Pendeteksian *Deadlock* dengan Algoritma Runut-balik

Rita Wijaya - 13509098
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Rita.wijaya@students.itb.ac.id

***Abstraksi*—Sistem operasi adalah seperangkat program yang mengelola sumber daya perangkat keras komputer dan menyediakan layanan umum untuk aplikasi perangkat lunak. Salah satu masalah pada sistem operasi dalam mengelola penggunaan sumber daya adalah terjadinya *deadlock*. Suatu himpunan proses disebut *deadlock* apabila masing-masing proses pada himpunan tersebut menunggu *event* yang hanya dapat disebabkan oleh proses lainnya pada himpunan proses tersebut. Terjadinya *deadlock* dapat dideteksi dengan mengaplikasikan algoritma runut-balik. Algoritma runut-balik sendiri merupakan algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus.**

***Kata Kunci*—algoritma runut-balik, *deadlock*, pendeteksian *deadlock*, sistem operasi.**

I. PENDAHULUAN

Salah satu tugas utama dari sistem operasi adalah menjadwalkan proses yang akan berjalan serta mengalokasikan sumber daya bagi proses yang memerlukannya. Sistem operasi terkini mengizinkan lebih dari proses aktif pada saat bersamaan. Di antara proses-proses yang aktif tersebut, mungkin ada yang mengakses sumber daya yang sama. Namun, kita tahu bahwa sumber daya yang ada jumlahnya terbatas. Apabila suatu proses tidak memperoleh sumber daya yang diinginkannya, maka proses tersebut tidak dapat selesai dan akan terus menunggu. Oleh karena itu, penting bagi sistem operasi untuk mengalokasikan suatu sumber daya pada proses yang tepat, tentu saja pada saat yang tepat.

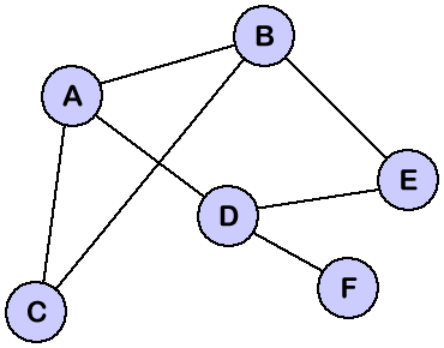
Apa yang terjadi apabila terjadi kesalahan dalam pengambilan keputusan untuk memberi suatu proses sumber daya? Pada kasus paling sederhana, hal ini mungkin hanya menyebabkan adanya proses lain yang harus menunggu lebih lama untuk melanjutkan prosesnya. Proses yang menunggu itu mungkin saja tidak akan pernah memperoleh gilirannya, sehingga eksekusi proses tidak pernah selesai. Namun, hal paling buruk yang dapat terjadi adalah *deadlock*. Mungkin saja proses yang diberikan sumber daya itu ternyata akan mengakses sumber daya lain yang sedang dipegang oleh proses yang juga ingin mengakses sumber daya yang baru diberikan tersebut. Kedua proses ini akan saling menunggu terus-menerus dan tidak akan pernah selesai. Bayangkan apabila proses yang terlibat dalam *deadlock* ini berjumlah sangat banyak, *deadlock* akan menjadi fenomena yang mengerikan dan ditakuti.

Sebelum *deadlock* terjadi, sebaiknya dilakukan pencegahan dengan cara memperhitungkan keadaan sistem setelah suatu sumber daya diberikan sebelum sumber daya tersebut sungguh-sungguh diberikan kepada suatu proses. Namun, apabila *deadlock* sudah terjadi, sistem harus dapat mendeteksinya terlebih dahulu untuk melakukan tindakan lebih lanjut. Pada dasarnya, teknik pencegahan dan pendeteksian adalah sama. Pada bab berikutnya akan dijelaskan lebih detil mengenai teknik pendeteksian *deadlock*.

II. TEORI DASAR

A. Graf

Graf G didefinisikan sebagai pasangan himpunan (V,E) ditulis dengan notasi G=(V,E), yang dalam hal ini V adalah himpuynan tidak-kosong dari simpul-simpul (vertices atau node) dan E adalah himpunan sisi (edges atau arcs) yang menghubungkan sepasang simpul.



Gambar 1 Graf tidak berarah dengan 6 simpul dan 7 sisi

Sebagai contoh definisi dari graf pada gambar 1 di atas adalah :

V = {1,2,3,4,5,6}

E = {(1,2),(1,5),(2,3),(3,4),(4,5),(5,2),(4,6)}.

