SISTEM TERDISTRIBUSI

Chapter 1

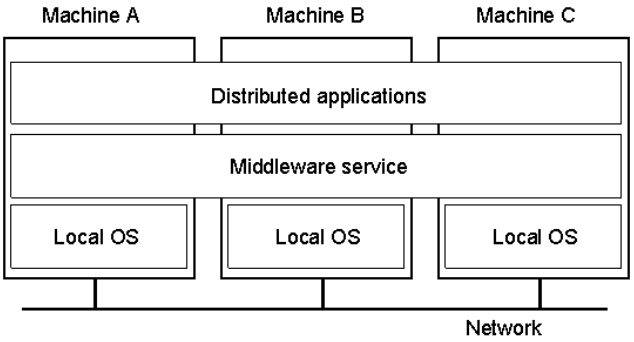
Introduction

I Made Andhika, S.Kom
dre_andhika@yahoo.co.id

Definisi Sistem Terdistribusi

- Sistem Terdistribusi adalah Kumpulan komputer independen yang tampak oleh pengguna sebagai sebuah sistem (komputer) tunggal yang koheren (saling berhubungan).
- Sebuah sistem dimana komponen software atau hardware-nya terletak di dalam jaringan komputer dan saling berkomunikasi menggunakan *message pasing*
- Sebuah sistem yang tersusun oleh dua atau lebih komputer dan memiliki koordinasi proses melalui pertukaran pesan sinkron atau asinkron
- Kumpulan komputer autonom yang dihubungkan oleh jaringan dengan software yang dirancang untuk menghasilkan fasilitas komputasi terintegrasi

Definisi Sistem Terdistribusi



- Sebuah sistem terdistribusi diatur sebagai middleware. Perlu diketahui bahwa lapisan middleware meluas sampai ke beberapa mesin

Karakteristik Sistem Terdistribusi

1. No Global Clock
   - Terdapat batasan pada ketepatan proses sinkronisasi clock pada sistem terdistribusi, oleh karena *asynchronous message passing*
   - Pada sistem terdistribusi, tidak ada satu proses tunggal yang mengetahui *global state system* saat ini (disebabkan oleh *concurrency* dan *message passing)*
2. Independent Failure
   - Kemungkinan adanya kegagalan proses tunggal yang tidak diketahui
   - Proses tunggal mungkin tidak mempengaruhi kegagalan keseluruhan sistem
3. Concurrency of Components
   - Contoh : beberapa pemakai *browser* mengakses suatu halaman web secara bersamaan
   - Bagaimana jika ada operasi update?

Alasan menggunakan Sistem Terdistribusi

- *Performance*
  - Sekumpulan processor dapat menyediakan kinerja yang lebih tinggi daripada komputer yang terpusat
- *Distribution*
  - Banyak aplikasi yang terlibat, sehingga lebih baik jika dipisah dalam mesin yang berbeda. (e.g : aplikasi perbankan, komersial)
- *Reliability*
  - Jika terjadi kerusakan pada salah satu mesin, tidak akan mempengaruhi kinerja sistem secara keseluruhan
- *Incremental Growth*
  - Mesin baru dapat ditambahkan jika kebutuhan proses meningkat
- *Sharing data / Resources*

  Resources adalah :
  - Segala hal yang dapat digunakan bersama dalam jaringan komputer
  - Meliputi *hardware* dan *software*
- *Communication*
  - *Menyediakan fasilitas komunikasi antar manusia*
- Ekonomis
  - Kumpulan mikroprosesor memberikan harga/unjuk kerja yang lebih baik dibandingkan dengan mainframe

Model Sistem Terdistribusi

1. Sistem Client – Server
   - Merupakan bagian dari model sistem terdistribusi yang membagi jaringan berdasarkan pemberi dan penerima jasa layanan
2. Sistem point to point
   - Merupakan bagian dari model sistem terdistribusi dimana sistem dapat sekaligus berfungsi sebagai *client* maupun *server*
3. Sistem terkluster
   - Adalah gabungan dari beberapa sistem individual (komputer) yang dikumpulkan pada suatu lokasi, saling berbagi tempat penyimpanan data (*storage*), dan saling terhubung dalam jaringan lokal (*Local Area Network*)

