

Bab II

LANDASAN TEORI

2.1 *File Transfer Protocol*

File Transfer Protocol (FTP) didefinisikan sebagai sebuah protokol untuk mengirim dan menerima *file* antara *host* (dalam *ARPANET*), dengan fungsi utama dari *FTP* adalah mengirim dan menerima *file* dengan efisien dan handal antara *host* dan mengizinkan penggunaan yang nyaman dari kemampuan untuk penyimpanan *file* secara *remote*.

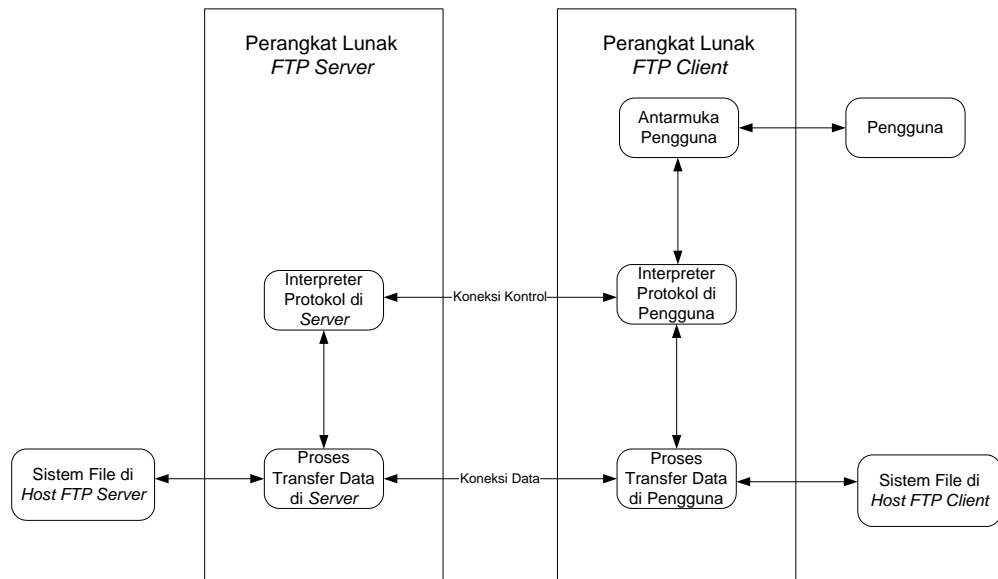
Tujuan dari *FTP* [RFC959] adalah:

1. Untuk mempromosikan share *file* (yang dapat berupa program komputer dan/atau data).
2. Untuk memacu penggunaan komputer *remote* secara tidak langsung atau implisit (via perangkat lunak).
3. Untuk melindungi pengguna dari variasi dalam sistem penyimpanan *file* diantara *host*.
4. Untuk mengirim dan menerima data secara handal dan efisien.

FTP adalah bagian dari protokol TCP/IP dan menggunakan TCP sebagai *transport layer*.

2.1.1 Model Layanan FTP

Gambar II-1 merupakan model layanan *FTP*.



Gambar II-2 Model dari penggunaan *FTP*

Pada Gambar II-1, pengguna akan menggunakan layanan *FTP* di *FS* dengan menggunakan antarmuka pengguna yang terdapat di perangkat lunak *FC*. Dalam berkomunikasi dengan perangkat lunak *FS*, perangkat lunak *FC* menggunakan dua proses, yaitu *interpreter* protokol dan proses transfer data. Proses ini juga terdapat di perangkat lunak *FS*. *Interpreter* protokol menangani pemrosesan perintah *FTP* antara *FS* dan *FC*, sedangkan proses transfer data menangani transfer data antara *FS* dan *FC*. Masing-masing proses ini berinteraksi dengan jenis proses yang sama di perangkat lunak *FS* dan *FC*. Koneksi antara proses *interpreter* di *FS* dan *FC* dinamakan koneksi kontrol, karena koneksi ini menjembatani proses *interpreter*, yang menangani kontrol dari aktivitas yang berlangsung antara *FS* dan *FC*. Sedangkan Koneksi antara proses transfer data di *FS* dan *FC* dinamakan koneksi data, karena koneksi ini menjembatani proses transfer data, yang menangani transfer data antara *FS* dan *FC*.

