PENGENALAN KONTROL INPUT/OUTPUT

DEFINISI DAN PERSYARATAN KONTROL I/O

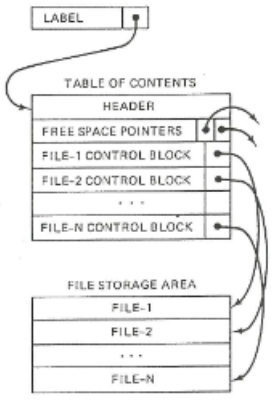
Sebuah sistem kontrol I/O bertujuan untuk memberikan bantuan kepada user untuk memungkinkan mereka mengakses berkas, tanpa memperhatikan detail dari karakteristik dan waktu penyimpanan. Kontrol I/O menyangkut manajemen berkas dan peralatan manajemen yang merupakan bagian dari sistem operasi.

Tugas dari sistem kontrol I/O adalah:

- Memelihara direktori dari berkas dan lokasi informasi.
- Menentukan jalan (*pathway*) bagi aliran data antara *main memory* dan alat penyimpanan sekunder.
- Mengkoordinasi komunikasi antara CPU dan alat penyimpanan sekunder.
- Menyiapkan berkas penggunaan input atau output
- Mengatur berkas, bila pengunaan input atau output telah selesai.

DIREKTORI BERKAS DAN KONTROL INFORMASI

Sebelum berkas dapat diakses oleh sebuah program, sistem operasi harus mengetahui pada alat penyimpanan sekunder yang mana berkas tersebut berada.



Gambar 1. Struktur Direktori Berkas

Direktori yang diperlihatkan pada gambar 1. tersebut adalah untuk satu unit (mis. *disk pack* atau *tape reel*) dari penyimpanan sekunder.

Labelnya berisi identifikasi informasi, akses kontrol informasi, dan sebuah *pointer* yang menunjuk ke isi tabel, yang berisi kontrol blok untuk setiap berkas pada unit tersebut.

Sebuah kontrol blok berisi informasi tentang nama berkas, atributnya (seperti panjang record, ukuran blok, organisasi berkas) dan batasannya pada media penyimpanan. Sebuah kontrol blok menunjukkan awal dari berkas yang bersangkutan. Jadi bila sebuah berkas dicari, isi tabel dari unit yang dimaksud diperiksa untuk menemukan berkas pada media penyimpanan.

KONTROL PERALATAN

Aktifitas I/O terutama mencakup perpindahan data antara *main memory* dengan alat penyimpanan sekunder atau alat I/O, seperti printer, terminal, dan *card reader/punch*. Operasi I/O memerlukan dukungan kontrol alat secara terinci.

Contoh:
Misalkan direktori berkas sudah diberitahu lokasi berkas yang diminta pada alat penyimpanan sekunder atau peralatan I/O sudah ditentukan. Supaya dapat menulis (*Write*) pada alat tersebut atau membaca (*Read*) dari alat tersebut.

Maka:
Jalur (*pathway*) antara memori utama dan peralatan harus sudah ditentukan dan juga harus ditentukan komponen-komponen yang dibutuhkan dari jalur tersebut (termasuk peralatan yang dituju) dan siap untuk digunakan.

CHANNEL

Pada kebanyakan sistem komputer, CPU tidak dibebani menangani tugas yang berhubungan dengan I/O.
Tetapi tanggung jawab untuk kontrol peralatan diserahkan pada prosesor I/O, yang dikenal sebagai saluran I/O (I/O *channel*).

Saluran I/O itu sendiri merupakan prosesor yang sudah di program. Program-program yang di *execute* ini disebut *channel* program. Channel program ini menentukan operasi, yang diperlukan untuk akses peralatan dan mengontrol jalur data (data *pathway*).

