KOMPONEN DASAR SISTEM OPERASI, *DEADLOCK*, DAN IMPLEMENTASI GRAF UNTUK MENDETEKSI *DEADLOCK* PADA SISTEM OPERASI

Monterico Adrian – NIM : 13505036

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
E-mail : if15036@students.if.itb.ac.id

Abstrak

Sistem operasi didefinisikan sebagai suatu program yang mengatur perangkat keras komputer, dengan menyediakan landasan untuk aplikasi yang berada di atasnya, serta bertindak sebagai penghubung antara pengguna dengan perangkat keras. Sistem operasi bertugas untuk mengendalikan (kontrol) serta mengkoordinasikan penggunaan perangkat keras untuk berbagai aplikasi untuk pengguna. Pengertian dari sistem operasi juga dapat dilihat dari berbagai sudut pandang. Dari sudut pandang pengguna, sistem operasi merupakan alat untuk mempermudah penggunaan komputer. Sebaliknya dalam lingkungan berpengguna banyak (*multi-user*), sistem operasi dapat dipandang sebagai alat untuk memaksimalkan penggunaan sumber daya komputer. Dari sudut pandang sistem, sistem operasi dapat dianggap sebagai alat yang menempatkan sumber daya secara efisien (*resource allocator*). Sistem operasi ialah manajer bagi sumber daya, yang menangani konflik permintaan sumber daya secara efisien. Sistem operasi juga mengatur eksekusi aplikasi dan operasi dari alat M/K (Masukan/Keluaran). Fungsi ini dikenal juga sebagai program pengendali (*control program*). Lebih lagi, sistem operasi merupakan suatu bagian program yang berjalan setiap saat yang dikenal dengan istilah kernel. Dari sudut pandang tujuan sistem operasi, sistem operasi dapat dipandang sebagai alat yang membuat komputer lebih nyaman digunakan (*convenient*) untuk menjalankan aplikasi dan menyelesaikan masalah pengguna. Dan ternyata komponen-komponen dasar sistem operasi memakai implementasi dari matematika diskrit.

Deadlock adalah suatu kondisi dimana proses tidak berjalan lagi ataupun tidak ada komunikasi lagi antarproses di dalam sistem operasi. *Deadlock* disebabkan karena proses yang satu menunggu sumber daya yang sedang dipegang oleh proses lain yang sedang menunggu sumber daya yang dipegang oleh proses tersebut. Dengan kata lain setiap proses dalam *set* menunggu untuk sumber yang hanya dapat dikerjakan oleh proses lain dalam *set* yang sedang menunggu. Jadi tidak ada satu pun proses yang bisa *running*, melepaskan sumber daya, atau dibangunkan. Sumber daya, proses, dan *deadlock* tersebut dapat digambarkan dengan graf.

Sedangkan graf adalah suatu struktur diskrit yang terdiri dari simpul dan sisi, dimana sisi menghubungkan simpul-simpul yang ada. Berdasarkan hubungan antara sisi dan simpulnya, graf dibagi menjadi dua, yaitu graf sederhana dan graf tak-sederhana. Graf sederhana tidak mengandung sisi paralel (lebih dari satu sisi yang menghubungkan dua simpul yang sama). Berdasarkan arahnya graf dapat dibagi menjadi dua yaitu graf berarah dan graf tidak berarah. Graf berarah memperhatikan arah sisi yang menghubungkan dua simpul, sedangkan graf tidak berarah tidak memperhatikan arah sisi yang menghubungkan dua simpul. Dalam hal ini akan dibahas mengenai implementasi graf dalam sistem operasi, yaitu penggunaannya untuk penanganan *deadlock* pada sistem operasi. Diantaranya adalah graf alokasi sumber daya dan graf tunggu. Graf alokasi sumber daya dan graf tunggu merupakan graf sederhana dan graf berarah. Dua graf tersebut adalah bentuk visualisasi dalam mendeteksi masalah *deadlock* pada sistem operasi.

Setiap sumber daya pada sistem operasi akan digunakan oleh proses-proses yang membutuhkannya. Mekanisme hubungan dari proses-proses dan sumber daya yang dibutuhkan atau digunakan itu dapat di diwakilkan dan digambarkan dengan graf alokasi sumber daya dan graf tunggu. Dengan adanya visualisasi dari graf tersebut, maka masalah *deadlock* pada sistem operasi dapat dideteksi dan diselesaikan.

Kata kunci: sistem operasi, *deadlock*, graf, sumber daya, diskrit, komponen dasar

1. Pendahuluan

Sebuah sistem komputer terdiri dari berbagai macam sumber daya (*resources*), seperti fisik (perangkat, memori), logik (*lock*, *database record*), sistem operasi (*printed circuit board slots*), dan aplikasi (berkas). Sumber daya itu sendiri adalah suatu komoditas yang diperlukan oleh proses. Berdasarkan cara pemakaiannya, sumber daya dibagi menjadi dua yaitu sumber daya yang bisa dipakai berulang-ulang (*serially reusable*) dan sumber daya sekali pakai (*consummable*). Sumber daya ada yang bersifat *preemptible* (contohnya CPU dan memori) dan *non-premptible* (contohnya *tape drives*). Sumber daya bisa digunakan secara bersama oleh beberapa proses maupun terdedikasi (secara ekslusif oleh satu proses saja). Urutan *event* yang dilakukan dalam penggunaan sumber daya adalah permintaan, penggunaan, dan pelepasan. Jika *request* ditolak, maka proses harus menunggu dengan cara diblokir atau mengeluarkan kode kesalahan. Kejadian *deadlock* selalu tidak lepas dari sumber daya dan proses tersebut, yaitu masalah sumber daya yang digunakan bersama-sama.