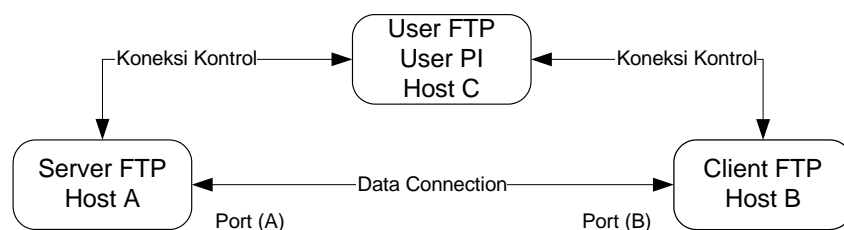
Protokol pada koneksi kontrol menggunakan protokol telnet. Pada saat inisiasi dari pengguna, perintah *FTP* standar dihasilkan oleh *interpreter* dari protokol pengguna dan ditransmisikan ke proses *server* melalui koneksi kontrol. Jawaban *FTP* standar

dikirimkan dari *interpreter* dari protokol server ke *interpreter* dari protokol pengguna melalui koneksi kontrol sebagai respon terhadap perintah-perintah.

Perintah-perintah *FTP* menspesifikasikan parameter-parameter untuk koneksi data (*port* data, mode transfer, tipe representasi, dan struktur) dan sifat dasar dari operasi pada sistem *file* (menyimpan, mengambil, menambah (*append*), menghapus). Proses transfer data di pengguna harus mendengarkan (*listen*) pada *port* data yang dispesifikasikan, dan *FS* menginisiasi koneksi data dan transfer data berdasarkan parameter-parameter yang telah dispesifikasikan. Harus dicatat bahwa *port* data tidak harus berada pada *host* yang sama dengan *host* yang menginisiasi perintah *FTP* via koneksi kontrol, tetapi pengguna atau proses *FC* harus memastikan bahwa terdapat sebuah *listen* pada *port* yang dispesifikasikan. Juga harus dicatat bahwa koneksi data mungkin digunakan untuk pengiriman dan penerimaan secara simultan.

Protokol memerlukan koneksi-koneksi kontrol yang terbuka sewaktu proses transfer data. Merupakan tanggung jawab pengguna untuk meminta penutupan dari koneksi-koneksi kontrol setelah selesai menggunakan layanan *FTP*, sementara *FS* yang melakukan aksi. *FS* dapat membatalkan transfer data jika koneksi-koneksi kontrol ditutup tanpa perintah.

Pada situasi lain, seorang pengguna mungkin menginginkan untuk mentransfer *file* antara dua *host*, dimana keduanya bukan *host* (lokal) pengguna tersebut. Pengguna membuat koneksi-koneksi kontrol ke dua *FS* dan kemudian mengatur untuk sebuah koneksi data diantara keduanya. Dalam hal ini, informasi kontrol dilewatkan ke *interpreter* protokol pengguna, tetapi data ditransfer antar proses-proses transfer data di *FS*. Berikut ini adalah sebuah model dari interaksi *server-server* ini.



Gambar II-3 Model Interaksi *Server-Server*

Protokol memerlukan koneksi-koneksi kontrol terbuka sewaktu transfer data dalam proses. Merupakan tanggung jawab pengguna untuk meminta penutupan dari koneksi-koneksi kontrol setelah selesai menggunakan layanan *FTP*, sementara *FS* lah yang melakukan aksi. *FS* dapat membatalkan transfer data jika koneksi-koneksi kontrol ditutup tanpa perintah.