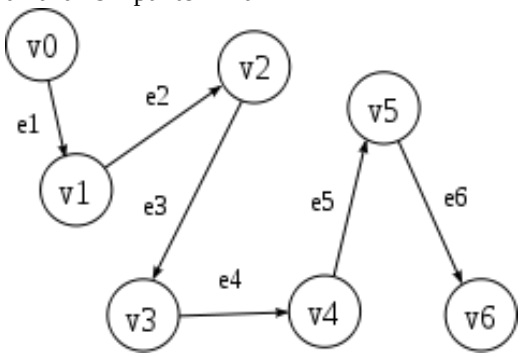
Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas 2 jenis:

1. Graf tak-berarah

   Graf tak berarah adalah graf yang sisinya tidak memiliki orientasi arah. Graf pada gambar 1 di atasmerupakan salah satu contoh dari graf tak-berarah. Pada graf tak-berarah, sisi (u,v) dan (v,u) adalah sama.

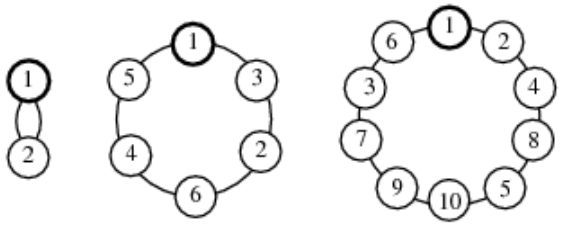
2. Graf Berarah

Graf berarah adalah graf yang setiap sisinya diberikan orientasi arah. Untuk busur (u,v) pada grak berarah, simpul u diamakan simpul asal dan simpul v dinamakan simpul terminal



Gambar 2 Graf berarah

Dalam graf, dikenal juga istilah lintasan. Lintasan yang panjangnya n dari simpul awal v_0 ke simpul tujuan v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk v_0, e_1, v_1, e_2, v_2, …, v_{n-1}, e_n, v_n sedemikian sehingga $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, …, $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G. Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus. Supaya lebih jelas, perhatikan gambar siklus pada gambar 3.



Gambar 3 Siklus

Suatu graf tak-berarah G disebut graf terhubung jika untuk setiap simpul u dan v di dalam himpunan V terdapat lintasan dari u ke v (yang juga berarti ada lintasan dati v ke u). Jika tidak, maka G disebut graf tak-terhubung.

B. Algoritma Runut-balik

Runut-balik (*backtracking*) aadlah algoritma yang berbasis pada *DFS* untuk mencari solusi persoalan secara lebih mangkus. Runut-balik secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada. Dengan metode ini, kita tidak perlu memeriksa semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan. Runut-balik lebih alami dinyatakan dalam algoritma rekursif. Kadang-kadang disebutkan pula bahwa runut-balik merupakan bentuk tipikal dari algoritma rekursif.

Untuk menerapkan metode runut-balik, berikut didefinisikan properti umumnya.

1. Solusi Persoalan

   Solusi dinyatakan sebagai vektor dengan n-*tuple*.
   $X = (x_1, x_2, ..., x_n)$, $x_i \epsilon$ himpunan berhingga S_i.
   Mungkin saja $S_1=S_2= ... =S_n$
   Contoh : $S_i = \{0,1\}$,
   $X_i = 0$ atau 1
2. Fungsi Pembangkit nilai x_k
   Dinyatakan sebagai :
   T(k)
   T(k) membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.
3. Fungsi Pembatas(dalam beberapa persoalan fungsi ini dinamakan fungsi kriteria)
   Dinyatakan sebagai
   $B(x_1, x_2, …, x_k)$
   Fungsi pembatas menentukan apakah $(x_1, x_2, …, x_k)$ mengarah ke solusi. Jika ya, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika tidak , maka $(x_1, x_2, ..., x_k)$ dibuang dan tidak akan dipertimbangkan lagi dalam pencarian solusi.

Langkah-langkah pencarian solusi dengan metode runut-balik ini adalah sebagai berikut:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti metode pencarian dalam (*DFS*). Simpul–simpul yang sudah dilahirkan dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand-node*). Simpul dinomori dari atas ke bawah sesuai dengan urutan kelahirannya.
2. Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut "dibunuh" sehingga menjadi simpul mati (*dead node*). Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan fungsi pembatas. Simpul yang sudah mati tidak akan pernah diperluas lagi.
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencaran solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat (simpul orangtua). Sealnjutnya simpul ini menjadi simpul E yang baru. Lintasan baru dibangun kembali sampai lintasan tersebut membentuk solusi.
4. Pencarian dihentikan bila telah ditemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

C. *Deadlock*

Secara formal, suatu himpunan proses disebut mengalami *deadlock* jika masing-masing proses dalam himpunan tersebut sedang menunggu terjadinya *event* yang hanya dapat disebabkan oleh proses lain dalam himpunan tersebut. Karena seluruh proses sedang menunggu, tidak akan ada satu pun dari proses itu yang menyebabkan terjadinya suatu *event* yang dapat

'membangunkan' proses lain. Akibatnya, seluruh proses akan menunggu terus-menerus selamanya.

