PENGATURAN *DOWNLOAD FILE* PADA *PC-CLIENT* MENGGUNAKAN *MULTITHREADING*

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer
dalam Bidang Ilmu Komputer

oleh :
MOCHAMAD RIFAI
0410963037-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011

LEMBAR PENGESAHAN SKRIPSI

PENGATURAN *DOWNLOAD FILE* PADA *PC-CLIENT*
MENGGUNAKAN *MULTITHREADING*

oleh:
MOCHAMAD RIFAI
0410963037-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 4 April 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	Pembimbing II,
Bayu Rahayudi,ST., MT	**Dany Primanita, ST**
NIP. 197407122006041001	**NIP. 197711162005012003**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP. 196709071992031001


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	: Mochamad Rifai
NIM	: 0410963037-96
Jurusan	: Matematika
Penulis Skripsi berjudul	: Pengaturan *download file* pada *PC-Client* menggunakan *multithreading*

Dengan ini menyatakan bahwa :

1. Isi dari Skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.
2. Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 4 April 2011
Yang menyatakan,

(Mochamad Rifai)
NIM. 0410963037-96

PENGATURAN *DOWNLOAD FILE* PADA *PC-CLIENT* MENGGUNAKAN *MULTITHREADING*

ABSTRAK

Bandwidth merupakan salah satu bagian penting dalam dunia internet. Tanpa *bandwidth* yang memadai layanan yang menggunakan internet seperti pendidikan, ekonomi dan bisnis akan mengalami hambatan. *Bandwith* berbanding terbalik dengan waktu, semakin besar *bandwidth* yang tersedia semakin kecil waktu yang dibutuhkan. Salah satu cara untuk mengatasi ketersedian *bandwidth* adalah dengan melakukan optimasi pada sisi klien dan salah satu metode yang dapat digunakan adalah *multithreading*.

Pada penelitian ini, penerapan *multithreading* dilakukan dengan memaksimalkan kerja *stream* yang dikombinasikan dengan konsep *threading*. Dengan konsep *threading* memungkinkan masing-masing *stream* dapat berjalan sendiri-sendiri sesuai dengan *threadnya* tanpa mengganggu *stream* yang lain. Implementasi *multithreading* dalam penelitian ini dilakukan dengan menggunakan java virtual machine dan diharapkan dapat diketahui pengaruh jumlah *thread* terhadap waktu *download*.

Berdasarkan implementasi dan analisis yang telah dilakukan, metode *multithreading*, *thread* dalam jumlah banyak tidak dapat langsung digunakan tetapi harus dimanajemen. Salah satu penyebab bertambahnya waktu *download* adalah penanganan *thread* pada java yang tidak bersifat *asynchronous* sehingga apabila terdapat *thread* yang mengalami kemacetan jaringan (*congestion*) akan berada pada kondisi *blocked state* sampai koneksi terbentuk.

PENGATURAN *DOWNLOAD FILE* PADA *PC-CLIENT* MENGGUNAKAN *MULTITHREADING*

ABSTRACT

Bandwidth is one important part di internet. Without enough bandwidth, services using internet such as education,economics and business wil have obstacles. Bandwidth is inversely prortional to time, the greater available bandwidth the less time required. Oen way to cope available bandwidth is to perform optimization on the client side and one method that can be used is multithreading.

In this research, multithreading implementation is doing by maximizing the stream that combined with threading concept. With the concept of threading allows each stream work alone in accordance with the thread without disturbing the other streams. multithreading implementation in this research using java virtual machine and expected to know the influence of number of threads to download time.

Based on analysis and implementation has been done, multithreading method can not maximal because the thread that running with large quantities can be directly used but must has managed. One of the cause that increased download time is the handling of the thread in java which is not asynchronous so the thread that get a network congestion will be in block state condition until a connection is established.

KATA PENGANTAR

Penelitian ini bertujuan untuk melakukan penerapan *multithreading* pada *PC-Client* untuk mengelola proses *download* dengan harapan dapat memaksimalkan *bandwidth* yang tersedia pada koneksi *client-server* dalam jaringan internet.

Dengan mengetahui hasil kinerja multithreading, diharapkan dapat dijadikan sebagai sebuah masukan penerapan *multithreading* sebagai sebuah solusi untuk memperpendek waktu *download*.

Dalam penyusunan tugas akhir ini, Penulis mengucapkan terima kasih kepada :

1. Bayu Rahayudi, ST., MT., selaku pembimbing utama penulisan tugas akhir ini.
2. Dany Primanita, ST., selaku pembimbing pendamping dalam penulisan tugas akhir ini.
3. Dr. Abdul Rouf Al-Ghofari, M.Sc, selaku Ketua Jurusan Matematika FMIPA Universitas Brawijaya..
4. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada Penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Jurusan Matematika FMIPA Universitas Brawijaya.
5. Kepada kedua orang tua Penulis yang tak pernah berhenti memberikan doa dan dukungannya kepada Penulis.
6. Rekan-rekan di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan bantuannya demi kelancaran pelaksanaan penyusunan tugas akhir ini.
7. Rekan – rekan Mabes Ilmu komputer yang selalu membantu baik tenaga maupun pikiran dalam membantu menyelesaikan laporan dan melakukan penelitian ini.
8. Segenap staff dan karyawan di jurusan Matematika yang sudah banyak membantu dalam urusan administrasi.

Penulis berharap semoga tugas akhir ini dapat memberikan manfaat kepada pembaca dan bisa diambil manfaatnya, baik oleh Penulis selaku mahasiswa maupun pihak-pihak lain yang tertarik

untuk menekuni dan berkecimpung di pengembangan dunia jaringan komputer, komunikasi data, dan internet.

Malang, 4 April 2011

Penulis

DAFTAR ISI

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman


UNIVERSITAS BRAWIJAYA

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1. 1. Latar Belakang Masalah

Perkembangan teknologi informasi pada saat ini tidak bisa dipisahkan dari Internet. Pada awal kemunculannya, Internet merupakan sesuatu yang mahal dan hanya dapat digunakan oleh perusahaan dan instansi besar serta memiliki dukungan finansial yang mapan namun saat ini paradigma tersebut telah berubah, Internet tidak hanya murah namun tersedia di berbagai tempat secara gratis dengan menggunakan media *wireless* (*hotspot*).

Bandwidth merupakan salah satu aspek yang vital dalam dunia Internet. Tanpa *bandwidth* yang memadai, layanan yang menggunakan Internet seperti bidang pendidikan, pertahanan keamanan, bisnis online, penyedia layanan email dan lain lain akan mengalami hambatan dalam menjalankan aktifitasnya, sayangnya *bandwidth* sendiri berbanding lurus dengan biaya, semakin besar *bandwidth* yang disewa maka semakin besar biaya yang dibutuhkan. Masalah yang sering muncul dalam penyediaan *bandwith* adalah pengaturan atau manajemen *bandwidth.* Implementasi manajemen *bandwidth* bertujuan untuk mengoptimalkan *bandwidth* yang tersedia sehingga mampu memberikan jaminan alokasi *bandwidth* kepada pengguna Internet untuk dapat menggunakan layanan yang dibutuhkan meskipun terjadi gangguan pada jaringan Internet, seperti kemacetan jaringan (*over traffic*) dan tidak stabilnya bandwidth yang diterima dari *Internet service provider* (ISP).

Optimasi *bandwidth* dapat dilakukan pada sisi *server* maupun *client*. Pada sisi *server* , optimasi *bandwidth* dilakukan pada proses *upload* sehingga file atau paket data yang dikirim bisa diterima *client* secara maksimal . Optimasi *bandwidth* juga bisa dilakukan pada sisi *client*, Kebalikan dari optimasi pada sisi server, apabila dilakukan pada sisi *client* maka yang dimaksimalkan adalah proses *downloadnya.* Optimasi *download* dapat dilakukan dengan memaksimalkan kerja *stream* yang dikombinasikan dengan konsep *threading.* Dengan konsep *threading* memungkinkan masing-masing *stream* dapat berjalan sendiri-sendiri sesuai dengan *threadnya* tanpa mengganggu *stream* yang lain. Berdasarkan latar belakang yang telah disebutkan sebelumnya, titik fokus dalam penelitian adalah

meneliti kinerja *multithreading* untuk memaksimalkan *bandwidth* yang tersedia.Sesuai dengan jurnal yang berjudul *multithreading – an efficient technique for enchanching aplication performance* yang menjadi dasar dalam penelitian ini, parameter yang digunakan dalam pengujian adalah waktu, karena waktu adalah parameter yang bisa diukur secara nyata oleh klien. Waktu berbanding terbalik dengan *available bandwidth* (*bandwidth* yang tersedia) , sehingga apabila *available bandwdith* kecil maka waktu yang dibutuhkan untuk download juga semakin lama. *Available bandwidth* umumnya *ditentukan* oleh penyedia jasa internet namun dari sisi klien juga bisa memaksimalkan waktu *download* dengan menggunakan *tools* bantuan seperti *proxy* dan *download manager.* Dengan menggunakan prinsip kerja serangkain *tools* tersebut dilakukan penelitian untuk memaksimalkan waktu *download* dengan menggunakan prinsip kerja *multithreading*.

1.2 Rumusan Masalah

Berdasar latar belakang tersebut maka rumusan masalah dalam penelitian yang akan dikerjakan adalah :

1. Bagaimana menerapkan *Multithreading* untuk memaksimalkan *bandwidth* .
2. Bagaimana pengaruh *multithreading* terhadap waktu *download* dan korelasinya terhadap banyaknya *thread* yang digunakan.

1. 3. Batasan Masalah

Batasan masalah dalam penelitian ini adalah :

1. URL (*Uniform resource Locater*) yang digunakan berupa *direct link.*
2. Protokol yang digunakan adalah HTTP (*Hypertext Transfer Protocol*).
3. Manajemen *thread* diatur sepenuhnya oleh Sistem Operasi.
4. *Bandwidth* yang akan digunakan dalam penelitian sebesar 15 KB .

1. 4. Tujuan

Tujuan yang ingin dicapai dalam penelitian ini adalah sebagai berikut :

1. Melakukan implementasi dan analisis kinerja *multithreading* untuk memaksimalkan *bandwidth* pada parameter waktu *download* .
2. Mengetahui perhitungan waktu *download* dan pengaruh penggunaan *thread* .

1. 5. Manfaat

Manfaat yang bisa diambil dari penulisan tugas akhir ini adalah mengetahui kegunaan dan penerapan *multithreading* untuk mengoptimalkan waktu *download* .

1. 6. Metodologi Pemecahan Masalah

Untuk dapat mencapai tujuan penelitian yang telah dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan tugas akhir ini adalah sebagai berikut :

1. Studi literatur, mempelajari teori dan dokumentasi yang berkaitan dengan *multithreading*.
2. Pendefinisian dan analisis masalah, melakukan definisi dan analisis masalah untuk mendapatkan sebuah solusi yang tepat.
3. Perancangan dan implementasi sistem sesuai dengan permasalahan yang akan diteliti khususnya tentang optimasi *bandwidth* dan mengimplementasikan hasil rancangan tersebut untuk diterapkan sesuai dengan permasalahan yang akan diteliti.
4. Uji coba dan evaluasi hasil implementasi; menguji sistem yang telah dibangun dan mengevaluasi hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan penelitian yang telah dirumuskan sebelumnya.

1. 7. Sistematika Penulisan

Sistematika penulisan tugas akhir ini akan disusun seperti berikut

BAB I PENDAHULUAN

Bab ini membahas mengenai latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi pemecahan masalah, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini membahas tinjauan pustaka yang digunakan dalam penelitian ini tentang *stream*, protokol HTTP, *bandwidth*, *thread, download* serta referensi lainya yang mendukung penelitian ini.

BAB III METODOLOGI PENELITIAN

Bab ini membahas mengenai metodologi yang akan digunakan dalam implementasi dan analisis kinerja *multithreading* seperti perangkat keras dan lunak yang digunakan, konfigurasi sistem dan metode yang digunakan untuk pengolahan data selama penelitian.

BAB IV ANALISIS DAN PEMBAHASAN

Bab ini berisi penerapan metode, analisis dan pembahasan dan perhitungan waktu yang yang didapat selama penelitian.

BAB V PENUTUP

Bab ini berisi kesimpulan dari pembahasan selama penelitian , serta saran yang diharapkan dapat bermanfaat bagi penelitian saat ini dan selanjutnya.