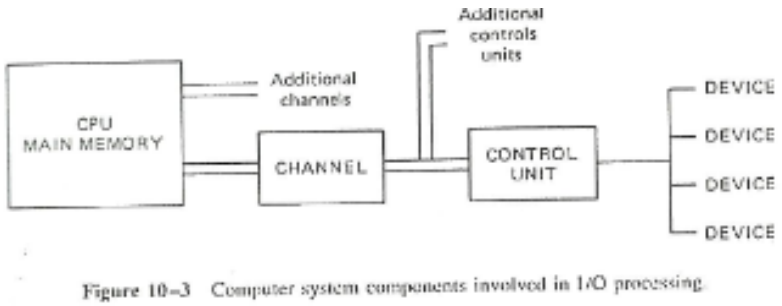


Figure 10–3 Computer system components involved in I/O processing

Gambar 2. Komponen Sistem Komputer dalam I/O Processing

Saluran I/O ini diperintah oleh sistem operasi dan kemudian oleh CPU. Saluran I/O ini tidak mengendalikan alat penyimpanan secara langsung. Tetapi saluran I/O ini menetapkan satu atau lebih unit alat pengontrol. Susunan ini penting untuk menyederhanakan *channel program*.

Sebelum opersi I/O dapat dimulai, jalur antara memori utama dengan peralatan harus sudah ditentukan. Jika saluran, unit pengontrol, atau peralatan yang dituju sedang sibuk, maka kontruksi dari *pathway* harus menunggu. Untuk menghindari menunggu, sistem komputer dilengkapi dengan beberapa saluran dan unit pengontrol.

MACAM-MACAM CHANNEL

- **Selector Channel**

Dapat mengatur aliran data antara memori utama dengan sebuah peralatan pada saat tersebut. Karena saluran merupakan prosesor-prosesor yang cepat, maka saluran selektor biasanya hanya menggunakan peralatan I/O dengan kecepatan tinggi, seperti disk. Penggunaan peralatan dengan kecepatan rendah, misal *card reader*.

- **Multiplexor Channel**

Dapat mengatur aliran data antara memori utama dengan beberapa perlatan. Saluran multiplexor lebih efektif bila menggunakan peralatan dengan kecepatan rendah, dibandingkan dengan selector channel. Dengan saluran multiplexor, beberapa peralatan dapat diaktifkan secara serentak, tetapi saluran harus melengkapi saluran program untuk satu peralatan sebelum memulai dengan saluran program lain.

- **Block Multiplexor Channel**

Mengatur aliran data ke berbagai peralatan. Block Multiplexor Channel dapat mengeksekusi satu intruksi dari saluran program untuk satu peralatan, kemudian dapat mengalihkan intruksi-intruksi dari saluran program itu ke peralatan yang lain.

MACAM-MACAM DEVICE

Dedicated Device
Digunakan untuk pengaksesan oleh satu orang pada setiap saat.
Contoh: Terminal

Shared Device
Digunakan untuk pengaksesan oleh banyak pemakai secara bersamaan.
Contoh: Disk

Aktifitas I/O untuk *shared device* adalah sangat kompleks dibanding aktifitas I/O pada *dedicated device*. Dua fungsi yang sangat penting dari *shared device* adalah alokasi tempat dan pemberian akses yang tepat.

Jenis ketiga dari peralatan adalah suatu unit dimana pemakai mengiginkan pemakaian secara bersama, tetapi sesungguhnya tidak cocok digunakan untuk penggunaan *concurrent*.

Contoh:

Berbagai program ditujukan ke printer secara bersamaan. Sistem pengontrol I/O menggunakan peralatan virtual untuk mengalamatkan jenis peralatan yang dibutuhkan. Program tersebut menulis ke alat pencetak (printer), tetapi sesungguhnya di tulis ke disk dahulu. Setelah berkas output diakhiri oleh program, berkas tersebut masuk dalalm antrian, kemudian ditulis seluruhnya ke printer.

Peralatan virtual I/O ini disebut *spooling*.
Maksud dari sistem *spooling* adalah untuk memberikan pemakaian peralatan secara bersamaan. Hampir semua komputer multi user menggunakan *spooling*.

AKTIFITAS SALURAN

Tujuan dari saluran I/O adalah sebagai perantara antara CPU-*main memory* dengan unit pengontrol penyimpan. CPU berkomunikasi dengan saluran melalui beberapa perintah yang sederhana.