Terdapat 4 kondisi yang harus dipenuhi sehingga *deadlock* dapat terjadi:

1. Setiap sumber daya sedang dialokasikan untuk tepat 1 proses atau sumber daya tersebut dapat digunakan.
2. Proses yang telah memperoleh sumber daya dapat meminta sumber daya baru.
3. Sumber daya yang telah diberikan harus dilepas secara eksplisit oleh proses yang memegangnya, tidak boleh diambil secara paksa.
4. Ada rantai sirkular yang terdiri dari 2 atau lebih proses yang masing-masing menunggu sumber daya yang dimiliki oleh anggota di depannya.

Secara umum, ada 4 strategi yang dapat digunakan untuk menghadapi *deadlock*:

1. Mengacuhkannya
   Tidak ada penanganan khusus, anggap saja tidak ada masalah.
2. Deteksi dan perbaiki
   Deteksi dan perbaiki: biarkan *deadlock* terjadi, kemudian deteksi dan lakukan penanganan.
3. Penghindaran dinamis
   Dengan strategi ini, kita harus berhati-hati dalam memberikan suatu sumber daya pada proses. Hal ini dilakukan dengan melakukan pengecekan setiap akan mengambil keputusan. Apabila berpotensi mengakibatkan *deadlock*, maka pemberian sumber daya ditunda.
4. Pencegahan
   Dengan strategi ini, pencegahan dilakukan secara stuktural dengan menghilangkan salah satu dari 4 kondisi yang menyebabkan *deadlock*.

III. PEMBAHASAN

A. Cara Pendeteksian *Deadlock* dengan Graf

Kita dapat memodelkan status proses dan sumber daya menggunakan graf berarah sehingga kita dapat meilhat kondisi proses secara keseluruhan. Graf tersebut memiliki 2 jenis simpul:

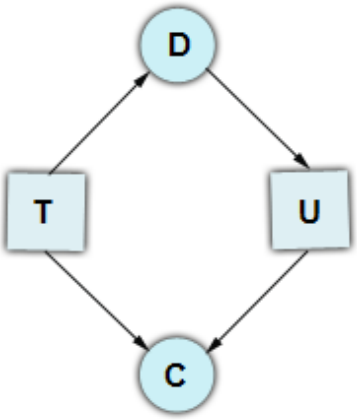
1. proses (dilambangkan dengan lingkaran)
2. sumber daya (dilambangkan dengan kotak).

Busur dari sumber daya yang menuju proses berarti proses tersebut sedang memegang sumber daya tersebut. Pada Gambar 4 di bawah, terdapat sumber daya R yang menunjuk ke proses A, hal ini berarti proses A sedang memegang sunber daya B. Sedangkan busur dari proses yang menuju sumber daya berarti proses tersebut sedang menunggu untuk mendapatkan sumber daya tersebut. Sebagai contoh, perhatikan Gambar 5 di bawah. Pada gambar tersebut sumber daya S yang ditunjuk oleh proses B, hal ini berarti proses B meminta sumber daya S dan sedang menunggu pengalokasian sumber daya S untuknya.

Gambar 4 Proses A sedang memegang sumber daya B

Gambar 5 Proses B sedang meminta sumber daya S

Dengan diketahuinya status proses dan sumber daya, keempat kondisi penyebab *deadlock* juga dapat kita lihat dengan mudah. Apabila pada graf terdapat siklus, maka itu berarti setiap proses yang membentuk siklus itu mengalami *deadlock*. Supaya lebih jelas, perhatikan Gambar 6 berikut.



Gambar 6 *Deadlock*

Dari gambar tersebut, dapat kita lihat bahwa proses C sedang menunggu sumber daya T. Namun, sumber daya T sedang dipegang oleh proses D. Proses D belum dapat melepas sumber daya T karena proses D sendiri sedang menunggu sumber daya U yang dimiliki oleh proses C. Dengan demikian, kedua proses tersebut tidak akan pernah mendapatkan sumber daya yang diinginkan dan harus menunggu selamanya. Pada gambar, keadaan *deadlock* ini dapat dilihat dari siklus yang dibentuk oleh D-U-C-T-D.