Masalah dalam Sistem Terdistribusi

- Perangkat Lunak
  - Tingkat kesulitan lebih tinggi dalam merancang program dalam Sistem Terdistribusi
- Ketergantungan pada infrastruktur jaringan (World Wide Wait…)
- Kemudahan akses sumber daya yang di-share memunculkan masalah keamanan

Tantangan dalam Sistem Terdistribusi

- Heterogenitas:
  - Jaringan
  - Hardware
  - Software
  - Language
- Solusi:
  - CORBA (Common Object Request Broker Architecture) adalah sebuah arsitektur software yang berbasis pada teknologi berorientasi obyek atau *Object Oriented* (OO) dengan paradigma *client-server*.
  - RMI (Remote Method Invocation) adalah cara programmer Java untuk membuat program aplikasi Java to Java yang terdistribusi. Program-program yang menggunakan RMI bisa menjalankan metode secara jarak jauh, sehingga program dari server bisa menjalankan method di komputer klien, dan begitu juga sebaliknya

Tantangan (2)

- Openness:
  - Mengikuti standard interface
- Security:
  - Privacy
  - Authentication
  - Availability
- Scalability
- Fault handling:
  - Detection
  - Retransmission
  - Redundancy
    - Rute
    - Replikasi data pada beberapa mesin
- Concurrency:
  - Penjadwalan
  - Deadlock avoidance

Transparency (menyembunyikan keragaman sistem)

Transparency	Deskripsi
Access	User menganggap bahwa semua resource adalah lokal
Location	User tidak perlumengetahui lokasi resurce
Migration / Mobility	Kemampuan melakukan relokasi sumber daya tanpa konfigurasi ulang oleh user
Relocation	User tidak perlu mengetahui perpindahan sumber daya ketika sedang digunakan
Replication	user tidak perlu mengatur replikasi data
Concurrency	user tidak perlu mengetahui keberadaan sistem paralel
Failure	User tidak perlu mengetahui kegagalan dan recovery sistem
persistence	User tidak perlu mengetahui tentang sumber daya (software) terdapat di memori atau disk

Berbagai bentuk transparansi dalam sistem terdistribusi

Contoh Transparency

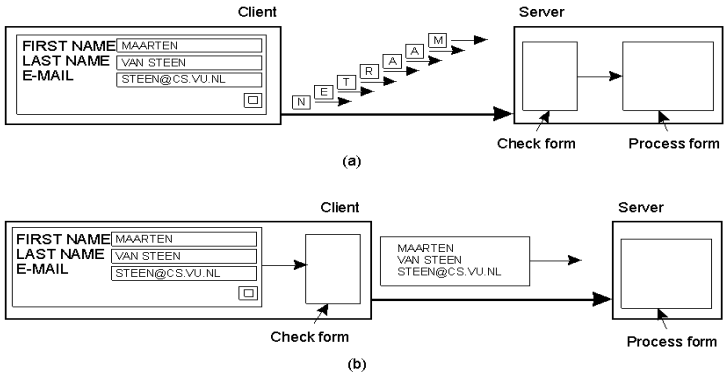
- http://www.google.com
  - Dimana lokasi fisiknya?
  - Tersebar dalam server farm?
- Pengguna HP tetap dapat berkomunikasi tanpa putus walaupun berpindah lokasi
- Akses ke tabel yang sama oleh beberapa user sekaligus
- Replikasi yahoo web server ke beberapa lokasi geografis yang lebih dekat dengan user
- User tidak mengetahui kapan data dipindah dari disk ke memori

Scalability Problem

- Masalah kinerja yg disebabkan oleh kapasitas server dan jaringan
- Tiga teknik scaling :
  - Hiding communication latency
  - Distribusi
  - Replikasi
- Contoh keterbatasan skalabilitas

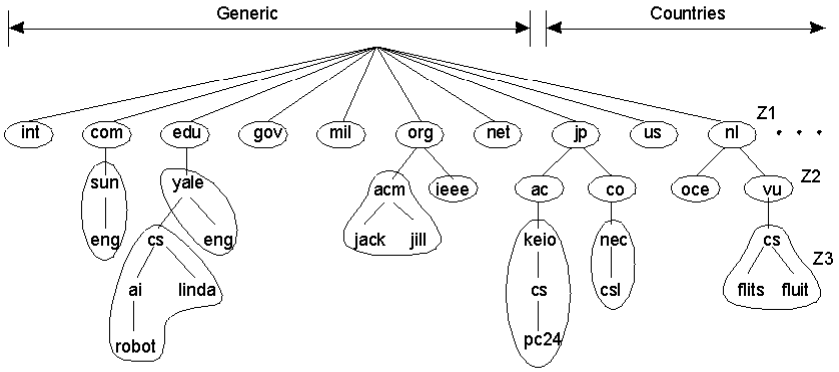
Konsep	Contoh
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized Algoritms	Doing routing based on complete information