2.1.2 Hubungan antara *FTP* dan Telnet

Dalam penggunaannya, *FTP* menggunakan protokol lain untuk salah satu koneksinya. Protokol telnet digunakan pada koneksi kontrol. Terdapat dua cara untuk mengimplementasikan hal ini: [RFC959]

1. Pertama, *interpreter* protokol pengguna atau *interpreter* protokol *server* dapat mengimplementasikan aturan-aturan dari protokol telnet secara langsung di prosedur mereka sendiri; atau,
2. Kedua, *interpreter* protokol pengguna atau *interpreter* protokol *server* dapat menggunakan modul telnet yang telah ada di sistem.

Pada pendekatan pertama, keuntungannya terdapat efisiensi eksekusi dan independensi implementasi dari perangkat lunak *FTP*. Sedangkan dari pendekatan kedua, didapat kemudahan dari implementasi, penggunaan ulang kode, dan pemrograman modular. Pada pelaksanaannya, *FTP* tergantung dari bagian yang sangat kecil dari protokol *telnet*, jadi pendekatan pertama tidaklah harus melibatkan banyak kode.

File ditransfer hanya melalui koneksi data. Koneksi kontrol digunakan untuk mentransfer perintah-perintah. Beberapa perintah berhubungan dengan transfer data antara *host*. Perintah-perintah transfer data termasuk perintah *MODE* yang menspesifikasikan bagaimana bit dari data akan ditransmisikan, dan perintah *STRU* (*STRU*cture) dan *TYPE*, yang digunakan untuk mendefinisikan bagaimana data akan direpresentasikan. Transmisi dan representasi secara mendasar berbeda tetapi mode transmisi stream tergantung pada atribut struktur *file* dan jika mode transmisi “*compressed*” digunakan, sifat dari *byte* pengisi tergantung pada tipe representasi.

Representasi data ditangani pada *FTP* dengan seorang pengguna menspesifikasikan sebuah tipe representasi. Tipe ini dapat secara implisit (seperti dalam ASCII atau EBCDIC) atau secara eksplisit (seperti dalam *local byte*) mendefinisikan sebuah ukuran *byte* untuk interpretasi yang akan diacu sebagai “ukuran *byte* logik”. Harus diperhatikan bahwa hal ini tidak berhubungan dengan ukuran *byte* yang digunakan untuk transmisi melalui koneksi data, yang disebut “ukuran *byte* transfer”, dan keduanya seharusnya tidak dibingungkan. Sebagai contoh, NVT-ASCII mempunyai ukuran *byte* logik delapan *bit*. Jika tipenya adalah *Local byte*, maka perintah TYPE mempunyai paramater kedua wajib yang menspesifikasikan ukuran *byte* logik. Ukuran *byte* transfer selalu delapan *bit*.

2.1.3 Fungsi Transfer *File*

Jalur komunikasi dari *interpreter* protokol pengguna ke *interpreter* protokol *server* dibangun sebagai sebuah koneksi TCP dari pengguna ke *port server* standar. Interpreter protokol pengguna bertanggungjawab untuk mengirimkan perintah *FTP* dan menginterpretasikan balasan yang diterima; *interpreter* protokol *server* menginterpretasikan perintah-perintah, mengirimkan balasan, dan mengarahkan proses transfer datanya untuk membuat koneksi data dan mentransfer data. Jika pihak kedua dalam transfer data (proses transfer pasif) adalah proses transfer data pengguna, maka ia dikendalikan melalui protokol internal dari *host FTP* pengguna; jika ia adalah sebuah proses transfer data *server* kedua, maka ia dikendalikan dengan *interpreter* protokolnya sendiri dengan perintah dari *interpreter* protokol pengguna.

2.1.4 Mode Transfer

Mode transfer mendefinisikan mode yang digunakan FTP dalam melakukan transfer data, berkaitan dengan inisiasi koneksi kontrol. Mode transfer terbagi tiga, yaitu :
[RFC959]

1. Mode Aktif

Mode Aktif adalah mode dimana *client*, sebagai peminta layanan *FTP*, yang secara aktif mendefinisikan jalur transfer data, dan karena itu *client* yang

menginisiasi koneksi data, dengan melakukan *listen* terlebih dahulu. Pada mode aktif, *FC* membuka (*listen*) sebuah *port* acak (> 1023), mengirimkan nomor *port* tersebut ke *FS* melalui koneksi kontrol, dan menunggu sebuah permintaan koneksi dari *FS*. Ketika *FS* menginisiasi koneksi data ke *FC*, *FS* akan melakukan *bind port* asal ke *port* 20 pada *FS*.