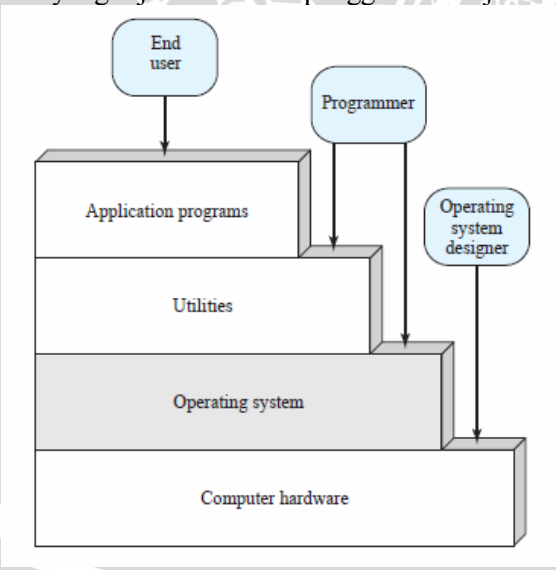
BAB II
TINJAUAN PUSTAKA

2. 1 Sistem Operasi

Sistem operasi adalah sebuah program yang mengendalikan eksekusi sebuah aplikasi dan bertindak sebagai jembatan antara aplikasi tersebut dan perangkat keras(Stallings,2003). Sistem operasi secara garis besar memiliki beberapa tujuan :

1. *Convinience*, sistem operasi dibuat untuk untuk membuat komputer semakin nyaman digunakan
2. *Efficiency,* sistem operasi memungkinkan penggunaan sumber daya komputer secara efisien
3. *Ability to Evolve*, sebuah sistem operasi harus dibangun sedemikian rupa sehingga mampu memungkinkan pengembangan lebih lanjut, kemampuan mengenali perangkat keras dan selalu menyediakan layanan seiring dengan perkembangan jaman.

Hubungan antara sistem operasi dengan perangkat keras, dan aplikasi yang dijalankan oleh pengguna ditunjukan pada gambar 2.1



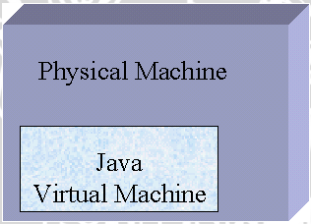
Gambar 2.1 *Layer* dalam Sistem operasi (Stallings,2003)

2.2 JVM (*Java Virtual Machine*)

JVM adalah sebuah mesin komputer yang abstrak dan dapat bekerja seperti mesin komputasi yang sesungguhnya, JVM memiliki sekumpulan set instruksi dan mampu memanipulasi berbagai area pada memori pada saat dijalankan.

Implementasi prototipe pertama dari mesin virtual java dilakukan di sun microsytem,inc. saat ini implementasi komponen JVM, SDK dan *Java runtime Environment* (JRE) meniru mesin virtual java pada win32 dan solaris dengan cara yang jauh lebih canggih.

JVM menggunakan beberapa format tertentu dalam bahasa java , seperti format biner, format file kelas. Sebuah file class berisi instruksi untuk dijalankan oleh JVM. Dengan alasan keaman , JVM memaksakan format yang kuat dan menggunakan struktur klasik pada file *class*. Meskipun demikian setiap bahasa pemrograman dengan menggunakan fungsi *class* bisa menggunakan JVM meskipun berbeda sistem operasi. *Virtual machine* ini merupakan emulasi dari prosesor yang sesungguhnya dan digambarkan pada gaambar 2.2(Lindholm,1999).

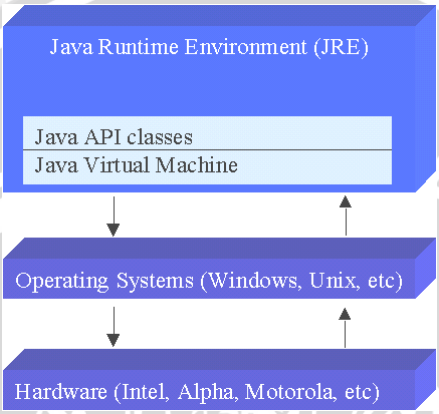


Gambar 2.2 ilustrasi JVM pada CPU (Lindholm,1999)

JVM bertanggung jawab menerjemahkan *java bytecode* dan meneruskanya untuk diproses sebagai sebuah *action* oleh sistem operasi. Sebagai contoh terdapat *request* tmelakukan koneksi antar *socket*, sistem operasi yang berbeda akan menangani koneksi *socket* tersebut dengan cara yang berbeda, namun sebagai *programmer* anda tidak perlu menghiraukan hal seperti itu, karena semua adalah tanggung jawab dari JVM(Lindholm,1999).

JVM adalah bagian dari sistem yang lebih luas yaitu JRE. Masing-masing sistem operasi dan arsitektur CPU membutuhkan JRE yang berbeda. JRE terdiri dari satu set class dasar yang merupakan implementasi dari JAVA API serta sebuah JVM, sifat

portable yang dimilki java salah satunya adalah karena hal ini (mampu berjalan dalam arsitektur CPU dan sistem operasi yang berbeda) dan dapat diilustrasikan pada gambar 2.3



Gambar 2.3 Hubungan antar JRE, sistem operasi dan *hardware*(Lindholm,1999)

2.3 *Thread*

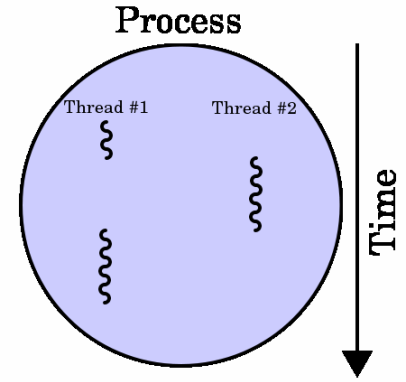
Thread adalah satuan unit dari sebuah proses kerja. *Thread* meliputi konteks prosesor (meliputi *program counter* dan *stack pointer*) dan masing-masing *thread* memiliki area data / area kerja tersendiri sehingga memungkinkan percabangan atau penggunaan *thread* dalam waktu bersamaan) (Stallings,2003). *Thread* dieksekusi secara berurutan dan dapat disela sehingga memungkinkan prosesor untuk menjalankan *thread* yang lain. Setiap proses yang dijalankan oleh prosesor dalam satu satuan waktu terdiri dari satu atau lebih *thread*, skema proses *threading* ditunjukan pada gambar 2.4

Multithreading adalah pengeksekusian beberapa thread terjadi dalam sebuah proses dalam waktu yang hampir bersamaan, saling berbagi sumber daya tetapi dapat dijalankan secara independen sesuai kebutuhan.

Keuntungan *Multithreading* :

1. **Responsif**, Aplikasi interaktif menjadi tetap responsif meskipun sebagian dari program sedang diblok atau melakukan operasi lain yang panjang. Sebagai contoh, sebuah *thread* dari *web*

browser dapat melayani permintaan pengguna sementara *thread* yang lain berusaha menampilkan gambar.



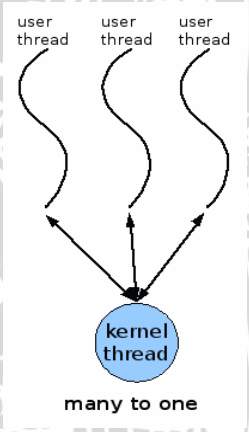
Gambar 2.4 Skema proses *Threading* (Carver,2006)

2. **Berbagi sumber daya**, Beberapa *thread* yang melakukan proses yang sama akan berbagi sumber daya. Keuntungannya adalah mengizinkan sebuah aplikasi untuk mempunyai beberapa *thread* yang berbeda dalam lokasi memori yang sama.
3. **Ekonomis,** Pembuatan sebuah proses memerlukan pengalokasian memori dan sumber daya. Alternatifnya adalah dengan menggunakan *thread*, karena *thread* membagi memori dan sumber daya yang dimilikinya sehingga lebih ekonomis untuk membuat *thread* dan *context switching thread*. Akan susah mengukur perbedaan waktu antara *thread* dan *switch*, tetapi secara umum pembuatan dan pengaturan proses akan memakan waktu lebih lama dibandingkan dengan *thread*. Pada Solaris, pembuatan proses memakan waktu 30 kali lebih lama dibandingkan pembuatan *thread* sedangkan *proses context switch* 5 kali lebih lama dibandingkan *context switching thread*.
4. **Utilisasi arsitektur multiprosesor,** Keuntungan dari *multithreading* dapat sangat meningkat pada arsitektur multiprosesor, dimana setiap *thread* dapat berjalan secara paralel di atas prosesor yang berbeda. Pada arsitektur prosesor tunggal, CPU menjalankan setiap *thread* secara bergantian tetapi hal ini berlangsung sangat cepat sehingga menciptakan ilusi paralel, tetapi pada kenyataanya hanya satu *thread* yang dijalankan CPU pada satu-satuan waktu.

Multithreading pada masing- masing sistem operasi mempunyai perlakuan yang berbeda tergantung pada model yang digunakan oleh sistem operasi tersebut.

Model – model *multithreading* :

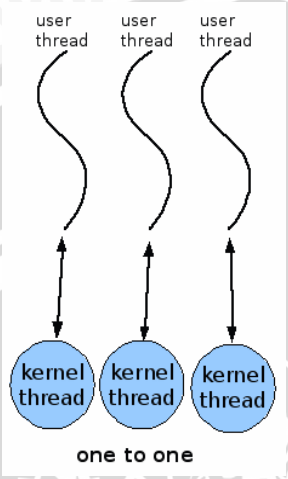
1. Model ***Many-to-One***, model ini memetakan beberapa *thread* tingkatan pengguna ke sebuah *thread*. tingkatan kernel. Pengaturan *thread* dilakukan dalam ruang pengguna sehingga efisien. Hanya satu *thread* pengguna yang dapat mengakses *thread* kernel pada satu saat. Jadi *Multiple thread* tidak dapat berjalan secara paralel pada multiprosesor. Contoh: Solaris Green Threads dan GNU Portable Threads. Model ***Many-to-One*** ditunjukan pada gambar 2.5.



Gambar 2.5 *Thread many to one*
(Stallings,2003)

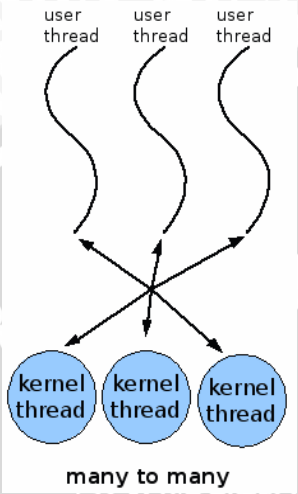
2. Model ***One-to-One***, model ini memetakan setiap *thread* tingkatan pengguna ke setiap *thread*. Ia menyediakan lebih banyak *concurrency* dibandingkan model *Many-to-One*. Keuntungannya sama dengan keuntungan *thread* kernel. Kelemahan model ini ialah setiap pembuatan *thread* pengguna memerlukan tambahan *thread* kernel. Karena itu, jika mengimplementasikan sistem ini maka akan menurunkan kinerja dari sebuah aplikasi sehingga biasanya jumlah *thread*

dibatasi dalam sistem. Contoh: Windows NT/XP/2000 , Linux, Solaris 9. Model ***One-to-One*** ditunjukan pada gambar 2.6.



Gambar 2.6 *Thread one to one*(Stallings,2003)

3. Model ***Many-to-Many*** . Model ini memultipleks banyak *thread* tingkatan pengguna ke *thread* kernel yang jumlahnya sedikit atau sama dengan tingkatan pengguna. Model ini mengizinkan developer membuat *thread* sebanyak yang ia mau tetapi *concurrency* tidak dapat diperoleh karena hanya satu *thread* yang dapat dijadwalkan oleh kernel pada suatu waktu. Keuntungan dari sistem ini ialah kernel *thread* yang bersangkutan dapat berjalan secara paralel pada multiprosesor. Contoh : Solaris 8, Tru64 Unix , *java virtual machine* di Solaris (Sun documentation,2010). Model ***Many-to-Many*** ditunjukan pada gambar 2.7



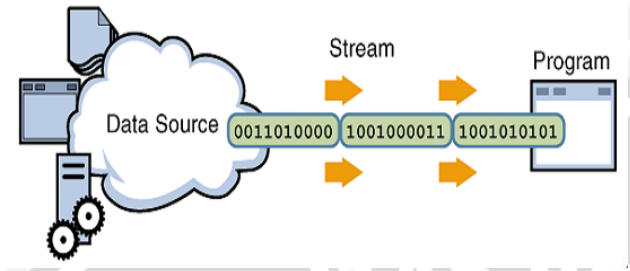
Gambar 2.7 *Thread many to many*
(Stallings,2003)

2.4 *Stream*

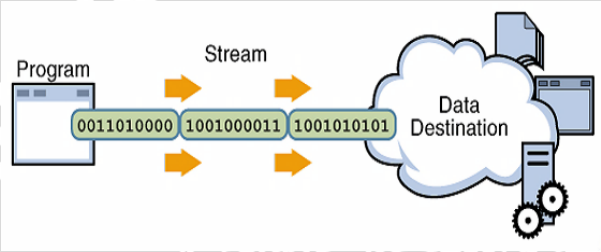
Pengertian stream atau *data stream* secara umum adalah bentuk urutan kode sinyal (paket data) yang digunakan dalam pertukaran informasi. *Stream* banyak digunakan karena sifatnya yang fleksibel, dapat diterapkan di berbagai macam sumber dan dikirim sesuai kebutuhan misalnya : array pada memori, disket, *socket* jaringan dll.