Beberapa saluran akan memberi perintah:

- Test I/O, untuk menentukan apakah jalur (*pathway*) yang menuju peralatan sedang sibuk.
- Start I/O, pada peralatan tertentu.
- Halt I/O, pada peralatan tertentu.

Saluran biasanya berkomunikasi dengan CPU melalui cara interupsi. Interupsi akan terjadi, jika keadaan *error* terdeteksi, misalnya instruksi CPU yang salah atau jika aktifitas I/O telah diakhiri.

Jika interupsi terjadi, kontrol akan bercabang melalui rutin pengendali interupsi (*interrupt-handler routine*), dimana kontrol akan menentukan penyebab dari interupsi, melakukan kegiatan yang tepat, kemudian mengembalikan kontrol pada pemanggil (*caller*).

PROSES PEMBACAAN

Jika sebuah program membutuhkan READ dari suatu berkas, maka langkah-langkah berikut, yang tampak pada *gambar 3* akan terjadi.

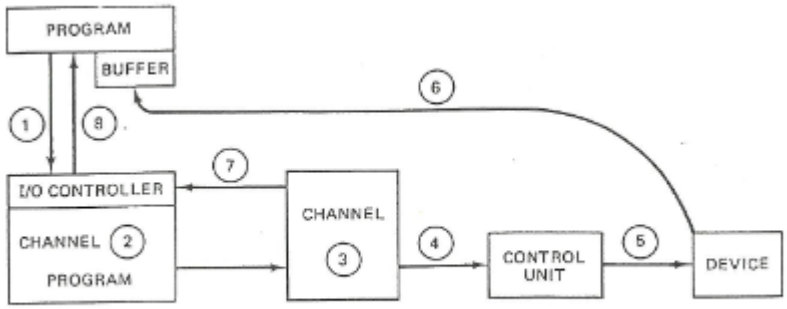


Figure 10–4 Sequence of events in processing a file READ.

Gambar 3. Urutan kejadian dalam proses file READ

Langkah-langkah tersebut adalah:

1. Program mengeluarkan sebuah READ, yang menginterupsi pengontrol I/O.
2. Pengontrol I/O membuat sebuah saluran program pada memori utama.
3. Saluran program dibaca dan di eksekusi oleh pemanggil saluran.
4. Sinyal yang tepat akan di transmisikan ke pemanggil unit kontrol.
5. Sinyal ini diterjemahkan oleh unit kontrol dan digunakan untuk mengontrol peralatan operasi untuk membaca data yang diminta.
6. Data yang diminta akan mengalir dari peralatan sepanjang jalur (*pathway*) ke daerah penampung berkas (*file buffer area*) dalam ruang memori utama.
7. Interupsi yang dikeluarkan oleh saluran, digunakan untuk meneruskan sinyal pada waktu eksekusi program.
8. Kontrol kembali ke program.

CATATAN:

Data dibaca kedalam *buffer*, dimana *buffer* ini merupakan suatu tempat pada memori utama yang disediakan untuk menampung data. Jika *buffer* penuh, program akan segera menggunakan data tersebut.

Rangkaian sekilas dari peristiwa yang terjadi, yaitu ketika sebuah program meminta WRITE pada sebuah berkas, akses I/O akan digiatkan. Peristiwa ini harus terjadi untuk setiap intruksi READ dan WRITE yang ditujukan pada peralatan.

BLOCKING RECORD

Teknik yang sering digunakan untuk mengurangi peralatan program akses adalah *block record*, sedemikian rupa, sehingga beberapa record akan dibaca/ditulis dalam suatu akses tunggal pada peralatan. Jika ada n record per blok, maka hanya setiap READ ke n yang akan diakses oleh suatu program pada suatu peralatan. Biasanya intruksi READ diubah menjadi intruksi GET oleh sistem kontrol I/O. Kemudian suatu channel program menerjemahkan instruksi GET keperalatan READ. Jika *buffer* kosong, mengakibatkan buffer akan diisi.