Untuk menggunakan mode aktif, *FC* akan mengirimkan perintah *PORT*, dengan *IP* dan *port* sebagai argumennya. Format dari *IP* dan *port* tersebut adalah 'h1,h2,h3,h4,p1,p2'. Setiap *field* adalah sebuah representasi desimal dari delapan *bit* dari *IP host*, diikuti dengan nomor *port*. Sebagai contoh, sebuah *FC* dengan *IP* 192.168.0.1, melakukan '*listen*' pada 1025 untuk koneksi data, maka perintah *PORT* yang dikeluarkan adalah '*PORT* 192,168,0,1,4,1'. *Field port* diinterpretasikan sebagai $p1 \times 256 + p2 = port$, atau, dalam contoh ini adalah, $4 \times 256 + 1 = 1025$.

2. Mode Pasif

Mode pasif adalah mode dimana *FC*, sebagai peminta layanan FTP, bertindak pasif dengan menunggu *FS* untuk menginisiasi koneksi data. Pada mode pasif, *FS* membuka sebuah *port* acak (> 1023), mengirimkan *IP Address* dan nomor *port* tersebut ke *FC* melalui koneksi kontrol, dan menunggu sebuah koneksi dari *FC*. Dalam hal ini, *FC* melakukan *bind port* asal dari koneksi ke *port* acak (> 1023) yang telah dispesifikasikan tadi.

Untuk menggunakan mode pasif, *FC* mengirimkan perintah *PASV* ke *FS*, dan *FS* akan mengirimkan balasan seperti '*227 Entering Passive Mode* (127,0,0,1,78,52)'. Sintaks dari *IP* dan *port* adalah sama dengan argumen pada perintah *PORT* untuk mode aktif diatas.

3. Mode *Extended* Pasif

Mode *Extended* Pasif adalah mode yang sama dengan mode pasif, ditinjau dari penginisiasi data. Pada mode extended pasif, *FS* beroperasi sama seperti mode pasif, namun ia hanya mentransmisikan nomor *port* (tidak mengirimkan alamat *IP*, seperti pada mode aktif dan/atau pasif diatas) dan *FC* akan berasumsi bahwa ia akan terhubung ke *IP Address* yang sama dengan *FS* yang telah terhubung sekarang. Mode extended pasif ditambahkan pada *RFC* 2428.

2.1.5 Mode Transmisi

Mode Transmisi pada FTP menspesifikasikan format transmisi data yang digunakan.

Terdapat tiga buah mode transmisi pada FTP, yaitu: [RFC959]

1. Mode *Stream*

Pada mode *stream*, *file* akan ditransmisikan sebagai aliran *byte* (*stream of bytes*). Untuk sebuah struktur *file*, tanda *EOF* (*End Of File*) ditandai dengan pengirim menutup koneksi data. Untuk struktur *record*, sebuah urutan 2-byte spesial menandai *EOR* (*End Of Record*) dan *EOF* (*End Of File*).

2. Mode *Block*

Pada mode *block*, *file* ditransfer sebagai sebuah seri dari *block* (*series of blocks*), dengan masing-masing didahului oleh satu atau lebih *byte header*.