Teknik Scaling



- Perbedaan antara mengijinkan :
  a. Server atau
  b. Client yang mengecek form setelah diisi

Teknik Scaling

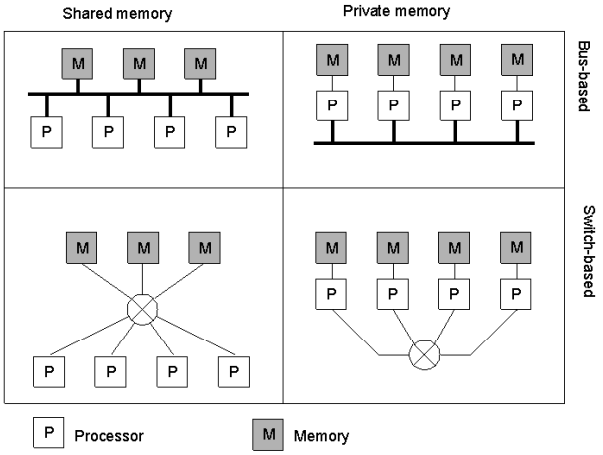


- An example of dividing the DNS name space into zones

Tujuan Sistem Terdistribusi

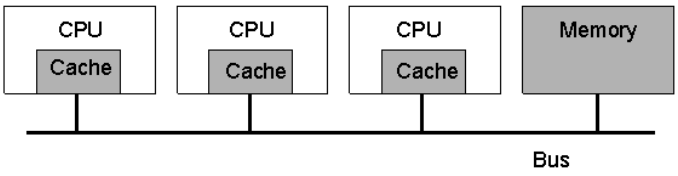
- Konektivitas user ke sumber daya (dalam hal ini data dan informasi), yaitu mempermudah user untuk mengakses dan men-sharing sumber daya dengan menggunakan sistem kendali.
- *Openness* (keterbukaan), yaitu sistem memberikan suatu layanan sesuai dengan aturan yang digambarkan melalui sintak dan semantik dari layanan tersebut
- *Scalability* (skalabilitas)
- *Transparancy* (transparansi)

Konsep Hardware



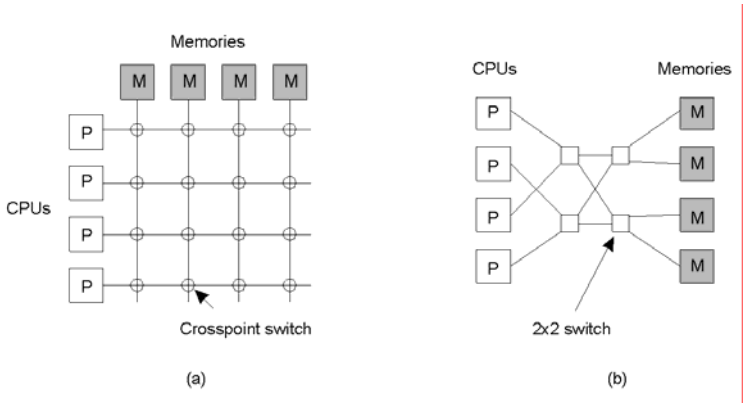
- Perbedaan mendasar antara organisasi dan memori dalam sistem komputer terdistribusi

Konsep Hardware



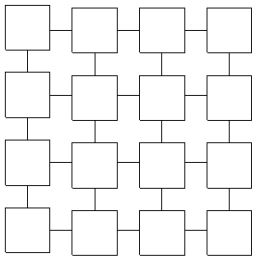
- Multiprosesor berbasis bus

Konsep Hardware

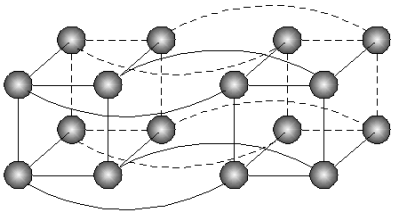


a. A crossbar switch
b. An omega switching network

Konsep Hardware



(a)



