Network Programming

Dr. Thaier Hayajneh
Computer Engineering Department

Client-Server Model

Outline

- learn about Client- server model
- Concurrency
- Processes
- Examples

CLIENT-SERVER MODEL

Introduction

- What is the purpose of computer network?
- Provide services to users. – remote user - - it must run a program. Program that request service from another program. One to provide a service and one to request a service.
- In the Internet the application programs are the entities that communicate with each other, not the computers or users.

Question that may be asked? (1)

- Should both application programs be able to request services and provide services?
- Usually a client with an application program running on a local machine requests a service from another application program called server running on the remote machine. (requesting --- providing)

Question that may be asked? (2)

- Should an application program provide services only to one specific application program installed somewhere in an internet or should it provide services for any application program that requests this service.
- Client-server relationship is many-to-one -- many clients use services of one server.

Question that may be asked? (3)

- When should an application program be running ?
- All the time?? Just when there is a need for the service?
- A client should run only when needed to request a service.
- A server should run all the time because it does not know when its service will be needed.

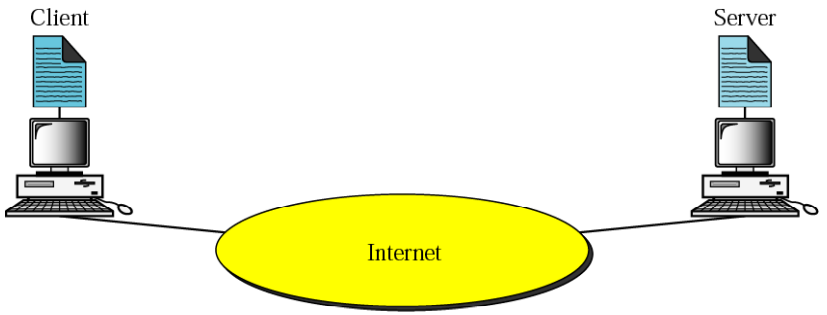
Question that may be asked? (4)

- Should there be only one universal application program that can provide any type of service?
- In TCP/IP services needed frequently and by many users have specific client-server application programs.

Client - Server

- A client is a process that is running on the local machine sends a request to an existing server requesting service and (usually) waits for a reply.
- A client program is finite:
  - Started by the user and terminates when the service is complete.
- Active open: the client opens the communication channel using the IP address of (?) and the (?) port number
- Active close: client closes the communication channel.

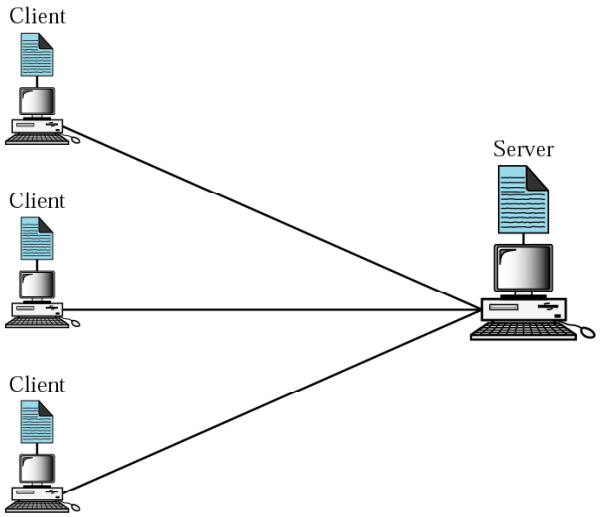
Client-server model



Client - Server

- A *server* is a process - not a machine !
- A server is a program running on the remote machine providing service to the clients.
- Passive open: when it starts it opens the door for incoming requests from clients, but it never initiates a service until it is requested to do so.
- A server is an infinite program: when it starts it runs infinitely unless a problem arises
- A server waits for a request from a client then it responds to the request either interactively or concurrently.

Client-server relationship



Client - Server Examples

- Server returns the time-of-day.
- Server returns a document.
- Server prints a file for client.
- Server does a disk read or write.
- Server records a transaction.