3. Mode *Compressed*

Pada mode *compressed*, sebuah *run-length encoding* sederhana mengkompresi adanya *byte* tertentu. Dalam sebuah *file text*, hal ini akan mengakibatkan pengkompresan spasi, dan dalam sebuah *binary file*, hal ini akan mengakibatkan kompresi dari *byte 0*. (mode ini jarang sekali digunakan atau didukung. Ada cara lain untuk melakukan kompresi *file* di *FTP*)

Sebagai perbandingan, mode pertama, dapat mem-format data dan mengijinkan prosedur *restart*; mode kedua, sama halnya dengan mode yang pertama tetapi juga mengijinkan penkompresan data untuk *transfer* yang efisien; dan mode ketiga, mode yang melewati data dengan sedikit atau tidak ada sama sekali pemrosesan. Pada kasus terakhir ini, mode berinteraksi dengan atribut struktur untuk menentukan tipe pemrosesan. Pada mode kompresi, tipe representasi menentukan *byte* pengisi.

Semua *transfer* data harus dilengkapi dengan sebuah penanda akhir *file* (*EOF*) yang secara eksplisit dinyatakan atau diakibatkan oleh penutupan koneksi data. Untuk *file* dengan struktur *record*, semua penanda akhir record (*EOR*) adalah eksplisit, termasuk yang terakhir. Untuk *file* yang ditransmisikan dalam struktur halaman, sebuah tipe halaman *last-page* digunakan.

2.1.6 Tipe Data

Dalam melakukan transfer data, FTP juga menspesifikasikan tipe data yang ditransfer. Tipe data ini adalah representasi internal karakter dari data. Terdapat empat jenis tipe data, yaitu : [RFC959]

1. ASCII

File text dikirim melewati koneksi data dalam bentuk ASCII. Hal ini menyebabkan pengirim harus mengubah format *file text* lokal menjadi ASCII, dan penerima harus mengubah dari ASCII ke format *file text* lokalnya sendiri. Akhir dari tiap baris ditransfer menggunakan representasi ASCII dari karakter *carriage return*, diikuti dengan sebuah *line feed*. Ini berarti bahwa penerima harus melakukan *scanning* setiap *byte*, untuk mencari pasangan *carriage return* dan *line feed*.

2. EBCDIC

Sebuah cara alternatif dalam melakukan transfer *file text* ketika kedua ujung dari layanan FTP adalah sistem EBCDIC.

3. *Image (Binary)*

Data dikirim sebagai sebuah alur bit bersambung (*contiguous stream of bits*). Biasanya digunakan untuk melakukan transfer *file binary*.

4. Local

Sebuah cara untuk melakukan transfer *file binary* antara *host* dengan ukuran *byte* yang berbeda. Jumlah *bit* per *byte* dispesifikasikan oleh pengirim. Untuk sistem yang menggunakan delapan *bit* per *byte*, sebuah tipe *file local* dengan ukuran *byte delapan bit* adalah sama dengan tipe *file image*.

2.1.7 Format Control

Format control adalah karakter kontrol yang digunakan untuk pengaturan (*format*) dari data. Terdapat tiga jenis *format control*, yaitu : [RFC959]

1. *Non-print*

File tidak mengandung informasi format vertikal.

2. *Telnet command control*

File mengandung kontrol format vertikal *telnet* yang bisa diinterpretasikan *printer*.

3. *Carriage control*

Karakter pertama dari setiap baris adalah karakter kontrol format *fortran*.

2.1.8 Struktur Data

Struktur data pada FTP mendefinisikan struktur internal dari data yang ditransfer. Terdapat tiga jenis struktur pada FTP, yaitu : [RFC959]

1. Struktur *file*

File dianggap sebagai sebuah aliran *byte* bersambung (*contiguous stream of bytes*). Tidak ada struktur *file* internal.

2. Struktur *record*

Struktur ini hanya digunakan pada *file text* (ASCII atau EBCDIC).

3. Struktur *page*

Setiap halaman ditransmisikan dengan nomor halaman untuk membiarkan penerima menyimpan halaman-halaman dalam sebuah urutan yang acak. Disediakan oleh Sistem Operasi TOPS-20. (RFC tentang *Host Requirement* merekomendasikan tidak mengimplementasikan struktur ini)

2.1.9 Perintah FTP

Perintah pada FTP dapat dilihat pada lampiran C.1. Perintah ini merupakan perintah yang terdapat pada [RFC959].