Deadlock pada sumber daya yang bisa dipakai berulang-ulang bisa terjadi di prosesor, kanal Masukan/Keluaran, *disk*, dan semafor. Contoh misalnya sebuah proses memakai *disk* A dan B, maka akan terjadi *deadlock* jika setiap proses sudah memiliki salah satu disk dan meminta disk yang lain. Contoh lainnya adalah yang berkaitan dengan jumlah proses yang memakai memori utama. Cara mengatasinya adalah dengan memakai memori maya (*virtual*),

Sedangkan contoh *deadlock* pada sumber daya sekali pakai misalnya ada dua fungsi yang saling berproses. Kedua fungsi tersebut ada yang bertindak untuk menerima dan memberi sumber daya, tetapi ada kalanya proses tidak mendapat sumber daya yang dibuat sehingga terjadi blok, karena itu terjadi *deadlock*. Tentu saja hal ini sangat jarang terjadi mengingat tidak ada batasan untuk memproduksi dan mengkonsumsi, tetapi ada suatu keadaan seperti ini yang mengakibatkan *deadlock*. Hal ini mengakibatkan *deadlock* jenis ini sulit untuk dideteksi. Selain itu *deadlock* ini dihasilkan oleh beberapa kombinasi yang sangat jarang terjadi.

Sistem operasi adalah salah satu sumber daya yang sangat penting. Sistem operasi memiliki delapan komponen dasar yaitu manajemen proses, manajemen memori utama, manajemen berkas, manajemen sistem masukan/keluaran, manajemen penyimpanan sekunder, sistem proteksi, jaringan, dan *command-interpreter system*. Bila komponen-komponen dasar ini tidak berjalan dengan baik, maka sistem operasi akan mengalami gangguan. Gangguan yang sering terjadi pada sistem operasi adalah *deadlock*. Salah satu cara pendeteksiannya adalah dengan memakai visualisasi dari graf.

2. Komponen Sistem Operasi

2.1 Manajemen Proses

Proses adalah sebuah program yang sedang dieksekusi. Sebuah proses membutuhkan beberapa sumber daya untuk menyelesaikan tugasnya. Sumber daya tersebut dapat berupa *CPU time*, memori, berkas, dan perangkat M/K. Sistem operasi mengalokasikan sumber daya tersebut saat proses itu diciptakan atau sedang diproses atau dijalankan. Ketika proses tersebut berhenti dijalankan, sistem operasi akan mendapatkan kembali semua sumber daya yang bisa digunakan kembali.

Sistem operasi bertanggung jawab atas semua aktivitas yang berkaitan dengan manajemen proses seperti membuat dan menghapus proses pengguna dan sistem proses, menunda atau melanjutkan proses, menyediakan mekanisme untuk proses sinkronisasi, menyediakan mekanisme untuk proses komunikasi, dan menyediakan mekanisme untuk penanganan *deadlock*. Komunikasi antarproses dapat digambarkan dengan graf. Jadi, manajemen proses memakai implementasi struktur diskrit graf.

2.2 Manajemen Memori Utama

Memori utama atau lebih dikenal sebagai memori adalah sebuah *array* yang besar dari *word* atau *byte*, yang ukurannya mencapai ratusan, ribuan, atau bahkan jutaan. Setiap *word* atau byte mempunyai alamat tersendiri. Memori utama berfungsi sebagai tempat penyimpanan instruksi atau data yang akses datanya digunakan oleh CPU dan perangkat Masukan/Keluaran. Memori utama termasuk tempat penyimpanan data yang bersifat volatile (tidak permanen), yaitu data akan hilang bila komputer dimatikan.

Sistem operasi bertanggung jawab atas semua aktivitas yang berkaitan dengan manajemen memori seperti menjaga *track* dari memori

yang sedang digunakan serta siapa yang menggunakannya dan memilih program yang akan di-*load* ke memori. Karena menggunakan rangkaian digital, maka manajemen memori utama memakai implementasi struktur diskrit aljabar boolean.

2.3 Manajemen Berkas

Berkas adalah kumpulan informasi yang berhubungan dan sesuai dengan tujuan pembuat berkas tersebut. Umumnya berkas merepresentasikan program dan data. Berkas dapat mempunyai struktur yang bersifat hirarkis (direktori, volume, dll.). Sistem operasi mengimplementasikan konsep abstrak dari berkas dengan mengatur media penyimpanan massa, misalnya *tapes* dan *disk*.

Sistem operasi bertanggung jawab atas semua aktivitas yang berhubungan dengan manajemen berkas seperti pembuatan dan penghapusan berkas, pembuatan dan penghapusan direktori, mendukung manipulasi berkas dan direktori, memetakan berkas ke *secondary storage,* dan mem-*back-up* berkas ke media penyimpanan yang permanen (*non-volatile*). Manajemen berkas memakai struktur diskrit pohon karena struktur direktorinya yang seperti pohon.

2.4 Manajemen Sistem M/K

Sering disebut *device manager*. Menyediakan *device driver* yang umum sehingga operasi M/K dapat seragam (membuka, membaca, menulis, dan menutup). Komponen sistem M/K yaitu penyangga (menampung sementara data dari/ke perangkat M/K), *spooling* (melakukan penjadualan pemakaian M/K sistem supaya lebih efisien), dan menyediakan *driver* (agar dapat melakukan operasi rinci untuk perangkat keras M/K tertentu). Manajemen ini memakai struktur diskrit kompleksitas algoritma karena melakukan penjadualan untuk pengefisienan.