B. Algoritma Pendeteksian *Deadlock*

Berdasarkan pendekatan menggunakan graf, kita tahu bahwa adanya siklus pada graf menandakan adanya *deadlock*. Algoritma pendeteksian berikut akan mencari adanya siklus pada graf menggunakan algoritma runut-balik dengan langkah-langkah:

1. Untuk setiap simpul N pada graf, lakukan 5 langkah di bawah ini dengan N sebagai node awal.
2. Buatlah sebuah *list* kosong L, dan inisialisasi semua *isVisited* pada busur dengan *false* (artinya semua busur belum ditandai).

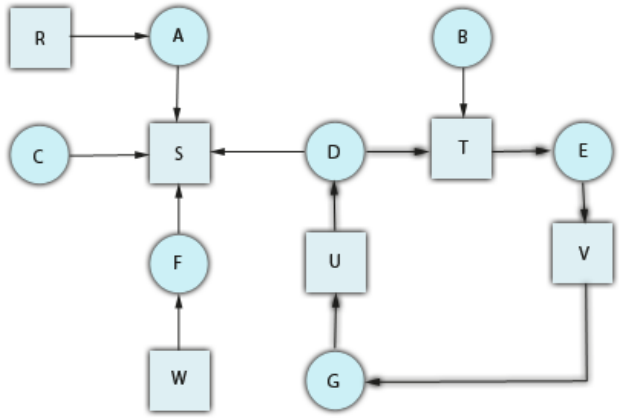
3. Cek apakah simpul tempat kita berada sekarang merupakan anggota dari *list* L. Apabila iya, berarti graf mengandung siklus, hentikan pencarian. Apabila tidak, masukkan simpul tempat kita berada sekarang ke akhir *list* L.
4. Cari busur yang belum dikunjungi yang keluar dari simpul tempat kita berada sekarang. Jika busur tersebut tidak ditemukan, maka lakukan langkah keenam. Jika ada, lanjutkan ke langkah kelima.
5. Dari busur yang ditemukan, pilih salah satu untuk diikuti. Tandai *isVisited* busur tersebut menjadi *true*, kemudian ikuti busur tersebut hingga kita menemukan simpul baru dan ulangi langkah ketiga.
6. Akhirnya kita tiba di daun graf. Hapus simpul terakhir tersebut dari *list* L dan kembali ke simpul sebelumnya. Ulangi pengecekan dari langkah ketiga. Apabila ia tidak memiliki simpul sebelumnya (ia merupakan awal *list* L), berarti kita tiba di simpul tempat kita memulai algoritma. Dengan demikian, graf tidak memiliki siklus dan aloritma dapat dihentikan.

Dalam implementasinya, graf berarah tersebut dapat kita representasikan dalam bentuk matriks berukuran mxn, dengan m merupakan jumlah proses dan n merupakan jumlah sumber daya yang ada. Matriks tersebut kemudian diisi dengan bilangan integer ϵ {0, 1, 2} yang merepresentasikan status proses-sumber daya. Misalnya pada matriks M, maka:

1. M[i][j] = 0, artinya tidak ada busur yang menghubungkan proses ke-i dengan sumber daya ke-j.
2. M[i][j] = 1, artinya terdapat busur dari proses ke-i menuju sumber daya ke-j
3. M[i][j] = 2, artinya terdapat busur dari sumber daya ke-j menuju proses ke-i.

C. Hasil Kerja Algoritma Pendeteksian *Deadlock*

Misalkan pada suatu saat, keadaan proses yang digambarkan dengan graf adalah sebagai berikut.



Gambar 7 Status Proses dan Sumber Daya dalam Graf

Pada kasus ini, terdapat 7 proses dan 6 sumber daya. Misalkan proses A-G diberi nomor 1-7 secara berurutan, dan sumber daya R-W diberi nomor 1-6 secara berurutan. Dengan demikian, keadaan kondisi matriks masukan adalah sebagai berikut.

No.	1	2	3	4	5	6	7
1	2	0	0	0	0	0	0
2	1	0	1	1	0	1	0
3	0	1	0	1	2	0	0
4	0	0	0	2	0	0	1
5	0	0	0	0	1	0	2
6	0	0	0	0	0	2	0

Sebagai keterangan, nomor kolom menunjukkan nomor proses, sedangkan nomor baris menunjukkan nomor sumber daya. Apabila matriks yang sama dijadikan input untuk fungsi DeadlockFound(matriks), maka hasil output program yang ditulis dalam bahasa C++ disertai keadaan *list* setiap tahapnya akan sama seperti gambar 8 di bawah ini.