2.1.10 Return Codes FTP

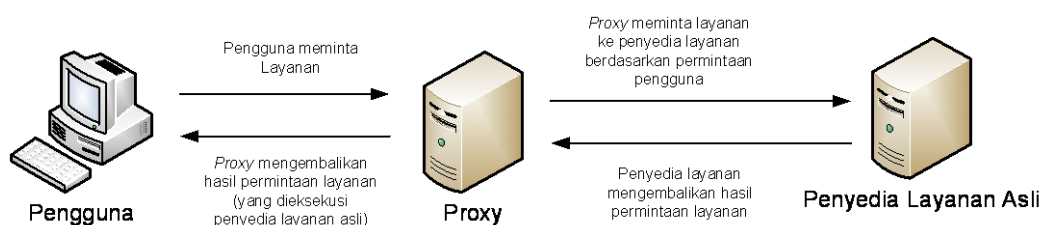
Return Code FTP dapat dilihat pada lampiran C.2. Return Codes ini merupakan *return codes* yang terdapat pada [RFC959].

2.2 Proxy

Menurut asal katanya, *proxy* berarti wakil. Dalam konteks jaringan, *proxy* berfungsi sebagai alat untuk meningkatkan ketersediaan dan performansi layanan dengan mengurangi beban dari jaringan dan *server* yang menyediakan *resource* [TAN022]. Hal ini dilakukan dengan cara melayani permintaan dari pengguna dengan meneruskan permintaan tersebut ke penyedia layanan yang sebenarnya, dan jika diinginkan dapat melakukan penyimpanan *resource* sementara yang disebut *cache*. *Cache* akan dijelaskan kemudian.

2.2.1 Cara Kerja Proxy

Dari sisi pengguna, *proxy* sama seperti penyedia layanan asli. Pengguna hanya perlu mengirimkan permintaan layanan, dan *proxy* akan melayani permintaan tersebut. Namun dalam proses eksekusi layanan tersebut, alih-alih mengeksekusinya sendiri, *proxy* melakukan permintaan layanan ke penyedia layanan asli. Setelah penyedia layanan asli memberikan hasil, kemudian *proxy* baru akan mengembalikan hasil eksekusi permintaan layanan ke pengguna. Sehingga dari sisi penyedia layanan asli, *proxy* sama seperti pengguna layanan. Gambar II-3 menggambarkan cara kerja *proxy* secara umum.



Gambar II-4 Cara kerja *proxy* secara umum

2.2.2 *Caching* dan Redundansi

Dalam melayani permintaannya, *proxy* dapat menambah kehandalan layanannya dengan melakukan proses *caching* dan penambahan redundansi.

Cache adalah sebuah penyimpanan dari koleksi data yang digunakan sebelumnya, yang letaknya lebih dekat dengan si peminta data daripada penyedia data aslinya [COU01]. Hal ini dapat diartikan bahwa data aslinya lebih mahal untuk diambil (misalnya karena waktu akses yang lama) atau untuk dikomputasi ulang, dibandingkan dengan biaya untuk membaca dari *cache*. Dengan kata lain, *cache* adalah sebuah area penyimpanan sementara dimana data yang sering diakses bisa selalu disimpan untuk pengaksesan yang cepat. *Caching* adalah aktivitas untuk membuat *cache*.

Sewaktu pengguna *cache* ingin mengakses sebuah datum yang mungkin terdapat di penyimpanan aslinya, pengguna terlebih dahulu akan mengecek *cache*. Jika sebuah entitas ditemukan, maka akan digunakan datum yang terdapat di *cache*. Situasi ini disebut *cache hit* [TAN021]. Persentase dari akses yang ternyata mengalami *cache hit* [COU01] dari akses keseluruhan adalah *hit rate* atau *hit ratio* dari *cache*. Situasi lainnya, dimana datum yang dicari tidak terdapat di *cache*, disebut *cache miss* [COU01].