Selain karena sifatnya yang fleksibel *stream* juga banyak digunakan untuk berbagai macam tipe data seperti *byte,* tipe data primitif (*integer,string dsb*), karakter dan tipe data obyek. *Input stream* adalah *stream* yang berasal dari sumber (*source*) kemudian dikirim ke tujuan (program) sedangkan *output stream* merupakan kebalikan dari *input stream* yaitu *stream* yang berasal dari program kemudian dikirim ke tujuan untuk selanjutnya diolah, dalam satu waktu hanya satu *input* maupun *output stream* yang dapat di proses oleh *thread.*

Penjelasan aliran data input dan output *stream* ditunjukan pada gambar 2.8 dan 2.9 .



Gambar 2.8 *input stream*



Gambar 2.9 *output stream*

2.5 Konsep Jaringan Komputer

Dalam bidang komputer, jaringan dapat diartikan sebagai dua atau lebih komputer yang dihubungkan sehingga dapat berhubungan dan dapat berkomunikasi, sehingga akan menimbulkan suatu efisiensi, sentralisasi, dan optimasi kerja (Stallings,2003). Suatu komputer dapat berhubungan dengan komputer lain dan saling berkomunikasi (salah satunya bertukar data) tanpa menggunakan disket, CD atau *flashdisk* dari satu komputer ke komputer lainnya seperti yang telah lazim dilakukan tanpa jaringan komputer.

Ada beberapa jenis jaringan komputer dilihat dari cara pemrosesan data dan pengaksesannya :

1. ***Host-terminal***

   Host-terminal adalah suatu konfigurasi dimana terdapat satu atau lebih server yang dihubungkan dalam suatu dumb terminal. Karena *dumb terminal* hanyalah sebuah monitor

yang dihubungkan dengan kabel RS-232, maka pemrosesan data dilakukan di dalam *server*, oleh karena itu maka suatu server haruslah sebuah sistem komputer yang memiliki kemampuan pemrosesan yang tinggi dan penyimpanan data yang sangat besar.

2. ***Client*-server**

   . *Client – server* adalah sebuah metode dimana satu atau lebih server dihubungkan dengan beberapa *client* atau terminal. Server sendiri memberikan layanan kepada *client* seperti *DHCP server*, *database server* dan berbagai macam layanan server lainnya, sedangkan *client* adalah pengguna dari layanan tersebut, oleh sebab itu server diharuskan memiliki kemampuan mampu menangani *load* kinerja yang tinggi.

3. ***Peer to peer***

   Peer to peer merupakan suatu metode dimana terdapat beberapa terminal komputer yan dihubungkan dengan media transmisi. Secara prinsip, hubungan *peer to peer* ini adalah bahwa setiap komputer dapat berfungsi sebagai server (penyedia layanan) maupun *client*, keduanya dapat difungsikan dalam suatu waktu yang bersamaan.

2.6 TCP / IP (*Transfer Control Protocol / Internet Protocol*)

TCP/IP dapat didefinisikan sebagai kumpulan aturan dan prosedur yang mengatur kumpulan komputer dan perangkat jaringan untuk dapat saling berkomunikasi atau bertukar paket data. Saat ini hampir seluruh jaringan komputer di dunia mengadopsi TCP/IP sebagai protokol standar karena dapat diterapkan di hampir semua perangkat keras dan sistem operasi (Casad,2003). Arsitektur TCP/IP dapat digambarkan seperti gambar 2.10.

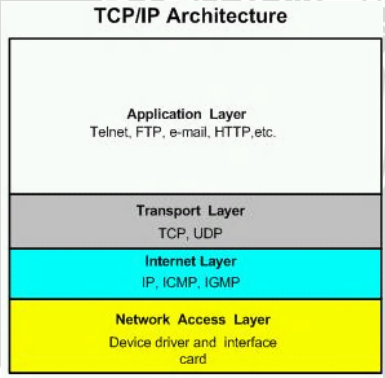
TCP adalah protokol data stream yang digunakan secara luas di Internet, sedangkan *Internet Protocol* (IP) adalah protokol lapisan *internetwork* (*internetwork layer* dalam *DARPA Reference Model*) yang digunakan oleh protokol TCP/IP untuk melakukan pengalamatan dan *routing* paket data antar *host-host* di jaringan komputer berbasis TCP/IP. Untuk mendukung layanan jaringan komputer, TCP/IP mempunyai tugas utama sebagai berikut :

1. Membagi paket data menjadi potongan–potongan yang dapat secara efisien melewati media transmisi.

2. Sebagai perantara dengan perangkat keras jaringan.
3. Mengatur paket data supaya dapat sampai ke tujuan dengan baik (*logical addressing* dan *physical addressing*) .
4. Mengatur supaya tidak terjadi kemacetan data dalam jaringan (*flow control*).
5. Mengatur supaya paket data yang dikirimkan ke tujuan tidak mengalami kerusakan (*error control*) .

Di bawah ini adalah ilustrasi arsitektur TCP/IP layer

a. *Network Access Layer* :bertanggung jawab dalam meletakkan frame-frame jaringan di atas media jaringan yang digunakan. TCP/IP dapat bekerja dengan banyak teknologi, mulai dari teknologi dalam LAN (seperti halnya Ethernet dan Token Ring).
b. *Internet layer* : mengizinkan *host* untuk memasukkan paket-paket ke dalam jaringan apapun dan terkirim ke tujuan secara independen. Pada *internet layer* juga diatur pengalamatan pada jaringan komputer.
c. *Transport layer* : berfungsi melakukan segmentasi pada *application layer* dan mengirimkan satu segmen dari satu *host* ke *host* lainnya..
d. *Application layer* : bertanggung jawab untuk menyediakan akses kepada aplikasi terhadap layanan jaringan TCP/IP. Protokol ini mencakup *Dynamic Host Configuration Protocol* (DHCP), *Domain Name System* (DNS), Hypertext Transfer Protocol (HTTP), *File Transfer Protocol* (FTP).



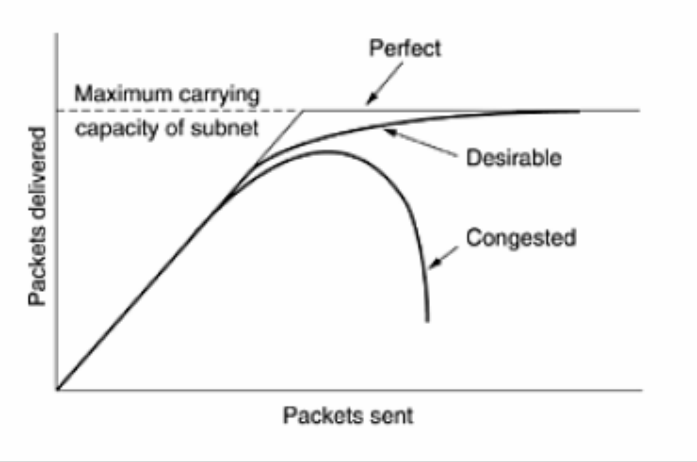
Gambar 2.10 Arsitektur TCP/IP (Stallings,2003)

2.7 *Bandwidth*

Bandwidth merupakan kapasitas maksimal dalam mengirim atau menerima data melalui koneksi jaringan per satuan waktu, umumnya diukur dalam satuan bit per detik. *Bandwidth* dapat dianalogikan sebagai lebar jalan raya dan mobil sebagai paket data, semakin lebar atau besar kapasitas *bandwidth* semakin besar pula data yang dapat lewat per satuan waktu. *Bandwidth* merupakan kapasitas yang diberikan sistem untuk melakukan transfer data selama terjadi koneksi.

Metode yang digunakan dalam memaksimalkan *bandwidth* sendiri bermacam – macam seperti *Stochastic Fairness Queuing* (SFQ), *Packets First-In First-Out* (PFIFO) *Hierarchical Token Bucket* (HTB) dan *Per Connection Queue* (PCQ), pemilihan metode tersebut harus disesuaikan dengan kebutuhan user dan besarnya jaringan yang harus ditangani.

Salah satu manfaat dari optimasi *bandwidth* adalah bagaimana mengatasi terjadinya kemacetan dalam jaringan komputer, umumnya kemacetan dalam jaringan komputer terjadi karena permintaan akan *bandwidth* lebih besar daripada kapasitas *bandwidth* yang tersedia.

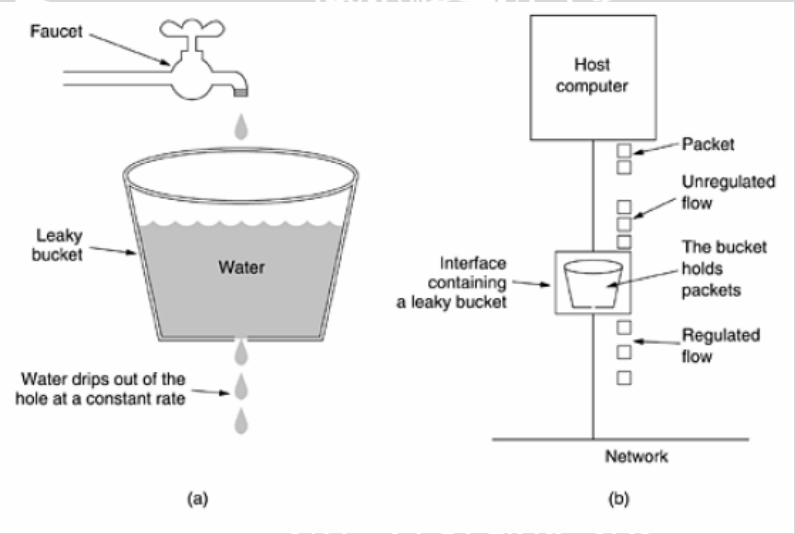


Gambar 2.11 Grafik hubungan pengiriman paket dan kemacetan (Tanenbaum,2003)

Seperti dapat dilihat pada gambar 2.11 apabila jumlah paket yang dikirim (*desirable*) masih mencukupi dengan jumlah kapasitas

dari subnet (*maximum carrying capacity of subnet*) maka paket data akan dapat dikirim mendekati sempurna, sedangkan apabila terjadi kemacetan dalam jaringan(*congested*), paket data yang dikirim akan menurun secara drastis dan hampir tidak ada paket yang dikirimkan (Tanembaum,2003)

Antrian atau *queue* dalam jaringan komputer dapat didefinisikan sebagai kumpulan paket data yang menunggu untuk diproses untuk selanjutnya dikirim ke tujuan. Untuk mendukung kinerja manajemen *bandwidth* terdapat konsep manajemen antrian yaitu menahan semua paket pada router kemudian mengeluarkan paket tersebut sesuai dengan kebutuhan. Analogi manajemen antrian bisa dilihat pada kasir swalayan, semakin banyak pelanggan akan semakin lama antrian, apabila tidak banyak pelanggan maka petugas kasir dapat langsung melayani pelanggan.



Gambar 2.12 (a) *leaky bucket* dengan air, (b) *leaky bucket* dengan paket (Tanenbaum,2003)

Sesuai dengan ilustrasi gambar 2.12, konsep antrian pada jaringan komputer dapat dianalogikan seperti *leaky bucket* atau ember bocor. Sedangkan ember dapat difungsikan sebagai router yang mengatur paket data, banyaknya air pada ember dapat dianalogikan sama dengan banyaknya antrian yang terjadi pada router, tugas dari router sendiri adalah menjaga supaya paket data

(aliran air) menuju host dapat stabil atau dengan kata lain manjamin kebutuhan *bandwidth* terhadap *host* dapat terpenuhi(Tanenbaum,2003).