(b)

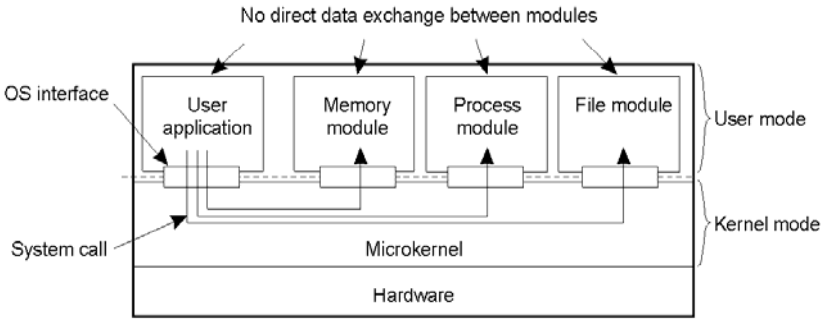
- Sistem multikomputer homogen
  a. Grid
  b. hypercube

Konsep Software

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

- An overview of
- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

Uniprocessor Operating Systems



- Separating applications from operating system code through
- a microkernel.

Multiprocessor Operating Systems (1)

```
monitor Counter {
private:
 int count = 0;
public:
  int value() { return count;}
  void incr () { count = count + 1;}
  void decr() { count = count – 1;}
}
```

A monitor to protect an integer against concurrent access.

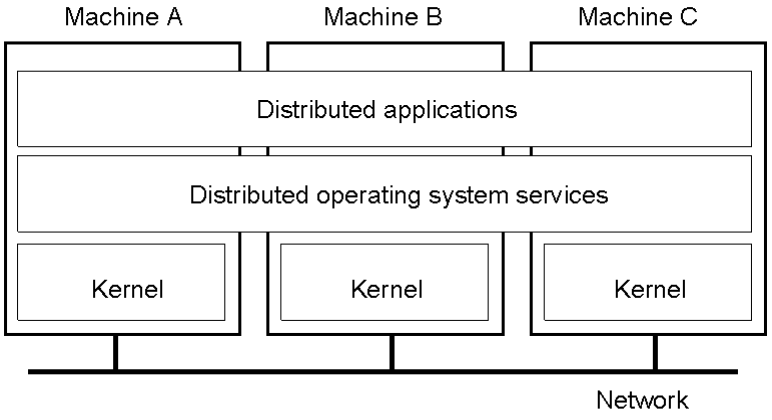
Multiprocessor Operating Systems (2)

```
monitor Counter {
private:
  int count = 0;
  int blocked_procs = 0;
  condition unblocked;
public:
  int value () { return count;}
  void incr () {
       if (blocked_procs == 0)
         count = count + 1;
       else
         signal (unblocked);
}
```

```
void decr() {
if (count ==0) {
  blocked_procs = blocked_procs + 1;
  wait (unblocked);
  blocked_procs = blocked_procs – 1;
   }
  else
   count = count – 1;
 }
}
```

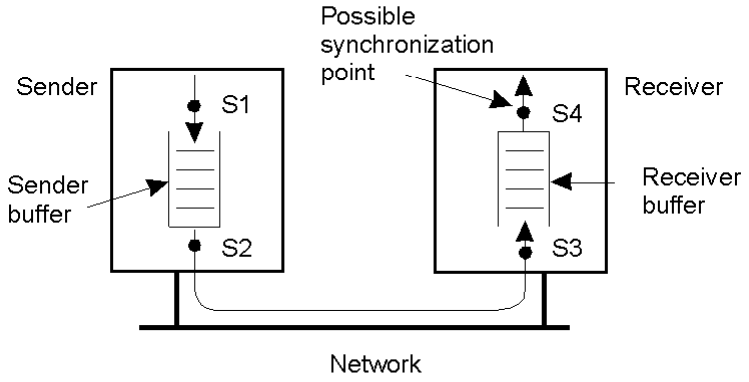
A monitor to protect an integer against concurrent access, but blocking a process.