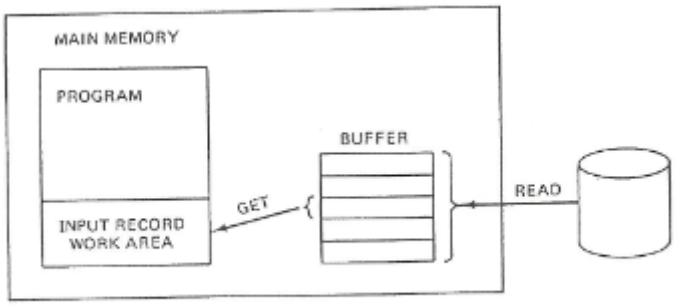


Figure 10–5 Device access with blocked records.

Gambar 4. Peralatan Akses dengan Block Record

MANAJEMEN BUFFER

- **Single Buffering**

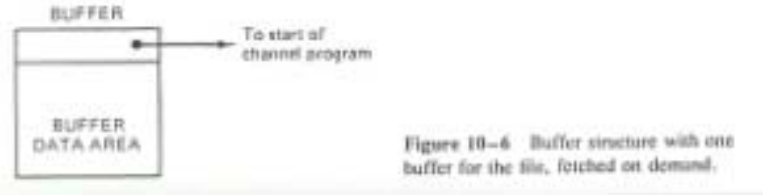


Figure 10–6 Buffer structure with one buffer for the file, fetched on demand.

Gambar 5. Struktur Buffer dengan satu buffer file

Pada gambar 5. menunjukkan struktur data dari *buffer* dalam bentuk yang sederhana, yang terdiri dari satu record per-*block* dan satu *buffer* per berkas, dimana *buffer* ini berfungsi mengisikan permintaan dari sebuah program. Struktur *buffer* ini berisi sebuah *pointer* pada alamat awal dan *channel* program untuk berkas.

Struktur dasar dari channel program untuk mengisi buffer adalah:

- Tunggu intruksi READ dari program
- Memberitahukan intruksi start I/O ke unit kontrol
- Tunggu hingga *buffer* dikosongkan
- Memberitahukan interupsi pada program, sehingga dapat mulai membaca dari *buffer*.

Masalah yang timbul disini adalah pemakai program menggangur pada saat menunggu *buffer* diisi.

- **Anticipatory Buffering**

Pendekatan lain yang dapat menghilangkan beberapa hal yang mungkin untuk menunggu CPU adalah dengan menggunakan *Anticipatory Buffering*.

Dengan *anticipatory buffering*, sistem kontrol I/O akan berusaha mendahulukan kebutuhan program akan data. Diusahakan agar *buffer* selalu penuh. *Channel* selalu menguji *flag* ini. Jika *buffer* mendekati kosong, karena pemakai program telah membaca isinya, maka *flag* itu akan di-*reset* dan *channel program* akan meng-*initates* pengisian kembali *buffer*.

Struktur dasar *channel program* untuk mengisi sebuah *buffer* dengan *anticipatory buffer* diperlihatkan pada gambar 6.

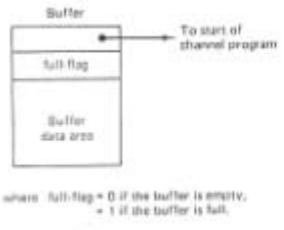


Figure 10–7 Buffer structure with one buffer per file and prefetching.

Gambar 6. Struktur buffer dengan satu buffer per file

```
loop  :  If full-flag = 1 go to loop
         ISSUE START - I/O Command to control unit
         wait while buffer is being filled
         full-flag:= 1
         go to loop
```

Rutin pelengkap untuk mengosongkan *buffer* kedalam tempat kerja record program selalu diperlukan. Rutin pengisian dan pengosongan

buffer selalu berpasangan: salah satu dari pasangan tersebut merepresentasikan sebuah PRODUCER, sedangkan pasangan yang lain bertindak sebagai CONSUMER.