2.8 Manajemen Kemacetan dan Antrian

Manajemen *congestion* merupakan pengertian umum yang mencakup penggunaan strategi antrian untuk mengatur situasi dimana permintaan akan *bandwidth* lebih besar daripada *bandwidth* yang tersedia pada jaringan. Bila terlalu banyak paket yang terdapat pada *subnet*, dapat mengakibatkan unjuk kerja jaringan kian menurun, yang pada akhirnya akan menyebabkan kemacetan (*congestion*).

Gambar 2.11 menggambarkan *symptom* kemacetan. Bila jumlah paket yang mengalir ke dalam *subnet* dari *host* masih berada dalam daya tampungnya, paket-paket tersebut seluruhnya akan diantarkan (kecuali untuk beberapa paket yang mengalami kesalahan dalam transmisi). Jumlah paket yang diantarkan proporsional dengan jumlah paket yang dikirimkan. Akan tetapi dengan semakin meningkatnya lalu lintas, router tidak mampu lagi menangani paket yang datang, dan router akan mulai kehilangan paket. Kecenderungan seperti ini semakin memperburuk keadaan. Pada lalu lintas yang sangat padat, unjuk kerja *subnet* sepenuhnya jatuh, dan hampir tidak ada paket yang dihantarkan. (Tanembaum, 20003).

Kemacetan bisa disebabkan oleh beberapa faktor. Bila semuanya terjadi dengan tiba-tiba, aliran paket yang datang pada tiga atau empat saluran input dan semuanya memerlukan saluran output yang sama, maka antrian mulai membesar. Bila tidak terdapat memori yang cukup untuk menampung seluruh antrian, maka paket akan hilang.

Permasalahan yang serius yang diakibatkan efek *congestion* adalah *deadlock*, yaitu suatu kondisi dimana sekelompok node tidak bisa meneruskan pengiriman paket karena tidak ada *buffer* yang tersedia. Teknik *deadlock avoidance* digunakan untuk merancang jaringan sehingga *deadlock* tidak terjadi. (Tim Penelitian dan Pengembangan Wahana Komputer, 2003).

Salah satu yang berhubungan dengan kemacetan dan atrian adalah *throughput* atau paket data rata-rata yang diterima dengan sukses tanpa kesalahan dalam saluran komunikasi. Perhitunga n throughput dilakukan sesuai dengan penelitian yang dilakukan oleh

Fulvio Risso dan Pavos Gevros dari *University College London* dalam karyanya yang berjudul *Operational and Performance Issues of a CBQ Router*, nilai akurasi dan persentase akurasi *troughput* terhadap *bandwidth* yang dialokasikan serta *loss datagram*. Dari hasil pengujian, akan dilihat perbandingan rata-rata *bandwidth* dan *troughput* yang diperoleh untuk masing masing pengujian dengan menggunakan *thread* yang berbeda, perhitungan yang digunakan adalah sebagai berikut :

1.Nilai akurasi *throughput*

nilai akurasi *troughput* terhadap nilai *bandwidth* teralokasi. Nilai akurasi merupakan selisih antara *bandwidth* teralokasi dengan *bandwidth* terukur. Hal ini bisa dirumuskan menjadi :

$$Nilai\_akurasi = abs(bandwidth\_teralokasi - bandwidth\ terukur) \quad (2.1)$$

2. Persentase akurasi *throughput*

Setelah nilai akurasi *throughput* terhadap nilai *bandwidth* teralokasi diperoleh, kemudian dihitung persentase keakurasiannya dengan menggunakan rumus :

$$prosentase\_akurasi = \frac{bandwidth\_teralokasi - nilai\_akurasi}{bandwidth\_teralokasi} * 100\% \quad (2.2)$$

3.*Loss datagram*

Loss datagram diasumsikan sebagai *datagram* yang hilang pada jaringan internet, *datagram* yang rusak dan kemudian dibuang, juga bisa berupa *datagram* yang tidak urut pada TCP. Pengecekan *loss datagram* dilakukan secara manual dengan utilitas *ping* sebelum, ketika, dan setelah proses transfer. Walaupun proses *ping* juga berarti mengirimkan paket menuju mesin tujuan, akan tetapi karena ukuran paket yang relatif kecil (32 bytes) dan utilitas ini sudah lazim digunakan untuk

melakukan pengecekan konektifitas jaringan sederhana, maka pengaruhnya bisa diabaikan.

2.9 HTTP *(Hypertext Transfer Protocol)*

HTTP merupakan protokol dasar yang digunakan oleh WWW (*worl wide web*) . HTTP mendifinisikan bagaimana pesan diformat dan dikirim dan tindakan apa yang harus dilakukan oleh *web server* dan *browser* dalam berbagai perintah yang telah disepakati dalam RFC 2616. HTTP berfungsi sebagai *request-response protocol* dalam bentuk klien-server. *Web browser* berfungsi sebagai klien dan *web hosting* penyedia situs berperan sebagai server. Klien akan mengirimkan pesan HTTP *request* kemudian server akan memproses pesan tersebut dan akan mengirimkan *response,* respon yang dikirim oleh server dapat berupa kode status HTTP atau tampilan sebuah halaman HTML atau dapat berupa file.

Di dalam HTTP terdapat kode status yang berisi response dari *server* (*web server, File Transfer Protocol server*) terhadap klien (*web browser, download manager*). Berikut ini adalah kode status yang dipergunakan dalam protokol HTTP :

Tabel 2.1 Kode status yang dipergunakan dalam protokol HTTP

No	Kode	Kode Status Pada Protokol HTTP
1	200	*request was fullfilled* kode status ini memberitahukan bahwa *request* terhadap server terpenuhi semua, semua verifikasi terhadap semua yang diminta klien dapat dipenuhi oleh server.
2	201	mengikuti perintah POST, memberitahukan status *success* atau berhasil tetapi ada beberapa bagian teks dan garis atau gambar tidak berhasil di-*load* dikarenakan terdapat kesalahan pada server .
3	202	202, sama dengan 201 yang mengindikasikan sukses tetapi gambar dan teks tidak berhasil di-*load* karena proses yang terhenti secara tiba-tiba umumnya terjadi karena kesalahan pada *browser.*
4	203	*request* terhadap server dapat diterima dan dipahami,

		dan informasi yang dikirim kembali tentang *response* tersebut berasal dari pihak ketiga, bukan dari server asli.
5	204	*request* terhadap server dapat diterima dan dipahami akan tetapi data yang *direquest* oleh klien tidak tersedia di server .

HTTP dibuat 10 tahun setelah FTP dan banyak digunakan karena mempunyai beberapa kelebihan dibanding FTP, antara lain :

1. Mendukung koneksi *multiple file* dalam 1 server.
2. HTTP melakukan kompresi data secara otomatis .
3. HTTP mendukung fitur proxy.

2.10 *URL* (Uniform Resource Locator)

URL diciptakan pada 1994 oleh Tim Berners-lee, URL adalah alamat global dokumen dalam *word wide web* (Darma,2010). URL mempunyai skema atau standar penulisan seperti unix yang menggunakan tanda "//" dan "/" sebagai pemisah antar fragmen. Skema penulisan dalam URL secara umum adalah :

< nama protokol > : // < hirarki > / [?query]

Penjelasan skema :

1. **Nama protokol** adalah protokol yang digunakan dalam mengakses URL tersebut misal HTTP,FTP atau HTTPS.
2. **Hirarki** berisi alamat identifikasi yang berisi hirarki atau *path* atau alamat yang di- *request* oleh klien (*web browser*). Hirarki dibagi menjadi :
   - **a.** ***Authority***, pada bagian ini dibagi lagi menjadi 3 bagian, yaitu bagian yang berisi informasi user dan diakhiri dengan tanda "@" , bagian yang berisi *hostname* biasanya berisi alamat IP atau alamat domain dan bagian terakhir yang bersifat opsional yang berisi nomor *port* .
   - **b.** ***Path,*** bagian ini adalah lokasi / urutan direktori dan dipisahkan oleh " / " dan mengarah pada sebuah file, baik berupa file HTML, citra atau dokument .

3. ***Query*** merupakan bagian dari skema URL yang berisi informasi tambahan seperti baris kode PHP. Pada bagian ini selalu diawali dengan " ? "

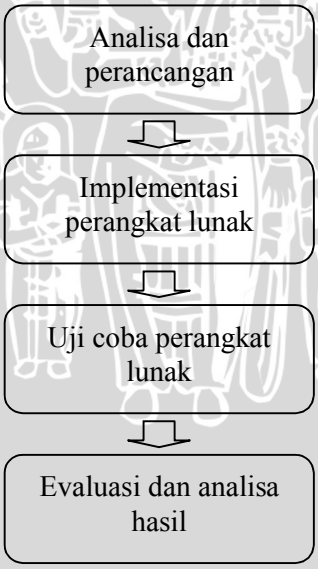
Beberapa contoh penerapan URL :

1. *HTTP:// username:password @namadomain.com:8042/over/ there/ index.html*
2. *FTP:// www. pcwebopedia.com/stuff.exe*
3. *HTTPS ://mail.google.com/mail/?shva=1*

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai metode dan tahap-tahap yang digunakan dalam mengimplementasikan *download manager* menggunakan *multithreading.* Adapun tahapan pembuatanya adalah sebagai berikut sebagaimana yang tergambar dalam gambar 3.1 .

1. Studi literatur mengenai penerapan *multithreading* untuk mengoptimalkan waktu *download.*
2. Merancang dan mengimplementasikan perangkat lunak yang akan digunakan dalam penelitian ini.
3. Melakukan implementasi *multithreading* ke dalam perangkat lunak *download manager* yang akan digunakan dalam penelitian ini.
4. Melakukan uji coba terhadap perangkat lunak *download manager* yang mendukung *multithreading* .
5. Melakukan evaluasi terhadap hasil yang diperoleh dari serangkaian percobaan dan membandingkanya dengan hasil yang didapat secara teoritis atau manual.



Gambar 3.1 Skema penelitian

3.1 Obyek Penelitian

Obyek yang dijadikan penelitian secara umum adalah *thread* dan secara khusus adalah kinerja *thread* dengan parameter waktu. *Thread* dalam penelitian ini akan digunakan dalam proses *download* file melalui internet dan menggunakan protokol HTTP. Penerapan *thread* diharapkan dapat memaksimalkan *bandwidth* yang tersedia sehingga diharapkan dapat memperkecil waktu untu me-*download* file dari internet.

3.2 Konfigurasi Sistem

Dalam penelitian ini akan digunakan konfigurasi sistem sebagai berikut :

1. Terdapat koneksi internet yang akan digunakan untuk proses *download* file.
2. File akan ditempatkan di dalam sebuah *web server*.
3. Aplikasi *multithreading* akan ditempatkan di klien dan akan diamati pengaruhnya terhadap proses *download* .

Topologi yang akan digunakan dalam penelitian ini ditunjukan pada gambar 3.2 :

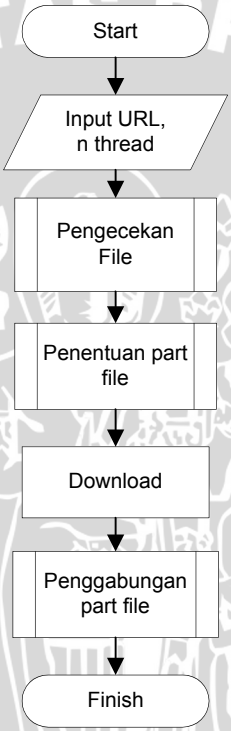
Gambar 3.2 Topologi jaringan

3.3 Skema Proses

Dalam penelitian ini akan ditentukan langkah-langkah proses sebagai berikut :

1. Akan dilakukan pengecekan URL untuk menentukan apakah *file* tersebut bisa di-*download* atau tidak.

2. Apabila *file* bisa di-*download* maka akan dilanjutkan dengan menentukan ukuran *part file* sesuai dengan jumlah *thread* yang akan digunakan.
3. Setelah ukuran *part file* ditentukan maka akan dilanjutkan dengan men-*download* masing-masing *part file* .
4. Setelah semua *part file* selesai di-*download* maka seluruh *part file* akan di gabung lagi menjadi satu *file* yang utuh.