Cache terdiri dari kumpulan entitas. Tiap entitas mempunyai sebuah datum yang merupakan salinan dari data asli di penyimpanan asli. Tiap entitas juga mempunyai sebuah *tag*, yang menspesifikasikan identitas datum di penyimpanan asli.

Sewaktu datum ditulis di *cache*, maka ia harus ditulis juga di penyimpanan aslinya. Kebijakan penentuan waktu yang ditentukan untuk penulisan ini disebut Kebijakan menulis (*write policy*).

Ada beberapa jenis *write policy* [RAN98], yaitu :

1. *Write-through*

Setiap data yang ditulis di *cache*, maka langsung akan ditulis di penyimpanan asli.

2. *Write-back*

Penulisan data tidak langsung dilakukan pada saat ada perubahan di *cache*. Namun, *cache* akan mencatat perubahan apa saja yang terjadi di *cache*, dan data ini akan ditulis di penyimpanan aslinya sewaktu data itu diambil dari *cache* (permintaan data dilakukan bukan oleh *cache*). Karena itu, mungkin saja terjadi kesalahan atau ketidak sinkronan data yang terdapat di *cache* dan penyimpanan asli, dimana mungkin terjadi perubahan di penyimpanan asli dan *cache* sekaligus. Sehingga untuk alasan ini, pada *cache* disediakan dua layanan untuk akses data, yaitu untuk mengakses datum, dan untuk menulis data yang berubah dari *cache* ke penyimpanan asli.

3. *No-write*

Pada *policy* ini, tidak diperbolehkan pengubahan data melalui *cache*.

Selain itu, terdapat juga kebijakan penggantian (*replacement policy*), yang menentukan datum mana yang akan dihapus jika penyimpanan pada *cache* tidak mencukupi lagi dan ada datum baru yang akan ditambahkan ke *cache*.

Jenis Kebijakan penggantian[NUT00], antara lain adalah :

1. *First In First Out (FIFO)*

Kebijakan penggantian ini akan menghapus datum yang paling awal masuk untuk diganti dengan datum yang baru.

2. *Least Recently Used (LRU)*

Kebijakan penggantian ini akan menghapus datum yang paling lama digunakan untuk diganti dengan datum yang baru. Algoritma ini memerlukan pencatatan datum apa saja yang digunakan pada setiap saat, yang cukup mahal untuk selalu memastikan bahwa datum yang dihapus adalah selalu yang paling lama digunakan.

3. *Most Recently Used (MRU)*

Kebijakan penggantian ini akan menghapus datum yang paling akhir digunakan untuk digantikan dengan datum yang baru. Mekanisme *caching* ini digunakan jika akses tidak dapat diprediksi, dan menentukan datum yang paling awal digunakan pada suatu sistem *cache* merupakan sebuah operasi

dengan kompleksitas yang tinggi. Salah satu contohnya adalah *cache* memori basis data.

4. *Least Frequently Used (LFU)*

Kebijakan penggantian ini akan menghapus datum yang paling sedikit digunakan untuk digantikan dengan datum yang baru.

Sedangkan, redundansi adalah suatu kegiatan untuk melakukan toleransi kegagalan system dengan menggunakan komponen yang redundan [COU01]. Hal ini dilakukan dengan tujuan untuk meningkat kehandalan dari sistem, biasanya sebagai *backup*. Secara asal kata, redundan berarti berlebih-lebihan. Redundan yang dimaksud dalam konteks perangkat lunak terdistribusi adalah komponen yang menduplikasi komponen aslinya. Penentuan komponen yang dibuat redundan ditentukan oleh perancang sistem, biasanya adalah komponen yang penting bagi sistem. Metode yang digunakan pada *caching* dapat pula digunakan pada redundansi. Perbedaan mendasar antara *caching* dan redundansi adalah penempatan data; bahwa data *caching* ditempatkan pada *host* penyedia layanan utama, sedangkan redundansi dapat diletakkan pada sebarang *host* yang menyediakan layanan penyimpanan dan pengambilan data.