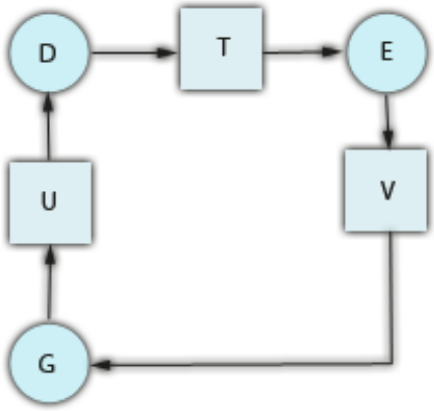
```
Matriks masukan:
2 0 0 0 0 0 0
1 0 1 1 0 1 0
0 1 0 1 2 0 0
0 0 0 2 0 0 1
0 0 0 0 1 0 2
0 0 0 0 0 2 0

Proses Backtrack:
proses 1
proses 2
proses 2 - source 3
proses 2 - source 3 - proses 5
proses 2 - source 3 - proses 5 - source 5
proses 2 - source 3 - proses 5 - source 5 - proses 7
proses 2 - source 3 - proses 5 - source 5 - proses 7 - source 4
proses 2 - source 3 - proses 5 - source 5 - proses 7 - source 4 - proses 4
proses 3
proses 4
proses 4
proses 4 - source 3
proses 4 - source 3 - proses 5
proses 4 - source 3 - proses 5 - source 5
proses 4 - source 3 - proses 5 - source 5 - proses 7
proses 4 - source 3 - proses 5 - source 5 - proses 7 - source 4

Deadlock ditemukan
```

Gambar 8 Hasil Output Program Pendeteksian Deadlock

Dari hasil eksekusi program ini diketahui bahwa ternyata terdapat *deadlock*. Deadlock ini terjadi antara proses 4, sumber daya 3, proses 5, sumber daya 5, proses 7 (proses D, sumber daya T, proses E, sumber daya V, proses G), sesuai potongan graf dari gambar 7 berikut.



Gambar 9 *Deadlock*

E. Aplikasi Algoritma Pendeteksian *Deadlock*

Algoritma pendeteksian *deadlock* ini dapat dimanfaatkan juga sebagai algoritma pencegahan *deadlock* dengan sedikit perubahan. Jika digunakan untuk mendeteksi, maka keadaan sekarang yang dimasukkan sebagai masukan. Namun, apabila ingin digunakan untuk mencegah *deadlock*, maka keadaan berikutnya setelah pemberian sumber daya yang dijadikan sebagai input. Jika pada hasil pengujian terjadi *deadlock*, maka sumber daya tidak jadi diberikan. Namun, jika hasil pengujian aman dari *deadlock*, maka sumber daya dapat diberikan.

Meskipun terdapat algoritma untuk menangani *deadlock* seperti ini, namun sebagian besar sistem operasi termasuk *Windows* dan *Linux* tidak melakukan penanganan terhadap *deadlock*. Hal ini disebabkan karena untuk melakukan pengujian seperti ini diperlukan waktu yang tidak sedikit dan akan merusak performansi pemrosesan. Oleh karena itu, hingga kini *deadlock* masih menjadi salah satu permasalahan yang diteliti oleh ilmuwan informatika.

IV. KESIMPULAN

Algoritma runut-balik dapat diaplikasikan untuk mendeteksi *deadlock*. Algoritma pencarian *deadlock* ini masih perlu ditingkatkan kemangkusannya supaya dapat diimplementasikan pada sistem operasi tanpa menurunkan performansi yang sudah ada.

REFERENSI

[1] Munir, Rinaldi, *Diktat Kuliah IF3051 Strategi Algoritma.* Program Studi Teknik Informatika ITB, 2009.

[2] Munir, Rinaldi, *Matematika Diskrit.* Informatika Bandung, Bandung, Agustus 2005.

[3] http://id.wikipedia.org/wiki/Sistem_operasi
tanggal akses: 7 Desember 2011

[4] http://en.wikipedia.org/wiki/Deadlock
tanggal akses: 8 Desember 2011

[5] Tanenbaum, *Modern Operating Systems 2nd Edition*, Prentice-Hall International, 2001.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Desember 2011

Rita Wijaya (13509098)