Multicomputer Operating Systems (1)



General structure of a multicomputer operating system

Multicomputer Operating Systems (2)



Alternatives for blocking and buffering in message passing.

Multicomputer Operating Systems (3)

Synchronization point	Send buffer	Reliable comm. guaranteed?
Block sender until buffer not full	Yes	Not necessary
Block sender until message sent	No	Not necessary
Block sender until message received	No	Necessary
Block sender until message delivered	No	Necessary

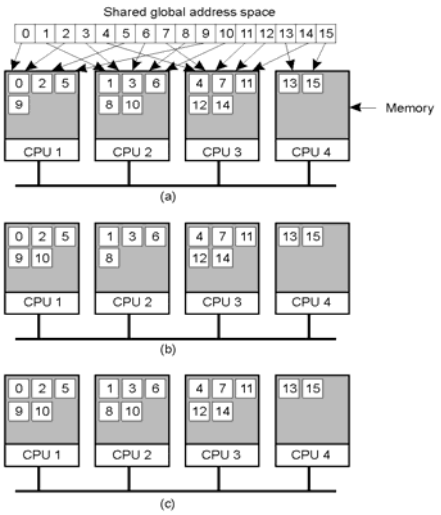
Relation between blocking, buffering, and reliable communications.

Distributed Shared Memory Systems (1)

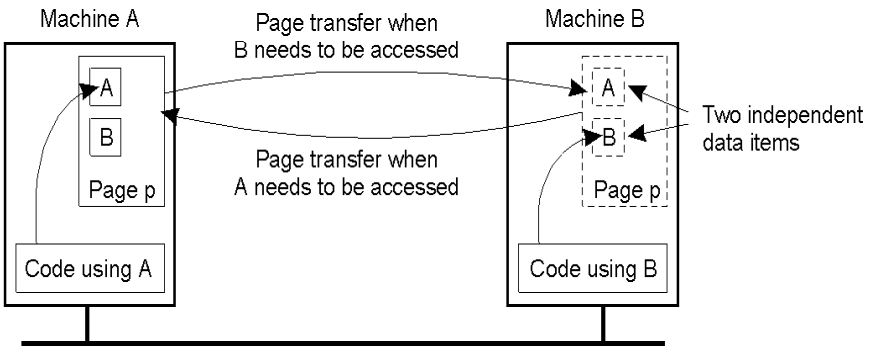
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used

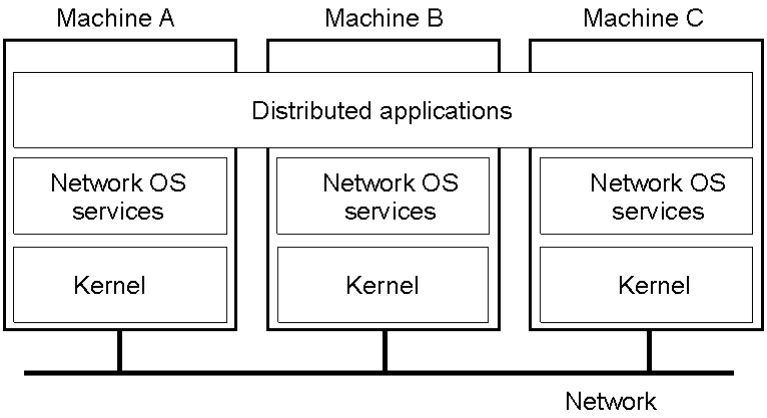


Distributed Shared Memory Systems (2)



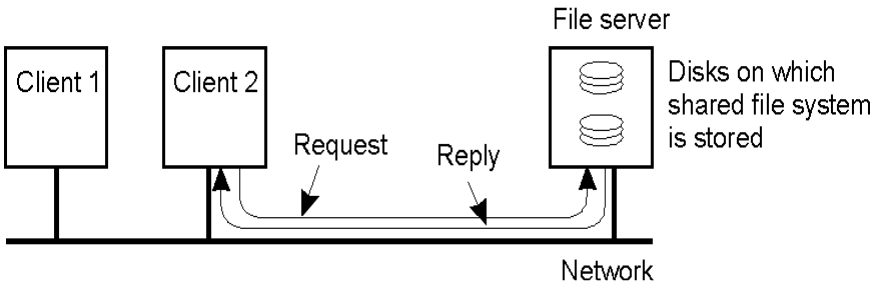
False sharing of a page between two independent processes.