2.5 Manajemen Penyimpanan Sekunder

Data yang disimpan dalam memori utama bersifat sementara dan jumlahnya sangat kecil. Oleh karena itu, untuk menyimpan keseluruhan data dan program komputer dibutuhkan penyimpanan sekunder yang bersifat permanen dan mampu menampung banyak data, sebagai *back-up* dari memori utama. Contoh dari penyimpanan sekunder adalah *harddisk*, disket, *flashdisk*, *optical storage*, dll.

Sistem operasi bertanggung jawab atas semua aktivitas yang berkaitan dengan manajemen *disk* seperti *free-space management*, alokasi penyimpanan, dan penjadualan *disk*. Rangkaian digital yang digunakan menggambarkan struktur diskrit aljabar boolean.

2.6 Sistem Proteksi

Proteksi mengacu pada mekanisme untuk mengontrol akses yang dilakukan oleh program, prosesor, atau pengguna ke sistem sumber daya. Tugas sistem proteksi antara lain membedakan antara penggunaan yang sudah diberi izin dan yang belum, menspesifikasi kontrol untuk dibebankan atau diberi tugas, dan menyediakan alat untuk pemberlakuan sistem. Sistem ini memakai struktur diskrit logika karena berhubungan dengan keamanan.

2.7 Jaringan

Sistem terdistribusi adalah sekumpulan prosesor yang tidak berbagi memori, atau *clock*. Setiap prosesor mempunyai memori dan *clock* tersendiri. Prosesor-prosesor tersebut terhubung melalui jaringan komunikasi. Sistem terdistribusi menyediakan akses pengguna ke berbagai sumber daya sistem. Akses tersebut menyebabkan kecepatan komputasi dan kemampuan penyediaan data meningkat. Hubungannya dapat digambarkan dengan struktur diskrit graf.

2.8 *Command-Interpreter System*

Sistem operasi menunggu instruksi dari pengguna (*command driven*). Program yang membaca instruksi dan mengartikan *control statements* umumnya disebut *control-card interpreter*, *command-line interpreter,* dan terkadang dikenal sebagai *shell*. *Command-Interpreter System* sangat bervariasi dari satu sistem operasi ke sistem operasi yang lain dan disesuaikan dengan tujuan dan teknologi perangkat M/K yang ada. Karena berhubungan dengan instruksi, maka struktur diskrit yang digunakan adalah logika.

3. Kondisi Untuk Terjadinya *Deadlock*

Ada empat kondisi yang dapat mengakibatkan *deadlock,* yaitu:

1. *Mutual Exclusive*
   Mutual Exclusion adalah suatu cara yang menjamin jika ada sebuah proses yang menggunakan variabel atau berkas yang sama

(digunakan juga oleh proses lain), maka proses lain akan dikeluarkan dari pekerjaan yang sama. Jadi, *Mutual Exclusive* terjadi ketika hanya ada satu proses yang boleh memakai sumber daya, dan 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 lagi, maksudnya menunggu hingga benar-benar sumber daya yang diminta tidak dipakai oleh proses lain, hal ini dapat menyebabkan kelaparan sumber daya sebab dapat 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. Untuk mendapatkan sumber daya tersebut, maka harus dilepaskan terlebih dahulu oleh proses yang memegangnya, selain itu seluruh proses menunggu dan mengizinkan hanya proses yang memiliki sumber daya yang boleh berjalan.

4. *Circular Wait*
Kondisi seperti rantai, yaitu sebuah proses membutuhkan sumber daya yang dipegang proses berikutnya.

4. Penanggulangan *Deadlock*

Ada beberapa cara penanggulangan *deadlock*:

A. Mengabaikan masalah *deadlock*

Metode ini lebih dikenal dengan Algoritma Ostrich. Dalam algoritma ini dikatakan bahwa untuk menghadapi *deadlock* ialah dengan berpura-pura bahwa tidak ada masalah apa pun. Hal ini seakan-akan melakukan suatu hal yang fatal, tetapi sistem operasi Windows dan UNIX menanggulangi *deadlock* dengan cara ini dengan tidak mendeteksi *deadlock* dan membiarkannya secara otomatis mematikan (*restart*) program sehingga seakan-akan tidak terjadi apa pun. Jadi jika terjadi *deadlock*, maka tabel akan penuh, sehingga proses yang menjalankan proses melalui operator harus menunggu pada waktu tertentu dan mencoba lagi. Cara ini dilakukan jika *deadlock* jarang terjadi atau algoritma lainnya berbiaya lebih tinggi. Tetapi *trade-off* dari cara ini adalah kenyamanan (*convenience*) versus keakuratan (*correctness*).

B. Mendeteksi dan Memperbaiki

Caranya ialah dengan cara mendeteksi jika terjadi *Deadlock* pada suatu proses maka dideteksi sistem mana yang terlibat di dalamnya. Setelah diketahui sistem mana saja yang terlibat maka diadakan proses untuk memperbaiki dan menjadikan sistem berjalan kembali.

Hal-hal yang terjadi dalam mendeteksi adanya *deadlock* adalah:

a. Permintaan sumber daya dikabulkan selama memungkinkan.
b. Sistem operasi memeriksa adakah kondisi circular wait secara periodik.
c. Pemeriksaan *deadlock* dapat dilakukan setiap ada sumber daya yang hendak digunakan oleh sebuah proses.
d. Memeriksa dengan algoritma tertentu

Ada beberapa cara untuk kembali dari *deadlock*:

1. *Preemption*
Dengan cara untuk sementara waktu menjauhkan sumber daya dari pemakainya dan memberikannya pada proses yang lain. Ide untuk memberi pada proses lain tanpa diketahui oleh pemilik dari sumber daya tersebut tergantung dari sifat sumber daya itu sendiri. Perbaikan dengan cara ini sangat sulit atau dapat dikatakan tidak mungkin. Cara ini dapat dilakukan dengan memilih korban yang akan dikorbankan atau diambil sumber dayanya utuk sementara, tentu saja harus dengan perhitungan yang cukup agar waktu yang dikorbankan seminimal mungkin. Setelah melakukan *preemption* dilakukan pengondisian proses tersebut dalam kondisi aman. Setelah itu proses dilakukan lagi dalam kondisi aman tersebut.