Gambar 3.3 *Flowchart download manager* dengan *multithreading*

3.3.1 Pengecekan *File*

Langkah pertama yang dilakukan adalah memeriksa kondisi file yang terdapat pada *web server* . Pengecekan dilakukan dengan memanfaatkan kode status yang menjadi standar di protokol HTTP, apabila status yang diberikan bernilai “2xx” maka akan dilanjutkan

dengan mendapatkan ukuran file tersebut. *Flowchart* pengecekan *file* ditunjukan pada gambar 3.4 . *Pseudocode* yang akan digunakan dalam menentukan ukuran masing-masing bagian file yang dipecah kemudian dilanjutkan dengan proses *download* sebagai berikut :

```
1. Open  HTTP socket
2. Connect  HTTP socket
3. If ( kode status / 200 ) = 2
4.   Cek ukuran file
5.   If ukuran file  > 0
6.     Tentukan ukuran part file
7.   Else
8.     Error
9. Else
10.    Error
11.Finish
```

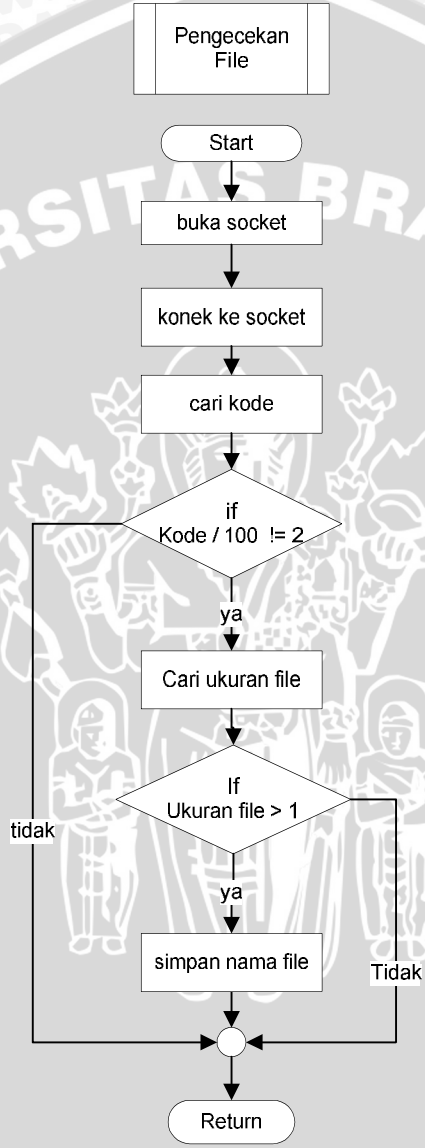
3.3.2 Penentuan ukuran *part file*

Setelah didapatkan ukuran ukuran file maka didapatkan variabel awal ukuran file yang akan digunakan untuk langkah selanjutnya yaitu memecah file ke sejumlah n bagian. variabel n akan digunakan untuk menentukan jumlah *thread* .*Pseudocode* yang akan digunakan dalam menentukan ukuran masing-masing bagian file yang dipecah kemudian dilanjutkan dengan proses *download* sebagai berikut :

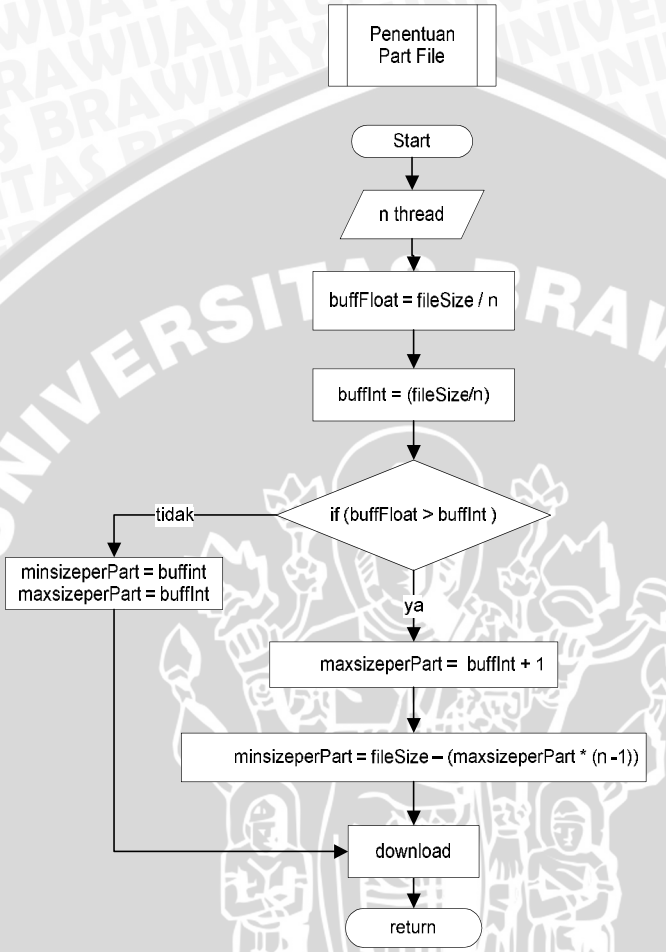
```
1. Tentukan jumlah n
2. buffFloat = fileSize / n
3. buffInt = (fileSize/n)
4. if (buffFloat > buffInt )
5.     maxsizeperPart = buffInt + 1
6. minsizeperPart = fileSize -  (maxsizeperPart * (n
       -1))
7. end if
8. else
9.     minsizeperPart = maxsizeperPart = buffInt
10.download (part)
```

Cara ini digunakan agar ukuran masing-masing bagian file cenderung sama besar. Kemudian bagian (n) akan dijadikan diproses

download tersendiri oleh masing-masing *thread*. *Flowchart* atau diagram alir pada bagian ini ditunjukan pada gambar 3.5 .



Gambar 3.4 *Flowchart* pengecekan *File*



Gambar 3.5 *Flowchart* penentuan ukuran *part file*

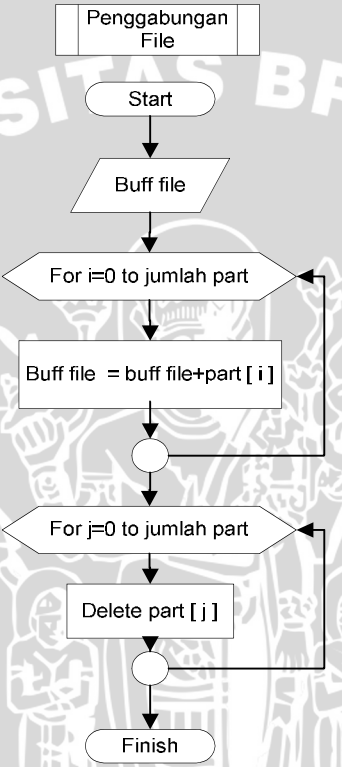
3.3.3 Penggabungan File

Proses penggabungan file dilakukan apabila masing-masing thread selesai melakukan prosesnya. *Part file* yang sudah selesai didownload semuanya akan digabung (*merge*) menjadi satu file yang utuh sesuai dengan gambar 3.6

Pseudocode yang akan digunakan dalam proses ini adalah :

1. `Inisialisasi buffFile`
2. `For i=0 to jumlahPart do`

```
3.     BuffFile = buffFile+part[i]
4.  End for
5.  For j=0 to jumlahPart do
6.     Delete part[j]
7.  End for
```



Gambar 3.6 *Flowchart* penggabungan *file*

3.4 Perancangan Hasil Analisis

Perancangan hasil analisis digunakan untuk menentukan langkah-langkah percobaan yang akan digunakan untuk pengambilan kesimpulan. Di dalam penelitian akan akan dicari perbandingan optimasi *download* dengan menggunakan 2 *thread*, 4 *thread*, 6 *thread*, 8 *thread* dan 10 *thread*. Ukuran file yang akan di-*download*

sebesar 1 MB , 2 MB, 3 MB, 4 MB dan 5MB . Tabel pengambilan data yang akan digunakan dapat dilihat pada tabel 3.1 .

Tabel 3.1 Contoh tabel pengambilan data dengan *thread* sejumlah n dengan sejumlah file dengan ukuran yang berbeda

Uji ke-	File (MB)	n thread	T
1			
2			
3			
X			

Keterangan :
n_{thread} : jumlah *thread*
t : waktu pemrosesan dengan satuan detik

Contoh :
Tabel 3.2 Contoh pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 100KB dengan *thread* sejumlah 2

Uji ke-	Ukuran File (MB)	n thread	t (ms)
1	1	2	6,3
2	1	2	6,1
3	1	2	6,8
4	1	2	7
5	1	2	6,5
6	1	2	7,7
7	1	2	6,5
8	1	2	6,4
9	1	2	6,3
10	1	2	6,2

Dari pengambilan data tersebut akan dilakukan analisis hasil percobaan sesuai dengan yang telah ditetapkan sebelumnya dan dari data hasil pengujian akan dibuat grafik yang mempresentasikan hasil dari pengujian secara keseluruhan.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Berikut ini adalah penjelasan tentang proses, *source code* dan tampilan program yang telah dibuat serta analisa terhadap data yang dihasilkan.

4.1 *Application Environment*

Lingkungan (*environment*) pengembangan aplikasi ini terdiri dari perangkat keras (*hardware*) dan perangkat lunak (*software*) .

4.1.1 *Hardware Environment*

Perangkat keras (*hardware*) yang digunakan dalam penelitian adalah sebagai berikut :

1. Laptop dengan *processor* Intel 1,3 Ghz
2. 2 GB RAM
3. 250 GB Hardisk
4. Koneksi internet 1Mbps

4.1.2 *Software Environment*

Sedangkan untuk perangkat lunak (*software*) yang digunakan adalah :

1. Microsoft windows 7 sebagai sistem operasi.
2. Netbeans 6.9 sebagai *compiler*.

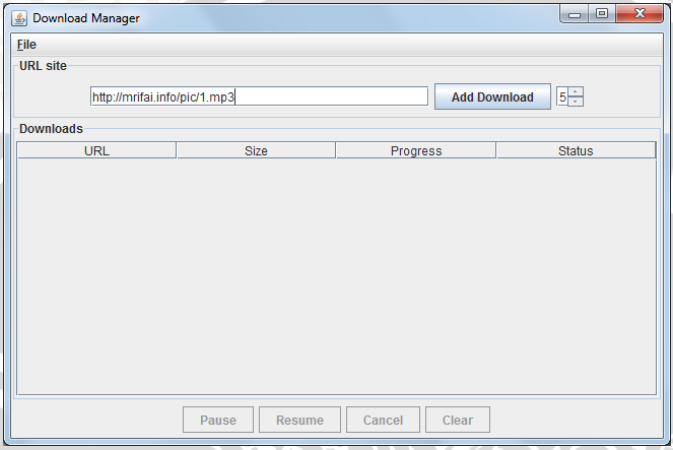
4.2 Implementasi

Secara keseluruhan sistem ini dibagi menjadi beberapa bagian yaitu pembuatan *thread* , menentukan ukuran *part file* , pengecekan *url*, proses *download* dan *merge* (menggabungkan) file yang sudah di download .

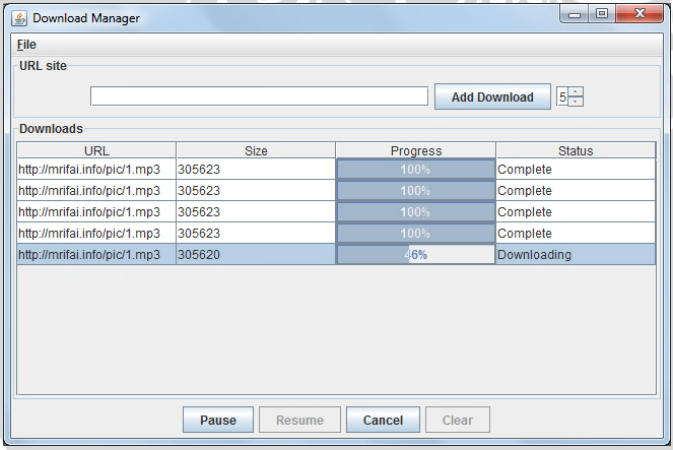
4.2.1 Implementasi Antarmuka

Antarmuka digunakan untuk memberi kemudahan dalam menjalankan fungsi yang disediakan aplikasi yang digunakan dalam penelitian ini. Implementasi antarmuka aplikasi dapat dilihat pada gambar 4.1. Seperti ditunjukan pada gambar 4.1 (a) terdapat *text field* yang digunakan untuk mengisi alamat

URL dan disebelah kanan terdapat komponen *spin edit* yang digunakan untuk mengisi jumlah *thread* yang akan digunakan dalam pengujian.