Network Operating System (1)



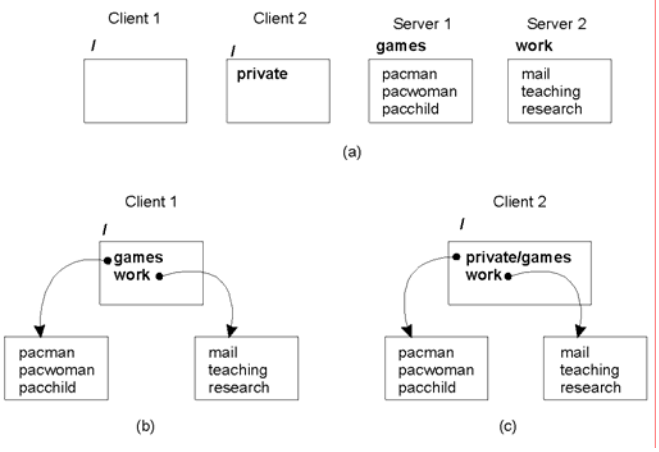
General structure of a network operating system.

Network Operating System (2)



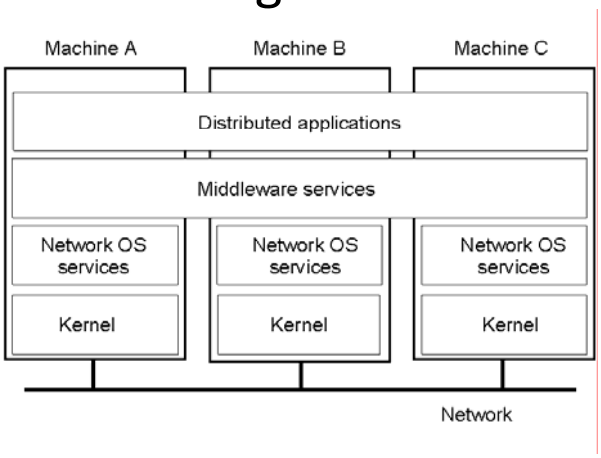
Two clients and a server in a network operating system.

Network Operating System (3)



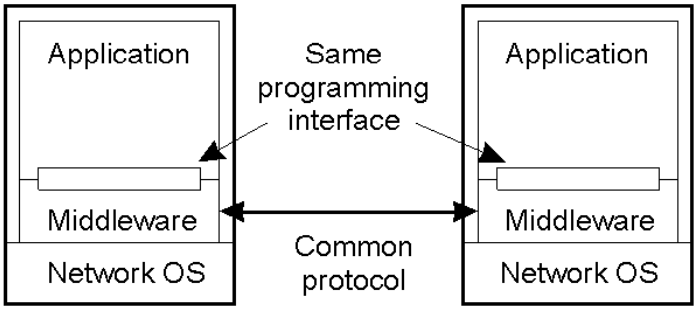
Different clients may mount the servers in different places.

Positioning Middleware



General structure of a distributed system as middleware.

Middleware and Openness



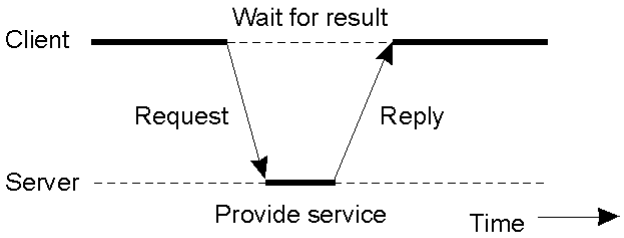
In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.

Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed systems.

Clients and Servers



General interaction between a client and a server.