2. Melacak kembali
Setelah melakukan beberapa langkah *preemption*, maka proses utama yang diambil sumber dayanya akan berhenti dan tidak dapat melanjutkan kegiatannya, oleh karena itu dibutuhkan langkah untuk kembali pada keadaan aman dimana proses masih berjalan dan memulai proses lagi dari situ. Tetapi untuk beberapa keadaan sangat sulit

menentukan kondisi aman tersebut, oleh karena itu umumnya dilakukan cara mematikan program tersebut lalu memulai kembali proses. Meski pun sebenarnya lebih efektif jika hanya mundur beberapa langkah saja sampai *deadlock* tidak terjadi lagi. Untuk beberapa sistem mencoba dengan cara mengadakan pengecekan beberapa kali secara periodik dan menandai tempat terakhir kali menulis ke *disk*, sehingga saat terjadi *deadlock* dapat mulai dari tempat terakhir penandaannya berada.

3. Menghentikan (membunuh) proses
Cara yang paling umum ialah membunuh semua proses yang mengalami *deadlock*. Cara ini paling umum dilakukan dan dilakukan oleh hampir semua sistem operasi. Namun, untuk beberapa sistem, kita juga dapat membunuh beberapa proses saja dalam siklus *deadlock* untuk menghindari d*eadlock* dan membiarkan proses lainnya kembali berjalan. Atau dipilih salah satu korban untuk melepaskan sumber dayanya, dengan cara ini maka masalah pemilihan korban menjadi lebih selektif, sebab telah diperhitungkan beberapa kemungkinan jika proses harus melepaskan sumber dayanya.

Kriteria seleksi korban ialah:

1.Yang paling jarang memakai prosesor
2.Yang paling sedikit hasil programnya
3.Yang paling banyak memakai sumber daya sampai saat ini
4.Yang alokasi sumber daya totalnya tersedkit
5.Yang memiliki prioritas terkecil

C. Menghindari *deadlock*

Pada sistem kebanyakan permintaan terhadap sumber daya dilakukan sebanyak sekali saja. Sistem sudah harus dapat mengenali bahwa sumber daya itu aman atau tidak (dalam arti tidak terkena *deadlock*), setelah itu baru dialokasikan.

Ada dua cara yaitu:

1. Jangan memulai proses apa pun jika proses tersebut akan membawa kita pada kondisi *deadlock*, sehingga tidak mungkin terjadi *deadlock* karena ketika akan menuju *deadlock* sudah dicegah.
2. Jangan memberi kesempatan pada suatu proses untuk meminta sumber daya lagi jika penambahan ini akan membawa kita pada suatu keadaan *deadlock*.

Jadi diadakan dua kali penjagaan, yaitu saat pengalokasian awal, dijaga agar tidak *deadlock* dan ditambah dengan penjagaan kedua saat suatu proses meminta sumber daya, dijaga agar jangan sampai terjadi *deadlock*. Pada *deadlock avoidance system,* dilakukan dengan cara memastikan bahwa program memiliki maksimum permintaan. Dengan kata lain cara sistem ini memastikan terlebih dahulu bahwa sistem akan selalu dalam kondisi aman. Baik mengadakan permintaan awal ataupun saat meminta permintaan sumber daya tambahan, sistem harus selalu berada dalam kondisi aman. Saat kondisi aman, maka suatu sistem dapat mengalokasikan sumber daya pada setiap proses (sampai pada batas maksimumnya) dengan urutan tertentu. Dengan mengenal arti dari kondisi aman ini, kita dapat membuat algoritma untuk menghindari *deadlock*. Idenya ialah dengan memastikan bahwa sistem selalu berada dalam kondisi aman. Dengan asumsi bahwa dalam kondisi tidak aman terkandung *deadlock*. Contoh penerapan algoritmanya ialah algoritma bankir (algoritma penjadualan).

Algoritma ini dapat digambarkan sebagai seorang bankir di kota kecil yang berurusan dengan kelompok orang yang meminta pinjaman. Jadi kepada siapa dia dapat memberikan pinjamannya, dan setiap pelanggan memberikan batas pinjaman maksimum kepada setiap peminjam dana.

Tentu saja si bankir tahu bahwa si peminjam tidak akan meminjam dana maksimum yang mereka butuhkan dalam waktu yang singkat melainkan bertahap. Jadi dana yang ia punya lebih sedikit dari batas maksimum yang dipinjamkan. Lalu ia memprioritaskan yang meminta dana lebih banyak, sedangkan yang lain disuruh menunggu hingga peminta dana yang lebih besar itu mengembalikan pinjaman berikut bunganya, baru setelah itu ia meminjamkan pada peminjam yang menunggu.

Jadi algoritma bankir ini mempertimbangkan apakah permintaan mereka itu sesuai dengan jumlah dana yang ia miliki, sekaligus memperkirakan jumlah dana yang mungkin diminta lagi. Jangan sampai ia sampai pada kondisi dimana dananya habis dantidak dapat meminjamkan uang lagi. Jika demikian maka akan terjadi kondisi *deadlock*. Agar kondisi aman, maka asumsi setiap pinjaman harus dikembalikan waktu yang tepat.