(a)



(b)

Gambar 4.1 (a) Tampilan awal saat aplikasi dijalankan, (b) Tampilan ketika proses download sedang berlangsung

Proses *download* ditunjukan pada gambar 4.1 (b) , ketika proses *download* berlangsung *file* yang di-*download* akan langsung terbagi menjadi sejumlah *part* sesuai dengan jumlah *thread* yang diisi pada *spin edit.*

4.2.2 Struktur Data

Struktur data berkaitan dengan pengaturan yang khusus untuk mengorganisir dan menyimpan data. Setiap struktur data didesain supaya dapat dikelola dan dipakai secara tepat. Strukur data dalam penelitian ini digunakan untuk menyimpan variabel yang diperlukan dalam melakukan *download.* Class DownloadData digunakan sebagai tipe data di dalam *linked list Source code* pembuatan class DownloadData ditunjukan pada gambar 4.2 .

```
public class DownloadData
{
        String fileName;
        int totalPart, completePart;
        int fileSize;
}
```

Gambar 4.2 *Source code* class DownloadData

Penggunaan class tersendiri dilakukan supaya variabel di dalam class DownloadData tersebut bisa digunakan di class lain. Di dalam aplikasi ini digunakan *native linked list* karena sudah mewakili fungsi yang dibutuhkan.

4.2.3 Pembuatan *Thread*

Bagian ini merupakan implementasi awal dari penelitian ini. *Thread* dibangkitkan di class Download dengan mengimplementasikan *interface runnable.* Di dalam class Download terdapat instans yang memanggil *thread* seperti dijelaskan di gambar 4.3 .

```
private void download()
 {
   Thread thread = new Thread(this);
   thread.start();
   System.out.println("starting
    : " + thread.getName());
 }
```

Gambar 4.3 *Source code* pembuatan instans *thread*

4.2.4 Pengecekan *File*

Pengecekan *File* diawali dengan membuka koneksi ke URL kemudian dilanjutkan dengan menghubungkan *socket* java ke *URL* tujuan. Ketika *socket* sudah terkoneksi, dilanjutkan dengan pengecekan kode status HTTP. Setelah semua langkah dilakukan maka hal terakhir yang dilakukan adalah pengecekan konten *URL* . *Source code* pengecekan URL dapat dilihat pada gambar 4.4 .

```
HttpURLConnection connection =
                    (HttpURLConnection)
url.openConnection();

connection.setRequestProperty("Range","bytes=" +
downloaded + "-");

connection.connect();
if (connection.getResponseCode() / 100     != 2)
{
  error();
}
int contentLength = connection.getContentLength();
if (contentLength < 1)
{
  error();
}
```

Gambar 4.4 *Source code* pengecekan *File*

4.2.5 Penentuan ukuran *part file*

Sebelum *file* di *download*, terdapat salah satu tahapan yaitu menentukan ukuran *part file. File* yang akan didownload harus dibagi menjadi beberapa bagian sesuai dengan banyaknya *thread* yang digunakan, misal *file* sebesar 5 MB akan di-*download* dengan

menggunakan 5 thread, maka ukuran *file* tersebut akan dibagi dengan jumlah *thread* dan menghasilkan *part file* dengan ukuran masing-masing sebesar 1 MB. *Source code* yang digunakan untuk proses penentuan ukuran *part file* digambarkan sesuai dengan gambar 4.5.

```
private void countingPart()
{
  if (part>1)
  {
     float bufFloat = float)fileSize/(float)part;
     System.out.println(bufFloat);
     int bufInt = fileSize/part;
     System.out.println(bufInt);

     if (bufFloat > bufInt)
     {
       this.maxSize = bufInt + 1;
     }
     Else
     {
       this.maxSize = bufInt;
     }
     minSize = fileSize - (maxSize * ( part-1));
  }

  else if (part==1)
  {
     minSize = maxSize = fileSize;
  }

  for(int i=0; i<part; i++)
  {
     if (i<part-1)
     System.out.println(i+1 + " : " + maxSize);
     else                System.out.println(i+1 +
     " : " + minSize);
  }
}
```

Gambar 4.5 *Source code* penentuan ukuran *part file*

4.2.6 Proses *Download*

Setelah *file* dipecah sesuai dengan jumlah *thread* maka *file* tersebut di-*download.* Nama *file* yang akan di-*download* diambil dari *URL* yang dituju ditambahkan ekstensi *part number* dan akan disimpan di *subfolder* temporary. Source code yang digunakan dalam proses download ditunjukan pada gambar 4.6 .

```
this.fileName = getFileName(url);
this.filePartName = fileName + "._" + partNumber;

file = new RandomAccessFile(".//temporary//" +
filePartName, "rw");

file.seek(downloaded);

stream = connection.getInputStream();
stream.skip(startingByte);

while (status == DOWNLOADING)
{
   byte buffer[];
   buffer = new byte[partSize - downloaded];
   int read = stream.read(buffer);
   System.out.println(partNumber + " : " + read
   + "\tdownloaded : " + downloaded);

   if ((read == -1) || (downloaded == partSize))
     break;

   file.write(buffer, 0, read);
   downloaded += read;
   stateChanged();
 }
```

Gambar 4.6 *Source code* proses *download*

4.2.7 *Merge File*

Proses *merge file* dilakukan apabila semua *part file* sudah selesai di-*download.* Langkah awal dalam proses ini adalah mencari *path file* dan *part file* di dalam *folder* temporary kemudian dilakukan pembacaan *file* dan dilanjutkan dengan pembuatan *file* baru yang berisi *part file* yang telah selesai di-*download* sebelumnya seperti dijelaskan dalam gambar 4.7 .

```
private void merge()
{
  byte[] bufByte = new byte[mergePart.fileSize];
  int part = 0;

 for(int i=0; i<mergePart.totalPart; i++)
 {
    String bufPartName = (System.getProperty("user.dir")
     + "\\temporary\\" + mergePart.fileName + "._" + i);

    Try
    {
        FileInputStream bufPartStream = new
        FileInputStream(bufPartName);
        int streamSize = bufPartStream.available();
        for(int j=0; j<streamSize; j++)
        {
           bufByte[part] = (byte)bufPartStream.read();
           part++;
        }

        bufPartStream.close();
    }
    catch(Exception e)
    {
      System.out.println("cannot open file : " + e);
    }

    File delFile = new File(bufPartName);
    delFile.delete();
  }

  try
  {
    FileOutputStream outStream = new
    FileOutputStream(outFile);
    outStream.write(bufByte);
    outStream.close();
  }
  catch(Exception e)
  {
    System.out.println("cannot write file " + e);
  }
}
```

Gambar 4.7 *Source code* proses *merge file*

4.3 Data Hasil Pengujian

4.3.1 Hasil Pengujian Jumlah *Thread* Terhadap Waktu *Download*

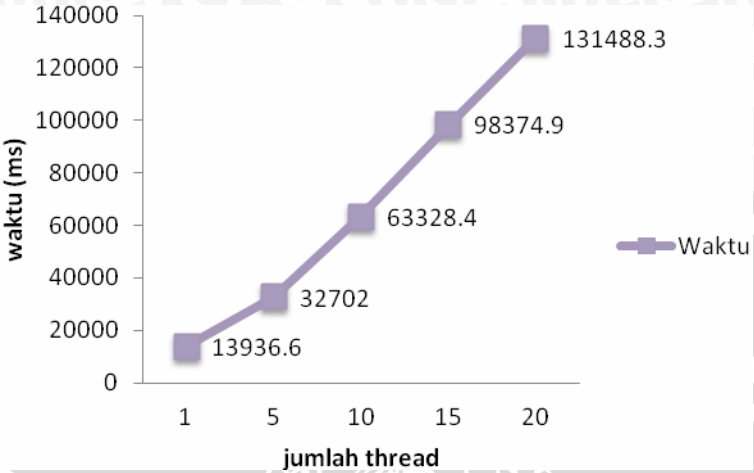
Dari serangkaian impementasi dan perancangan sistem yang telah dilakukan maka akan didapat data hasil pengujian. Hasil pengujian jumlah *thread* terhadap waktu *download* dengan ukuran *file* sebesar 1,2 MB menghasilkan nilai rata-rata seperti tertera pada tabel 4.1.

Pada tabel 4.1 menunjukan waktu rata-rata yang dibutuhkan untuk men-*download file* sebesar 1,2 MB dengan menggunakan 1 *thread* sebesar 13936,6 ms atau 13,9466 detik. Pengujian selanjutnya akan dilakukan dengan men-*download file* yang sama dengan ukuran yang sama (1,2 MB) dengan jumlah *thread* sebanyak 5,10,15 dan 20. Hasil pengujian menghasilkan nilai rata-rata 32702 ms untuk 5 *thread*, 63328,4 ms untuk 10 *thread*, 98374,9 ms untuk 15 *thread* dan 131488,3 ms untuk 20 *thread*.

Tabel 4.1 Tabel rata-rata waktu *download* terhadap jumlah *thread* untuk *file* sebesar 1,2 MB

No.	Thread	Rata-rata waktu (ms)
1	1	13936,6
2	5	32702
3	10	63328,4
4	15	98374,9
5	20	131488,3

Dari tabel 4.1 menunjukan jumlah *thread* berbanding lurus dengan waktu yang dibutuhkan, semakin banyak jumlah *thread* yang digunakan akan menambah waktu *download* . Grafik hubungan antara *thread* dan rata-rata waktu *download* ditunjukan pada gambar 4.8 .

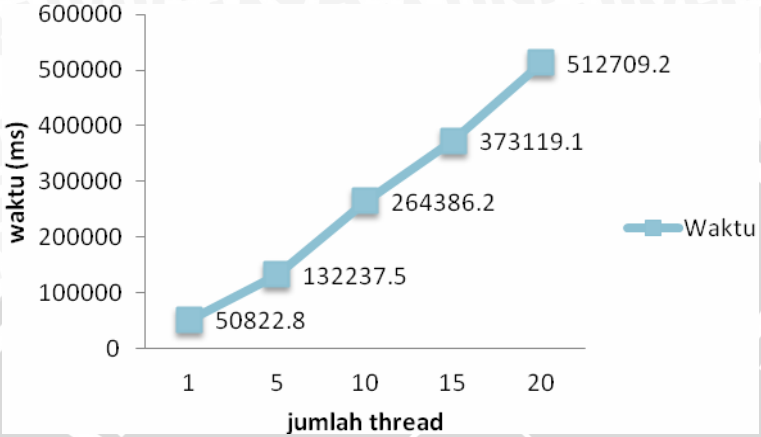


Gambar 4.8 Grafik pengaruh jumlah *thread* terhadap waktu *download* yang dibutuhkan untuk file sebsar 1,2 MB

Pengujian selanjutnya dilakukan dengan ukuran *file* sebesar 5 MB dan jumlah *thread* sebanyak 1,5,10,15 dan 20. Dari hasil pengujian diatas didapat nilai rata-rata waktu yang dibutuhkan dan dapat dilihat pada tabel 4.2 . Sesuai dengan tabel 4.2 jumlah *thread* berbanding lurus dengan waktu *dowload* . Grafik hubungan antara waktu *download* dengan jumlah *thread* pada pengujian menggunakan file sebesar 5 MB dapat dilihat di gambar 4.9 .