An Example Client and Server (1)

```
/* Definitions needed by clients and servers.                    */
#define TRUE             1
#define MAX_PATH         255    /* maximum length of file name      */
#define BUF_SIZE         1024   /* how much data to transfer at once */
#define FILE_SERVER      243    /* file server's network address    */

/* Definitions of the allowed operations */
#define CREATE           1      /* create a new file                */
#define READ             2      /* read data from a file and return it */
#define WRITE            3      /* write data to a file             */
#define DELETE           4      /* delete an existing file          */

/* Error codes. */
#define OK               0      /* operation performed correctly    */
#define E_BAD_OPCODE     -1     /* unknown operation requested      */
#define E_BAD_PARAM      -2     /* error in a parameter             */
#define E_IO             -3     /* disk error or other I/O error    */

/* Definition of the message format. */
struct message {
     long source;               /* sender's identity                */
     long dest;                 /* receiver's identity              */
     long opcode;               /* requested operation              */
     long count;                /* number of bytes to transfer      */
     long offset;               /* position in file to start I/O    */
     long result;               /* result of the operation          */
     char name[MAX_PATH];       /* name of file being operated on   */
     char data[BUF_SIZE];       /* data to be read or written       */
};
```

The *header.h* file used by the client and server.

An Example Client and Server (2)

```
#include <header.h>
void main(void) {
     struct message ml, m2;                 /* incoming and outgoing messages        */
     int r;                                 /* result code                           */

     while(TRUE) {                          /* server runs forever                   */
          receive(FILE_SERVER, &ml);        /* block waiting for a message           */
          switch(ml.opcode) {               /* dispatch on type of request           */
               case CREATE:    r = do_create(&ml, &m2); break;
               case READ:      r = do_read(&ml, &m2); break;
               case WRITE:     r = do_write(&ml, &m2); break;
               case DELETE:    r = do_delete(&ml, &m2); break;
               default:        r = E_BAD_OPCODE;
          }
          m2.result = r;                    /* return result to client               */
          send(ml.source, &m2);             /* send reply                            */
     }
}
```

A sample server.

An Example Client and Server (3)

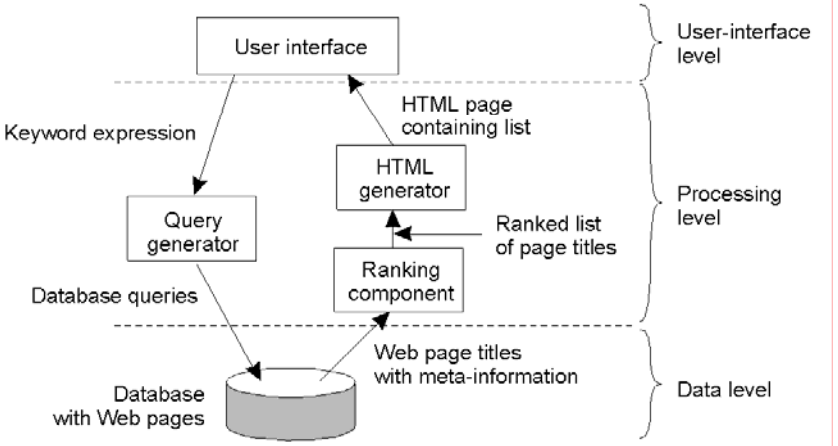
```
#include <header.h>
int copy(char *src, char *dst){             /* procedure to copy file using the server  */
     struct message ml;                     /* message buffer                           */
     long position;                         /* current file position                    */
     long client = 110;                     /* client's address                         */

     initialize( );                         /* prepare for execution                    */
     position = 0;
     do {
          ml.opcode = READ;                 /* operation is a read                      */
          ml.offset = position;             /* current position in the file             */
          ml.count  = BUF_SIZE;                                      /* how many bytes to read*/
          strcpy(&ml.name, src);            /* copy name of file to be read to message  */
          send(FILESERVER, &ml);            /* send the message to the file server      */
          receive(client, &ml);             /* block waiting for the reply              */

          /* Write the data just received to the destination file.                      */
          ml.opcode = WRITE;                /* operation is a write                     */
          ml.offset = position;             /* current position in the file             */
          ml.count  = ml.result;            /* how many bytes to write                  */
          strcpy(&ml.name, dst);            /* copy name of file to be written to buf   */
          send(FILE_SERVER, &ml);           /* send the message to the file server      */
          receive(client, &ml);             /* block waiting for the reply              */
          position += ml.result;            /* ml.result is number of bytes written     */
     } while( ml.result > 0 );              /* iterate until done                       */
     return(ml.result >= 0 ? OK : ml result);  /* return OK or error code            */
}
```