Secara umum algoritma bankir dapat dibagi menjadi empat struktur data:

1.Tersedia: jumlah sumber daya atau dana yang tersedia
2.Maksimum: jumlah sumber daya maksimum yang diminta oleh setiap proses
3.Alokasi: jumlah sumber daya yang dibutuhkan oleh setiap proses
4.Kebutuhan: sumber daya yang sedang dibutuhkan oleh setiap proses

D. Pencegahan *deadlock*

Caranya adalah dengan menanggulangi penyebab *deadlock* yaitu:

1. *Mutual Exclusive*
Kondisi ini tidak dapat dilarang, jika aksesnya perlu bersifat spesial untuk satu proses, maka hal ini harus didukung oleh kemampuan sistem operasi. Jadi diusahakan agar tidak mempergunakan kondisi spesial tersebut sehingga sedapat mungkin *deadlock* dapat dihindari. Cara lainnya adalah hindari pengalokasian sumber daya jika tidak benar-benar diperlukan, usahakan sesedikit mungkin proses mengklaim sumber daya, dan *spooling* sumber daya (misalnya *printer*).

2. *Hold and Wait*
Penanggulangan *deadlock* dari kondisi ini lebih baik dan menjanjikan, asalkan kita dapat menahan proses yang memegang sumber daya untuk tidak menunggu sumber daya lain, kita dapat mencegah *deadlock*. Caranya ialah dengan meminta semua sumber daya yang ia butuhkan sebelum proses berjalan. Tetapi masalahnya sebagian proses tidak mengetahui keperluannya sebelum ia berjalan. Jadi untuk mengatasi hal ini, kita dapat menggunakan algoritma bankir. Yang mengatur hal ini bisa sistem operasi atau pun sebuah protokol. Hasil yang dapat terjadi ialah sumber daya lebih dispesifikasi dan kelaparan sumber daya atau proses yang membutuhkan sumber daya yang banyak harus menunggu sekian lama untuk mendapat sumber daya yang dibutuhkan.

3. *No Preemption*
Jangan sampai ada *preemption* pada sumber daya yang telah dialokasikan. Untuk memastikan hal ini, kita dapat menggunakan protokol. Jadi jika sebuah proses meminta sumber daya yang tidak dapat dipenuhi saat itu juga, maka proses mengalami *preempted*. Atau dengan kata lain ada sumber daya dilepaskan dan diberikan ke proses yang menunggu, dan proses itu akan menunggu sampai kebutuhan sumber dayanya dipenuhi.

Atau kita harus mengecek sumber daya yang diinginkan oleh proses, dicek dahulu apakah tersedia. Jika cukup maka kita langsung alokasikan, sedangkan jika tidak tersedia maka kita melihat apakah ada proses lain yang menunggu sumber daya juga. Jika ada, maka kita ambil sumber daya dari proses yang menunggu tersebut dan memberikan pada proses yang meminta tersebut. Jika tidak tersedia juga, maka proses itu harus menunggu. Dalam menunggu, beberapa dari sumber dayanya dapat saja di *preempted*, tetapi jika ada proses yang memintanya. Cara ini efektif untuk proses yang menyimpan dalam memori atau register.

4. *Circular Wait*
Dapat ditangani oleh sebuah protokol yang menjaga agar sebuah proses tidak membuat lingkaran siklus yang dapat mengakibatkan *deadlock*.

Cara lain yang dapat digunakan selain keempat cara tersebut adalah dengan metode *two-phase locking* yaitu:

- Fase 1
  Proses mencoba mengunci semua *record* yang diperlukan secara satu per satu. Jika *record* yang diperlukan dikunci, mulai kembali. Jika fase ini berhasil, mulai ke fase 2
- Fase 2
  Lakukan *update* dan lepaskan kunci

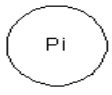
Cara ini mirip dengan meminta semua sumber sekaligus. Algoritma berfungsi jika *user* dapat mengatur berhenti dan *restart* program

5. Pendeteksian *Deadlock* Dengan Graf

5.1. Graf Alokasi Sumber Daya

5.1.1 Komponen

Pada dasarnya graf G=(V,E) terdiri dari dua komponen yaitu simpul dan sisi. Untuk graf alokasi sumber daya, simpul maupun sisinya dibedakan menjadi beberapa bagian.



Proses Pi

Simpul terdiri dari dua jenis yaitu:

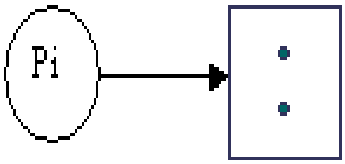
1. Proses P = {P0, P1, P2, P3, ... Pi, Pm}
   Terdiri dari semua proses yang ada di sistem. Untuk proses, simpulnya digambarkan sebagai lingkaran dengan nama prosesnya.

2. Sumber Daya R = {R0, R1, R2, R3,..Rj, Rn}
   Terdiri dari semua sumber daya yang ada di sistem. Untuk sumber daya, simpulnya digambarkan sebagai segi empat dengan instans yang dapat dialokasikan serta nama sumber dayanya. Dalam hal ini jumlah proses dan sumber daya tidak selalu sama.