Untuk berkas input, *producer* adalah *channel program* dan *consumer* adalah pemakai program.
Untuk berkas output, *producer* adalah pemakai program dan *consumer* adalah *channel program*.

Rutin consumer yang mendampingi rutin producer di atas adalah:

```
wait :  If full-flag = 0 go to wait
        read the buffer contents into the record work area
        full-flag:= 0
        go to wait
```

Keadaan awal, full-flag = 0, mengakibatkan channel program mengisi buffer.

- **Double Buffering**

Untuk mengurangi kemungkinan dari program menunggu, maka *double buffer* dapat digunakan. Dua dari tempat *buffer* yang ada, hanya satu yang ditetapkan untuk berkas.

Ide dasar dari *double buffering* adalah jika *consumer* mengosongkan salah satu buffer, maka producer dapat mengisikan kedalam buffer yang lain, pada saat *buffer* pertama sudah kosong, maka *buffer* yang kedua harus dalam keadaan penuh. Kemudian *consumer* dapat mengkosongkan *buffer* yang kedua, pada saat *producer* mengisi *buffer* yang pertama, demikian seterusnya.

Struktur *buffer* untuk *double buffering* terdiri dari sebuah *pointer* yang menunjuk ke *buffer* berikutnya.

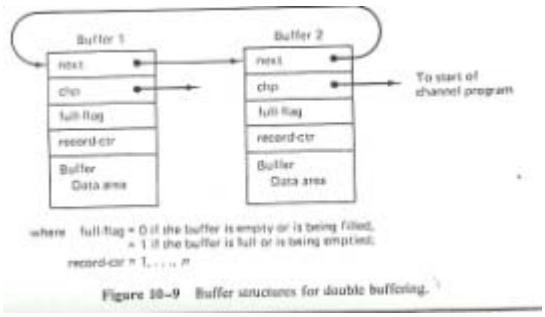


Figure 10–9 Buffer structures for double buffering.

Gambar 7. Struktur buffer untuk dua buffer

- **Three Buffers**

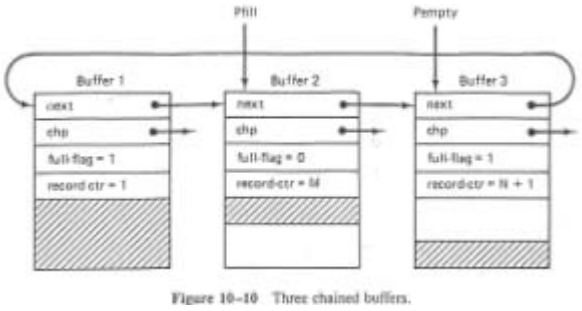


Figure 10–10 Three chained buffers.

Gambar 8. Three Chained Buffer

pfill : yang menunjukkan *buffer* berikutnya akan diisi atau sedang diisi

pempty : yang menunjukkan buffer berikutnya akan dikosongkan atau sedang dikosongkan.

Keadaan ini dapat dilihat sebagai berikut:

Buffer 1 penuh
Buffer 2 sedang diisi
Buffer 3 sedang dikosongkan, record ke m di dalam buffer akan dibaca kedalam tempat kerja record berikutnya.
Buffer berikutnya yang akan kosong adalah buffer 1
Buffer berikutnya yang akan di isi adalah buffer 3

Pada pendekatan ini dianggap jika suatu *buffer* kosong, *producer* akan bergerak mengisinya.

Contoh:

Jika setelah *buffer* 3 diisi pada keadaan di atas, *consumer* tetap akan mengosongkan *buffer* 1, kemudian *producer* mengisi *buffer* 2. Selanjutnya *consumer* akan menunggu.

Jika *producer* secara konsisten mengisi *buffer* pada kecepatan yang lebih rendah dibanding kecepatan pada saat *consumer* mengosongkannya, maka *consumer* akan menunggu.

Keuntungan menggunakan lebih dari satu *buffer* adalah kemampuannya untuk saling melengkapi operasi pengisian dan pengosongan, dengan demikian mengurangi waktu tunggu.