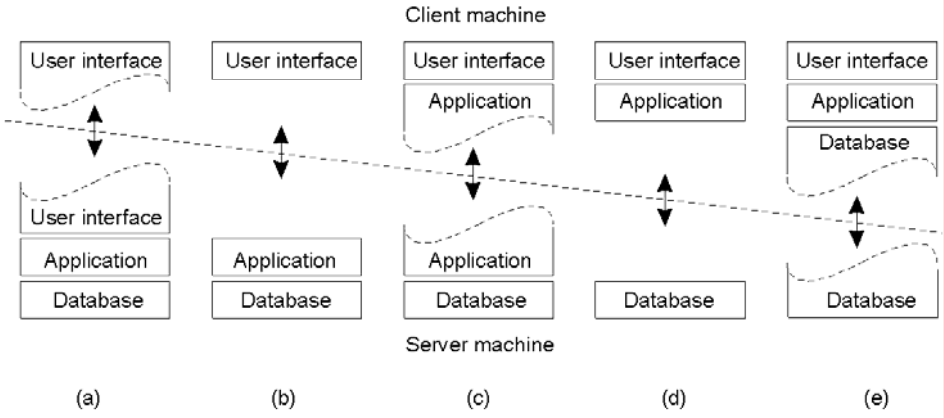
A client using the server to copy a file.

Processing Level



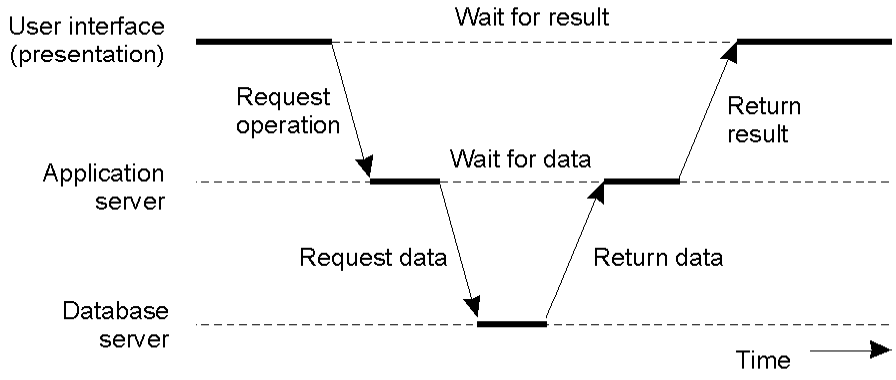
The general organization of an Internet search engine into three different layers

Multitiered Architectures (1)



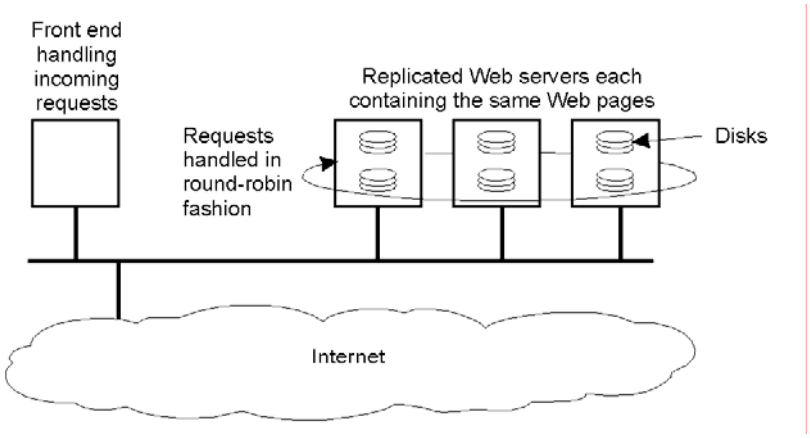
Alternative client-server organizations (a) – (e).

Multitiered Architectures (2)



An example of a server acting as a client.

Modern Architectures



An example of horizontal distribution of a Web service.

Daftar Pustaka

- Andrew S. Tanenbaum, Maarten van Steen, "Distributed Systems : Principles and Paradigms", 1st Edition, 2002
- George Coulouris, Jean Dollimore, Tim Kindberg, "Distributed Systems : Concepts and Design", 3rd Edition, 2000
- Andrew S. Tanenbaum, "Distributed Operating System", 1994
- http://www.komputasi.lipi.go.id/utama.cgi?cetakartikel&1080002265
- http://kambing.ui.ac.id/bebas/v09/onno-ind-1/application/sekilas-tentang-java-rmi-1998.rtf