Sumber Daya Rj dengan dua instans

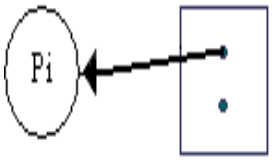
Sisi, E = {Pi->Rj,Rj->Pi} terdiri dari dua jenis yaitu:

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

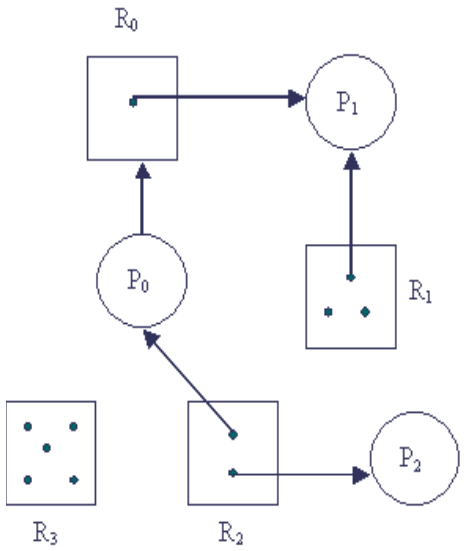


Proses Pi meminta sumber daya Rj

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



Sumber daya Rj mengalokasikan salah satu instansnya pada proses Pi



Graf alokasi sumber daya

Graf diatas terdiri dari 7 simpul:
V={P0, P1, P2, P3, R0, R1, R3},
dan 5 sisi:
E={P0->R0,R0->P1,R1->P1,R2->P0, R2->P2}

Graf diatas menunjukkan:

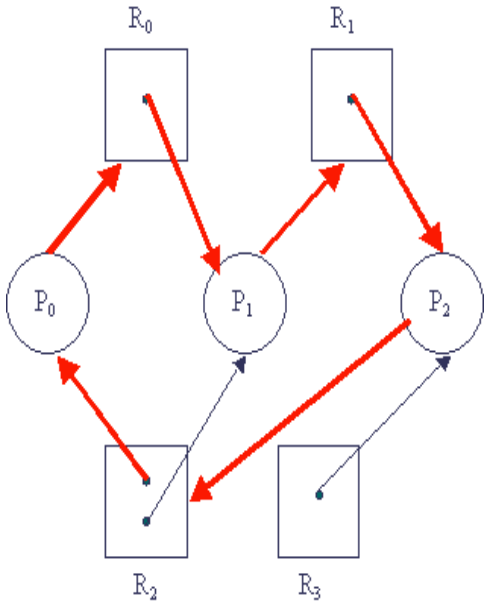
1.P0 meminta sumber daya dari R0.
2.R0 memberikan sumber dayanya kepada P1.
3.R1 memberikan salah satu instans sumber dayanya kepada P1.
4.R2 memberikan salah satu instans sumber dayanya kepada P0.
5.R2 memberikan salah satu instans sumber dayanya kepada P2.

Setelah suatu proses telah mendapatkan semua sumber daya yang diperlukan maka sumber daya tersebut dilepas dan dapat digunakan oleh proses lain.

5.1.2 Deteksi *Deadlock* Dengan Graf Alokasi Sumber Daya

Untuk mengetahui ada atau tidaknya *deadlock* dalam suatu graf dapat dilihat dari perputaran dan sumber daya yang dimilikinya. Jika tidak ada perputaran berarti tidak *deadlock*. Jika ada perputaran, ada potensi terjadi *deadlock*. Sumber daya dengan instan tunggal dan perputaran mengakibatkan *deadlock*. Berikut adalah gambar *deadlock graph* dan graf yang memiliki perputaran tetapi tidak terjadi *deadlock*.

Deadlock Graph 1



Dari gambar diatas terlihat bahwa ada perputaran yang memungkinkan terjadinya *deadlock* dan semua sumber daya memiliki satu instans kecuali sumber daya R2.

Graf diatas memiliki minimal dua perputaran:

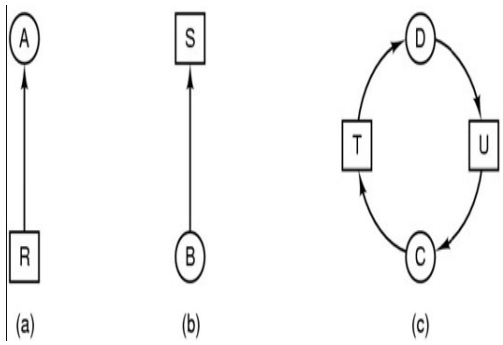
1. R2 -> P0 -> R0 -> P1 -> R1 -> P2 -> R2
2. R2 -> P1 -> R1 -> P2 -> R2

Gambar di atas menunjukkan beberapa hal sebagai berikut:

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.

Keterangan tersebut dapat mengakibatkan *deadlock* sebab P0 memerlukan sumber daya R0 untuk menyelesaikan prosesnya, sedangkan R0 dialokasikan untuk P1. Di lain pihak P1 memerlukan sumber daya R1 sedangkan R1 dialokasikan untuk P2. P2 memerlukan sumber daya R2 akan tetapi R2 mengalokasikan sumber dayanya pada R3.

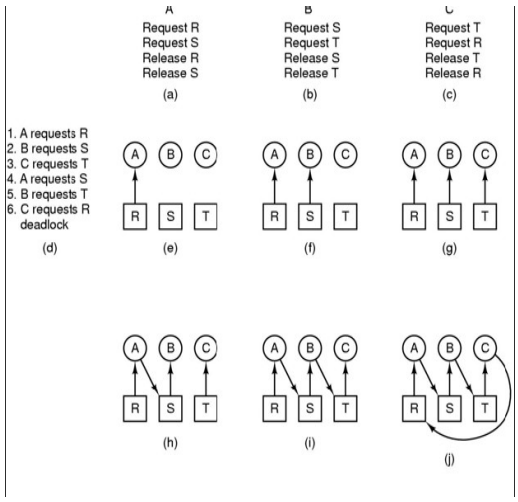
Deadlock Graph 2



Gambar diatas menunjukkan:

