (2006). Diktat Kuliah IF2153 Matematika Diskrit. Departemen Teknik Informatika, Institut Teknologi Bandung.

[3] Yuliarso, Dani dkk. (2003). Sistem Operasi. Gabungan Kelompok Kerja 2-28 Semester Genap 2002/2003 dan 41-49 Semester Ganjil 2003/2004.

[4] Silberschat. (2003). *"Operating System Concepts -- Fourth Edition"*. John Wiley & Sons.