Tabel 4.2 Tabel rata-rata waktu *download* terhadap jumlah *thread* untuk *file* sebesar 5 MB

No.	Thread	Rata-rata waktu (ms)
1	1	50822,8
2	5	132237,5
3	10	264386,2
4	15	373119,1
5	20	512709,2

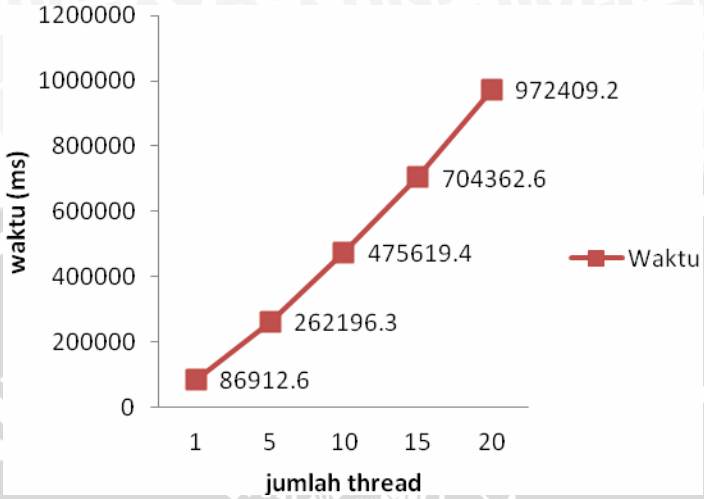


Gambar 4.9 Grafik pengaruh jumlah *thread* terhadap waktu *download* yang dibutuhkan untuk file sebesar 5 MB

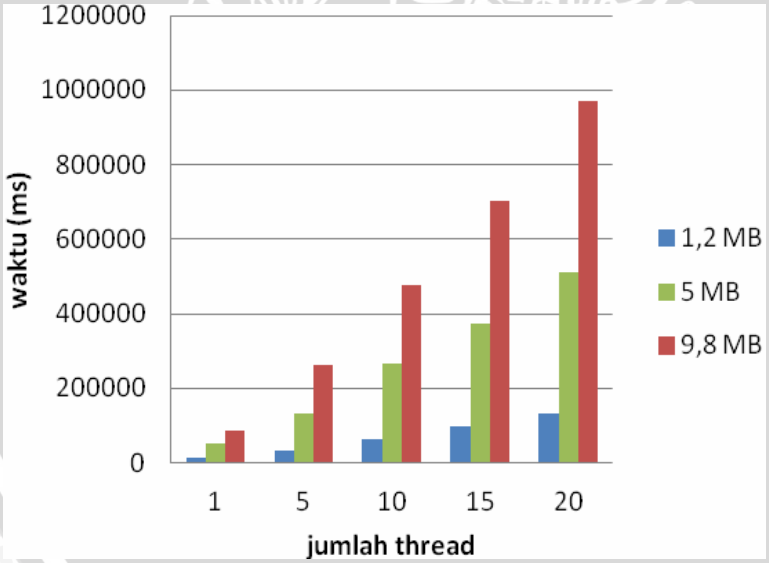
Pengujian yang terakhir dilakukan dengan men-*download file* sebesar 9,8 MB dengan jumlah *thread* sebanyak 1,5,10,15 dan 20. Hasil pengujian waktu *download* terhadap jumlah *thread* untuk *file* sebesar 9,8 MB dapat dilihat menghasilkan rata-rata waktu seperti tertera pada tabel 4.3. Rata-rata waktu *download* yang dibutuhkan selalu berbanding lurus dengan *thread* yang digunakan . Dari tabel 4.3 akan didapatkan grafik hubungan antara waktu *download* terhadap jumlah *thread* pada *file* sebesar 9,8 MB seperti pada gambar 4.10 .

Tabel 4.3 Tabel rata-rata waktu *download* terhadap jumlah *thread* untuk *file* sebesar 9,8 MB

No.	Thread	Rata-rata waktu (ms)
1	1	86912,6
2	5	262196,3
3	10	475619,4
4	15	704362,6
5	20	972409,2



Gambar 4.10 Grafik pengaruh jumlah *thread* terhadap waktu *download* yang dibutuhkan untuk file sebesar 9,8 MB



Gambar 4.11 Grafik pengaruh jumlah *thread* terhadap waktu *download* yang dibutuhkan untuk file sebesar 1,2 MB, 5 MB dan 9,8 MB

Dari semua pengujian sebelumnya seperti tertera pada tabel 4.6, 4.12 dan 4.18 akan didapat grafik hubungan atara waktu download dan thread yang digunakan untuk file sebesar 1,2 MB, 5MB dan 9,8 MB seperti pada gambar 4.11.

4.3.2 Hasil Pengujian Jumlah *Thread* Terhadap *Throughput*

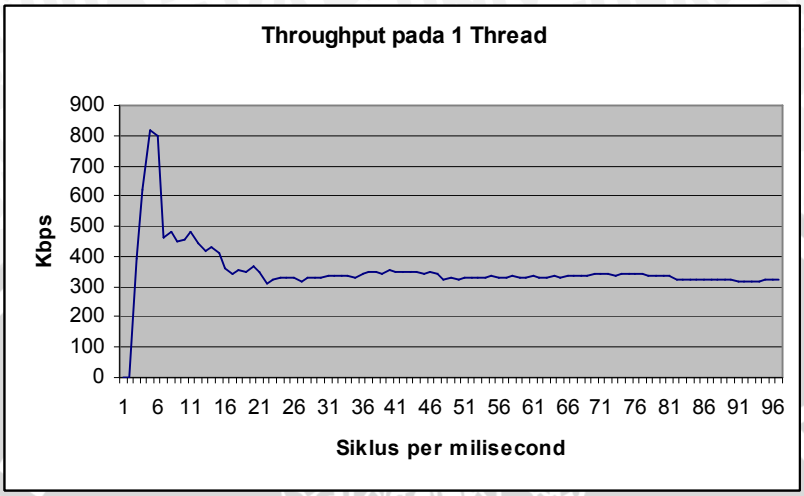
Di dalam semua pengujian yang telah dilakukan juga didapatkan data hasil pengujian berupa *throughput*. Nilai *Throughput* yang didapatkan dalam setiap pengujian (1, 5, 10, 15 dan 20 *thread*) ditampilkan dalam table 4.4. Dalam pengujian ini juga dihitung nilai akurasi bandwidth dan presentasi keakuratan berdasarkan persamaan 2.1 dan 2.2. Dalam tabel tersebut dapat diamati bahwa nilai *throughput* cenderung turun seiring dengan bertambahnya *thread* yang digunakan.

Pada pengujian jumlah *thread* terhadap *throughput* digunakan *bandwidth* sebesar 1 MB untuk melihat karakteristik dan pola *throughput* terhadap jumlah *thread.* Dengan menggunakan maksimal *bandwidth* sebesar 1 MB diharapkan apabila terdapat nilai fluktuatif *throughput* baik berupa kenaikan atau penurunan yang signifikan dapat lebih mudah diamati.

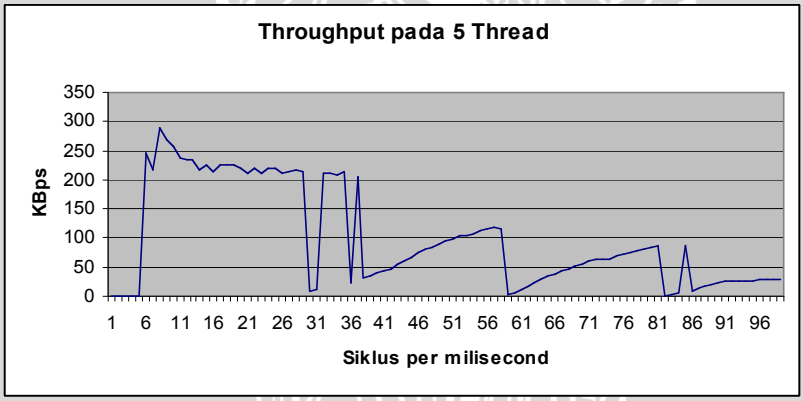
Throughput pada 1 *thread* bernilai sesuai dengan *available bandwidth* yang diberikan yaitu 1MB/s. Pada pengujian ini *throughput* rata-rata yang diterima bernilai 91,2 KBps dan pada grafik juga dapat dilihat *throughput* cenderung naik pada 20 ms pertama dan selanjutnya cenderung stabil sampai *download* selesai. Pada grafik ini tidak ditemukan adanya pencilan ataupun nilai yang fluktuatif.

Tabel 4.4 Tabel pengambilan data *throughput* terhadap jumlah thread dalam satuan KB/s (*Kilobyte per second*)

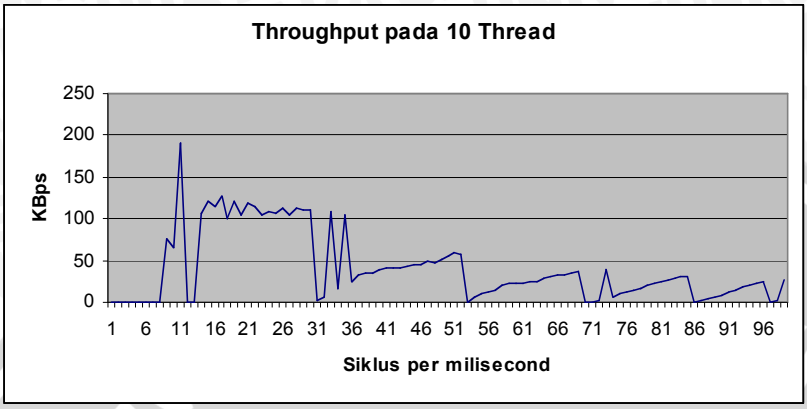
No.	Thread	Throughput (KBps)	Akurasi	Persentase (%)
1	1	91,82	8,18	91,82
2	5	80,98	19.02	80,98
3	10	40,67	59.33	40,67
4	15	27,30	72.7	27,3
5	20	11,20	88.8	11,2



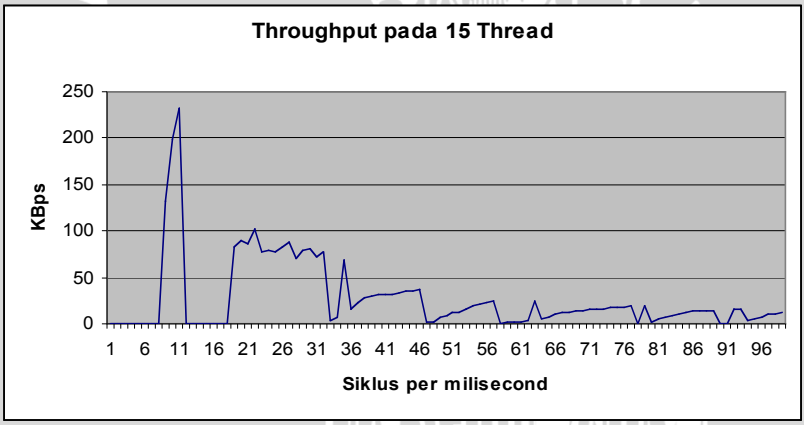
Gambar 4.12 Grafik nilai *throughput* pada proses *download* dengan menggunakan 1 *thread*



Gambar 4.13 Grafik nilai *throughput* pada proses *download* dengan menggunakan 5 *thread*



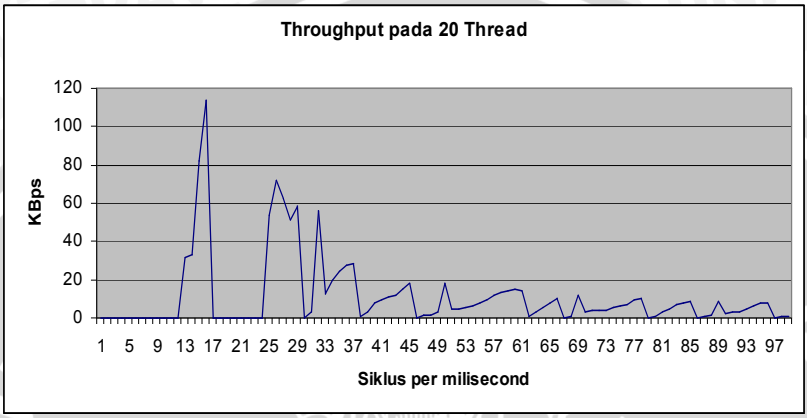
Gambar 4.14 Grafik nilai *throughput* pada proses *download* dengan menggunakan 10 *thread*



Gambar 4.15 Grafik nilai *throughput* pada proses *download* dengan menggunakan 15 *thread*

Dari tabel 4.4 juga dapat dilihat adanya penurunan akurasi *throughput* hal ini bisa terjadi karena *thread* yang dibentuk masih belum bisa menangani kemacetan jaringan. Kemacetan jaringan terjadi karena masing-masing *thread* berebut untuk mendapatkan *bandwidth* (hal ini terjadi masing-masing *thread* mendapat perlakukan yang sama atau tanpa prioritas). Seperti dijelaskan pada tinjauan pustaka, apabila terjadi kemacetan dalam jaringan yang terjadi adalah *deadlock*, yaitu suatu kondisi dimana sekelompok *node*

tidak bisa meneruskan pengiriman paket karena tidak ada *buffer* yang tersedia.



Gambar 4.16 Grafik nilai *throughput* pada proses *download* dengan menggunakan 20 *thread*

Pada 5,10,15 dan 20 *thread* grafik yang didapat berbeda dengan grafik *throughput* rata-rata pada satu *thread*. Pada grafik 4.13, 4.14, 4.15 dan 4.16 dapat dilihat terdapat nilai fluktuatif yang jumlahnya berbanding lurus dengan jumlah *thread,* semakin banyak jumlah t*hread* semakin banyak terjadi kenaikan dan penurunan *throughput*, hal ini juga ikut mempengaruhi lamanya waktu download. Nilai fluktuatif disebabkan terjadinya *context switching* pada *thread* atau terjadinya pergantian giliran *thread* untuk diproses oleh prosesor.