1. Sumber daya R dialokasikan ke proses A
2. Proses B meminta atau menunggu sumber daya S
3. Proses C dan D *deadlock* terhadap sumber daya T dan U

Deadlock Graph 3

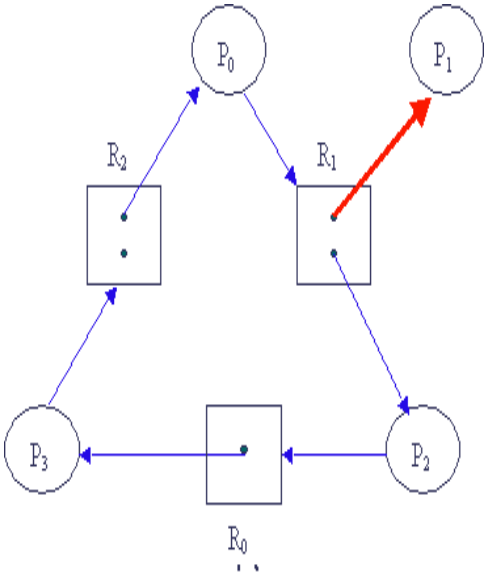


Gambar diatas menggambarkan *circular wait request* yang tidak akan pernah berhenti.

Dengan kata lain, tidak ada satu pun dari proses tersebut yang dapat menyelesaikan tugasnya sebab sumber daya yang diperlukan sedang digunakan oleh proses lain. Sedangkan proses lain juga memerlukan sumber daya lain. Semua sumber daya yang diperlukan oleh suatu proses tidak dapat dipenuhi sehingga proses tersebut tidak dapat melepaskan sumber daya yang telah dialokasikan kepadanya. Dan terjadi proses tunggu-menunggu antarproses yang tidak dapat

berakhir. Inilah yang dinamakan *deadlock*.
Jika ada perputaran, ada potensi juga untuk tidak terjadi *deadlock*. Ini bisa saja terjadi jika proses yang satu dengan yang lain tidak saling menunggu seperti gambar diatas.

Graf tanpa deadlock



Gambar diatas adalah graf yang memiliki perputaran tetapi *deadlock* tidak terjadi. Pada gambar diatas, graf memiliki 1 perputaran yaitu: **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.

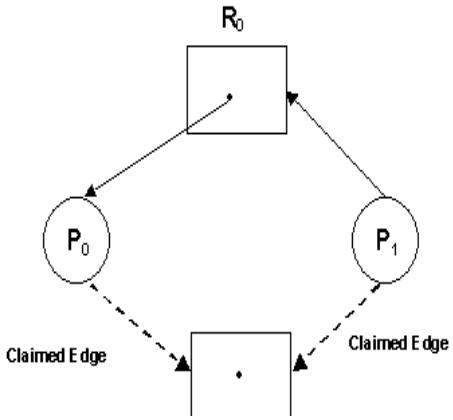
Hal ini tidak menyebabkan *deadlock* walaupun ada perputaran sebab semua sumber daya yang diperlukan P1 dapat terpenuhi sehingga P1 dapat melepaskan semua sumber dayanya dan sumber daya tersebut dapat digunakan oleh proses lain.

5.1.3 Algoritma Graf Alokasi Sumber Daya Untuk Mencegah *Deadlock*

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

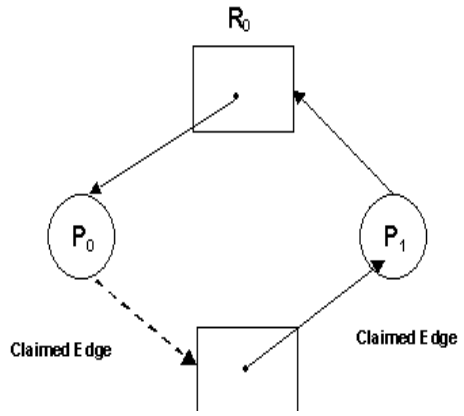
Claimed edge Pi -> Rj berarti bahwa proses Pi akan meminta sumber daya Rj pada suatu waktu. *Claimed edge* sebenarnya merupakan sisi permintaan yang digambarkan sebagai garis putus-putus. Ketika proses Pi memerlukan sumber daya Rj, *claimed edge* diubah menjadi sisi permintaan. Dan setelah proses Pi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

Dengan algoritma ini bentuk perputaran pada graf tidak dapat terjadi. Sebab untuk setiap perubahan yang terjadi akan diperiksa dengan algoritma deteksi perputaran. Algoritma ini memerlukan waktu n^2 dalam mendeteksi perputaran dimana n adalah jumlah proses dalam sistem. 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.



Graf alokasi sumber daya dalam status aman

Pada saat ini R1 sedang tidak mengalokasikan sumber dayanya, sehingga P1 dapat memperoleh sumber daya R1. Namun, jika *claimed edge* diubah menjadi sisi permintaan dan kemudian diubah menjadi sisi alokasi, hal ini dapat menyebabkan terjadinya perputaran.



Graf alokasi sumber daya status tidak aman

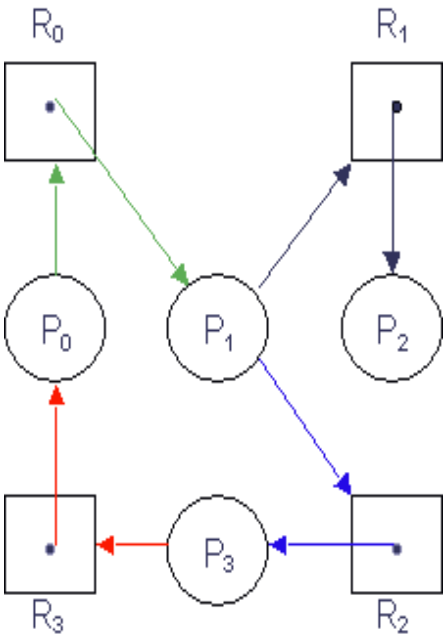
5.2 Graf Tunggu

5.2.1 Komponen

Komponen pada graf tunggu tidak jauh berbeda dengan komponen pada graf alokasi. Bedanya sumber daya pada graf tunggu hanya mempunyai satu instans.

5.2.2 Deteksi *Deadlock* Dengan Graf Tunggu

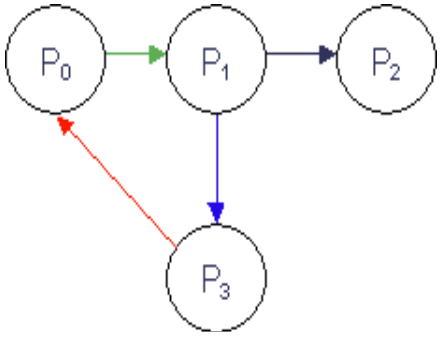
Jika semua sumber daya hanya memiliki satu instans, *deadlock* dapat dideteksi dengan mengubah graf alokasi sumber daya menjadi graf tunggu.



Graf alokasi sumber daya

Caranya sebagai berikut:

1.Cari sumber daya Rm yang memberikan instansnya pada Pi dan Pj yang meminta sumber daya pada Rm.
2.Hilangkan sumber daya Rm dan hubungkan sisi Pi dan Pj dengan arah yang bersesuaian yaitu Pj->Pi.
3.Lihat apakah terdapat perputaran pada graf tunggu. Deadlock terjadi jika dan hanya jika pada graf tunggu terdapat perputaran.



Graf Tunggu

5.2.3 Algoritma Graf Tunggu Untuk Mencegah Deadlock

Untuk mendeteksi *deadlock*, sistem perlu membuat graf tunggu dan secara berkala memeriksa apakah ada perputaran atau tidak. Untuk mendeteksi adanya perputaran diperlukan operasi sebanyak n^2, dimana n adalah jumlah simpul dalam graf alokasi sumber daya.

6. Kesimpulan

Kesimpulan yang dapat diambil dari komponen dasar sistem operasi, *deadlock*, dan implementasi graf untuk pendeteksian *deadlock* pada sistem operasi adalah:

- Komponen-komponen dasar sistem operasi ada delapan yaitu manajemen proses, nanajemen memori utama, manajemen berkas, manajemen sistem M/K, manajemen penyimpanan sekunder, sistem proteksi, jaringan, dan *command-interpreter system*. Dan ternyata semua komponen dasar tersebut memakai implementasi dari matematika diskrit diantaranya graf, logika, pohon, aljabar boolean, dan kompleksitas algoritma.

•Kondisi yang dapat menyebabkan *deadlock* ada empat yaitu *mutual exclusive*, *hold and wait*, *no preemption*, dan *circular wait*.

•Cara untuk menanggulangi *deadlock* ada empat yaitu mengabaikan masalah *deadlock*, mendeteksi-memperbaiki, menghindari *deadlock*, dan mencegah *deadlock* terjadi dengan antisipasi dari kondisi yang akan menyebabkan *deadlock*.

•Cara pendeteksian *deadlock* dengan implementasi graf ada dua yaitu dengan graf alokasi sumber daya dan graf tunggu.

•Untuk mengetahui ada atau tidaknya *deadlock* dalam suatu graf alokasi sumber daya dapat dilihat dari perputaran dan sumber daya yang dimilikinya. Jika tidak ada perputaran berarti tidak *deadlock*. Jika ada perputaran, ada potensi terjadi *deadlock*. Sumber daya dengan instans tunggal dan perputaran pasti akan mengakibatkan *deadlock*.

•Pada graf tunggu, *deadlock* terjadi jika dan hanya jika pada graf tersebut ada perputaran. Untuk mendeteksi adanya perputaran diperlukan operasi sebanyak n^2, dimana n adalah jumlah simpul dalam graf alokasi sumber daya.

DAFTAR PUSTAKA

[1] Masyarakat Digital Gotong Royong. (2005). *Pengantar Sistem Operasi Komputer Plus Ilustrasi Kernel Linux*. http://bebas.vlsm.org/v06/Kuliah/SistemOperasi/BUKU/SistemOperasi.pdf. Tanggal akses: 25 Desember 2006 pukul 11.45.

[2] Zubir, Henny Y.. (2003). *IF3191-Manajemen Proses Deadlock*. http://kur2003.if.itb.ac.id/file/CN-IF3191-Deadlock.pdf. Tanggal akses: 30 Desember 2006 pukul 14.30.

[3] Munir, Rinaldi. (2004). *Diktat Kuliah IF2151 Matematika Diskrit Edisi Keempat*. Departemen Teknik Informatika, Institut Teknologi Bandung.

[4] Gabungan Kelompok Kerja 21–28 Semester Genap 2002/2003 dan 41–49 Semester Ganjil 2003/2004 Mata Kuliah Sistem Operasi Fakultas Ilmu Komputer Universitas Indonesia. (2003). *Bahan Kuliah IKI-20230 Sistem Operasi*. http://ftp.ui.edu/bebas/v06/Kuliah/SistemOperasi/2003/44/produk/SistemOperasi.rtf. Tanggal akses: 27 Desember 2006 pukul 09.30.

[5] Ruseffendi, E.T.. (1976). *Dasar-dasar Matematika Modern Edisi Kedua*. Jurusan Matematika, IKIP Bandung.