4.4 Analisis Hasil Pengujian

Sesuai dengan hasil pengujian sebelumnya didapat hasil yang hampir sama di tiga pengujian untuk *file* sebesar 1,2 MB, 5 MB dan 9,8 MB yang menunjukan bahwa penggunaan *thread* tidak sesuai dengan yang diharapkan. Penggunaan *thread* justru menambah waktu *download*.

Penggunaan *thread* seharusnya bisa memaksimalkan waktu *download* karena masing-masing *thread* seharusnya dapat bekerja dalam waktu yang bersamaan sehingga dengan asumsi ini apabila ada satu *thread* yang mengalami kemacetan transfer data (*congested*) maka *thread* yang lain tidak akan terpengaruh dan *thread* yang mengalami congested tersebut dapat melanjutkan proses

download kembali. Tetapi berdasarkan pengujian yang terjadi adalah sebaliknya, penambahan *thread* justru berbanding lurus dengan waktu *download.*

Hal ini bisa terjadi karena *thread* yang mengalami *congested* harus memverifikasi ulang seperti pengecekan *URL* , sedangkan di dalam pengecekan URL sendiri terjadi proses antara lain , membuka dan melakukan koneksi kembali ke port HTTP sesuai dengan tahapan yang telah dijelaskan pada subab 3.3.1 .

Manajemen *thread* juga ikut mempengaruhi penerapan penggunaan *thread*, hal ini bisa dilihat selama pengujian tidak semua *thread* dapat berjalan dalam waktu hampir secara bersamaan. Manajemen *thread* di dalam penelitian ini diserahkan sepenuhnya kepada sistem operasi sehingga pengaturan kapan *thread* harus berhenti dan kapan harus berjalan tidak sesuai dengan yang diharapkan. Penerapan *thread* yang digunakan dalam penelitian hanya sebatas layer aplikasi.

Pengujian dilakukan sesuai dengan perancangan perangkat keras pada subbab 4.1.1. Perangkat keras yang digunakan dalam pengujian menggunakan arsitektur prosesor tunggal sehingga dalam satu satuan waktu hanya dapat menjalankan satu *thread* . Salah satu pengaruh penerapan *multithreading* pada arsitektur prosesor tunggal adalah masing-masing *thread* harus diproses secara bergantian (tanpa prioritas) oleh prosesor .

JVM (*java virtual machine*) tidak menspesifikasikan penerapan penjadwalan pada *thread.* Secara umum JVM bekerja sama dengan sistem operasi untuk melakukan penjadwalan *multithreading. Thread* yang digunakan dalam aplikasi tidak menggunakan prioritas atau dengan asumsi semua thread mempunyai prioritas yang sama. JVM menggunakan model *multithreading Many-to-One,* pada model ini multithreading diterapkan pada level user atau JVM akan tetapi secara kesuluruhan *multithreading* hanya dianggap sebuah proses oleh sistem operasi.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian ini adalah :

1. Untuk mendapatkan proses yang lebih baik tidak cukup hanya menggunakan *thread* yang banyak.
2. *Thread* dalam jumlah banyak harus dimanajemen penggunaanya karena tanpa manajemen *thread* yang baik atau hanya menyerahkan manajemen *thread* kepada JVM (*java virtual machine*) dapat menyebabkan terjadinya *congestion* dalam jaringan yang berujung *deadlock.*
3. *Thread* pada java bekerja di dua level, *user thread* dan *kernel thread*. Aplikasi yang dijalankan di java dengan JVM berada pada level *user thread* dan koneksi antara JVM dengan sistem operasi berada di level *kernel thread* .

5.2 Saran

Saran yang dapat diberikan setelah melakukan penelitian ini adalah:

1. Subjek penelitian berikutnya hendaknya juga bisa diperluas dengan tidak hanya mengamati perlakuan terhadap layanan HTTP saja, tetapi layanan-layanan lainnya seperti FTP, SMTP dan lain sebagainya untuk mengetahui pengaruh adanya kemungkinan pengaruh penggunakan *multithreading* terhadap protokol yang berbeda.
2. Penelitian berikutnya juga bisa dikembangkan dengan tidak hanya menggunakan JVM, tetapi menggunkan perangkat lunak lain seperti C#, PHP dan sebagainya untuk mengetahui pengaruh pengelolaan *thread* pada aplikasi yang berbeda.

DAFTAR PUSTAKA

Casad, J. 2003. *Sams Teach Yourself TCP-IP in 24 Hours*. Sams Publishing.USA

Carver, R . 2006. *Modern Multithreading*. John Wiley & sons. New Jersey

Darma, Jarot S.2010, Shenia A, Buku Pintar Menguasai Internet, halaman 416. Mediakita. Indonesia

Gunawan, A.H. 2003. *Quality of Service* dalam Data Komunikasi. *http://www.gematel.com/Gematel 40 - Analisis Teknologi - Quality of Service dalam Data Komunikasi.htm*., tanggal akses : 13 Maret 2006

Jim, K. , Ross, K. and Wesley, A. 2002. *Computer Networking: A Top Down Approach Featuring the Internet*, 2nd edition. Addison Wesley. USA.

Lindholm,tim and Frank Yellin.1999 *The Java™ Virtual Machine Specification, 2nd Edition*. Sun Microsystem. California.USA

Risso, F dan Gevros P. 1999. Operational and Performance Issues of a CBQ Router. University College. London.

Shanthi, M dan Irudhayaraj, A.A.2009. *Multithreading - An Efficient Technique for Enhancing Application Performance*. *International Journal of Recent Trends in Engineering*, Vol 2. Academy Publisher.

Stallings, W. 2003. *Data and Computer Communications, 7th Edition*. Prentice Hall. USA.

Tanenbaum, A. S.. 2003. *Computer Networks, 4th Edition*. Prentice Hall. USA.

Tim Penelitian dan Pengembangan Wahana Komputer. 2003. Konsep Jaringan Komputer dan Pengembangannya. Salemba Infotek. Jakarta

LAMPIRAN

Lampiran 1. Pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 1,2 MB, 5 MB dan 9,8 MB dengan *thread* sejumlah 1, 5, 10, 15 dan 20

Tabel 1.1 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 1,2 MB dengan *thread s*ejumlah 1

Uji ke-	File (MB)	n thread	t (ms)
1	1,2	1	14495
2	1,2	1	13181
3	1,2	1	14233
4	1,2	1	13280
5	1,2	1	14268
6	1,2	1	14376
7	1,2	1	14243
8	1,2	1	12265
9	1,2	1	14613
10	1,2	1	14412
Rata-rata			13936,6

Tabel 1.2 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 1,2 MB dengan *thread s*ejumlah 5

Uji ke-	File (MB)	n thread	t (ms)
1	1,2	5	32696
2	1,2	5	33735
3	1,2	5	31298
4	1,2	5	32225
5	1,2	5	33006
6	1,2	5	32928
7	1,2	5	32230
8	1,2	5	33621
9	1,2	5	32108

10	1,2	5	33173
Rata-rata			32702

Tabel 1.3 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 1,2 MB dengan *thread s*ejumlah 10

Uji ke-	File (MB)	n thread	t (ms)
1	1,2	10	58546
2	1,2	10	70411
3	1,2	10	59188
4	1,2	10	80596
5	1,2	10	60256
6	1,2	10	61471
7	1,2	10	67156
8	1,2	10	59086
9	1,2	10	58396
10	1,2	10	58178
Rata-rata			63328,4

Tabel 1.4 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 1,2 MB dengan *thread s*ejumlah 15

Uji ke-	File (MB)	n thread	t (ms)
1	1,2	15	89746
2	1,2	15	86761
3	1,2	15	92756
4	1,2	15	85939
5	1,2	15	90696
6	1,2	15	137267
7	1,2	15	95881
8	1,2	15	106524
9	1,2	15	109797
10	1,2	15	88382
Rata-rata			98374,9

Tabel 1.5 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 1,2 MB dengan *thread s*ejumlah 20

Uji ke-	File (MB)	n thread	t (ms)
1	1,2	20	135020
2	1,2	20	142322
3	1,2	20	129172
4	1,2	20	133465
5	1,2	20	149222
6	1,2	20	110984
7	1,2	20	138674
8	1,2	20	113686
9	1,2	20	134462
10	1,2	20	127876
Rata-rata			131488,3

Tabel 1.6 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 5 MB dengan *thread s*ejumlah 1

Uji ke-	File (MB)	n thread	t (ms)
1	5	1	49633
2	5	1	49676
3	5	1	49678
4	5	1	53946
5	5	1	50834
6	5	1	49786
7	5	1	52097
8	5	1	51714
9	5	1	50414
10	5	1	50450
Rata-rata			50822,8

Tabel 1.7 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 5 MB dengan *thread s*ejumlah 5

Uji ke-	File (MB)	n thread	t (ms)
1	5	5	131440
2	5	5	133243
3	5	5	131551
4	5	5	133243
5	5	5	130765
6	5	5	131996
7	5	5	133481
8	5	5	132199
9	5	5	132140
10	5	5	132317
Rata-rata			132237,5

Tabel 1.8 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 5MB dengan *thread s*ejumlah 10

Uji ke-	File (MB)	n thread	t (ms)
1	5	10	254651
2	5	10	339714
3	5	10	247829
4	5	10	250681
5	5	10	251241
6	5	10	265411
7	5	10	258818
8	5	10	261514
9	5	10	243312
10	5	10	270691
Rata-rata			264386,2

Tabel 1.9 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 5 MB dengan *thread s*ejumlah 15

Uji ke-	File (MB)	n thread	t (ms)
1	5	15	364546
2	5	15	365741
3	5	15	431036
4	5	15	368658
5	5	15	378891
6	5	15	356675
7	5	15	367565
8	5	15	352123
9	5	15	367844
10	5	15	378112
Rata-rata			373119,1

Tabel 1.10 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 5 MB dengan *thread s*ejumlah 20

Uji ke-	File (MB)	n thread	t (ms)
1	5	20	468567
2	5	20	555621
3	5	20	550818
4	5	20	692172
5	5	20	499311
6	5	20	486744
7	5	20	465639
8	5	20	467222
9	5	20	471121
10	5	20	469877
Rata-rata			512709,2

Tabel 1.11 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 9,8 MB dengan *thread s*ejumlah 1

Uji ke-	File (MB)	n thread	t (ms)
1	9,8	1	87121
2	9,8	1	87201
3	9,8	1	87064
4	9,8	1	87279
5	9,8	1	87276
6	9,8	1	87543
7	9,8	1	86742
8	9,8	1	85676
9	9,8	1	86112
10	9,8	1	87112
Rata-rata			86912,6

Tabel 1.12 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 9,8 MB dengan *thread s*ejumlah 5

Uji ke-	File (MB)	n thread	t (ms)
1	9,8	5	261235
2	9,8	5	261378
3	9,8	5	261066
4	9,8	5	261352
5	9,8	5	261011
6	9,8	5	261241
7	9,8	5	278322
8	9,8	5	250098
9	9,8	5	260826
10	9,8	5	265434
Rata-rata			262196,3

Tabel 1.13 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 9,8 MB dengan *thread s*ejumlah 10

Uji ke-	File (MB)	n thread	t (ms)
1	9,8	10	478572
2	9,8	10	479387
3	9,8	10	479284
4	9,8	10	478608
5	9,8	10	478828
6	9,8	10	473111
7	9,8	10	476199
8	9,8	10	469984
9	9,8	10	474443
10	9,8	10	467778
Rata-rata			475619,4

Tabel 1.14 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 9,8 MB dengan *thread s*ejumlah 15

Uji ke-	File (MB)	n thread	t (ms)
1	9,8	15	704708
2	9,8	15	710328
3	9,8	15	731441
4	9,8	15	697082
5	9,8	15	696774
6	9,8	15	701241
7	9,8	15	697114
8	9,8	15	709812
9	9,8	15	696951
10	9,8	15	698175
Rata-rata			704362,6

Tabel 1.15 Tabel pengambilan data untuk mekanisme percobaan dengan ukuran file sebesar 9,8 MB dengan *thread s*ejumlah 20

Uji ke-	File (MB)	n thread	t (ms)
1	9,8	20	947415
2	9,8	20	985680
3	9,8	20	979730
4	9,8	20	953214
5	9,8	20	978323
6	9,8	20	961632
7	9,8	20	974322
8	9,8	20	987124
9	9,8	20	986631
10	9,8	20	970021
Rata-rata			972409,2