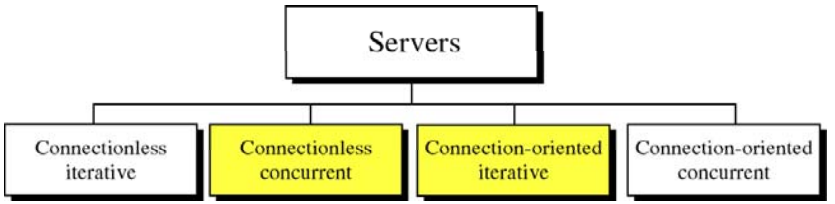
CONCURRENCY

Concurrency in Clients

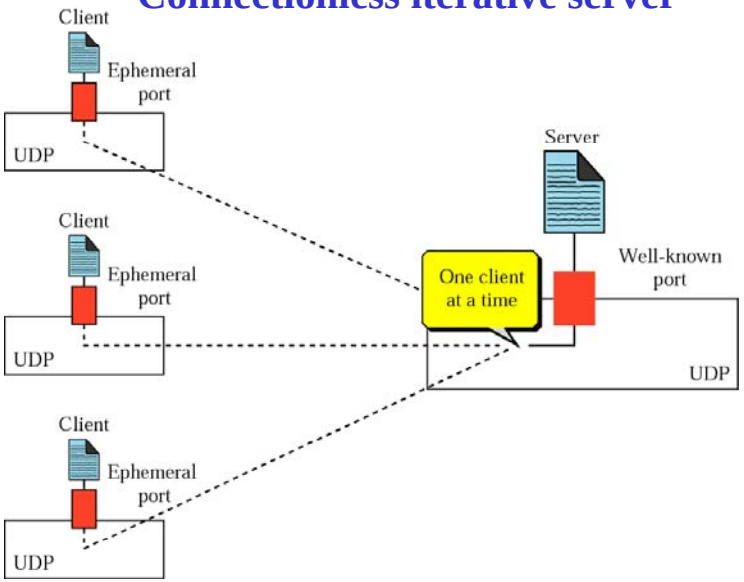
- A *client can run on a machine iteratively or concurrently.*
- Iteratively means running them one by one (one client must start, run, and terminate before the machine can start another client)
- Concurrent: two or more clients can run at the same time

Server types

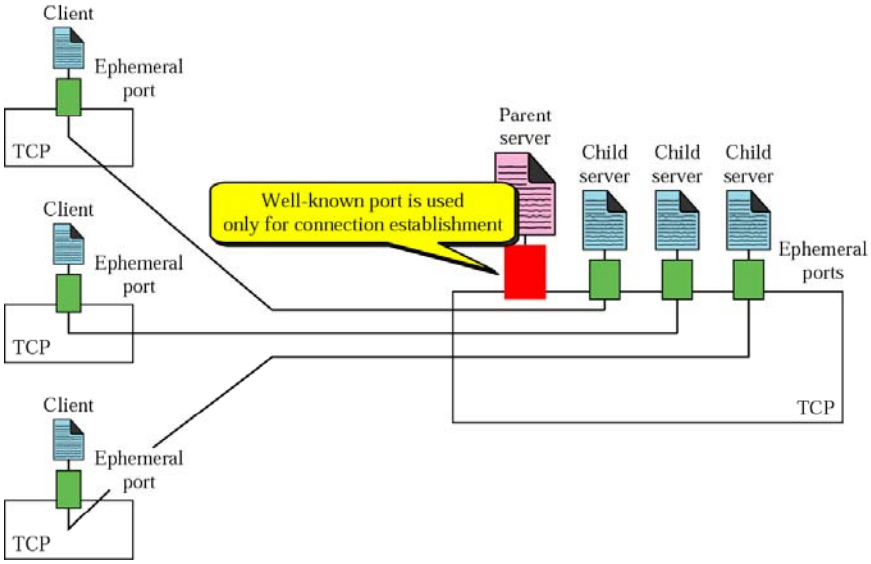
- Iterative server can process only one request at a time, it receives it, process it, and respond to it then handles new one.
- Concurrent server can process many requests at the same time and thus can share its time between many requests.



Connectionless iterative server



Connection-oriented concurrent server



PROCESSES

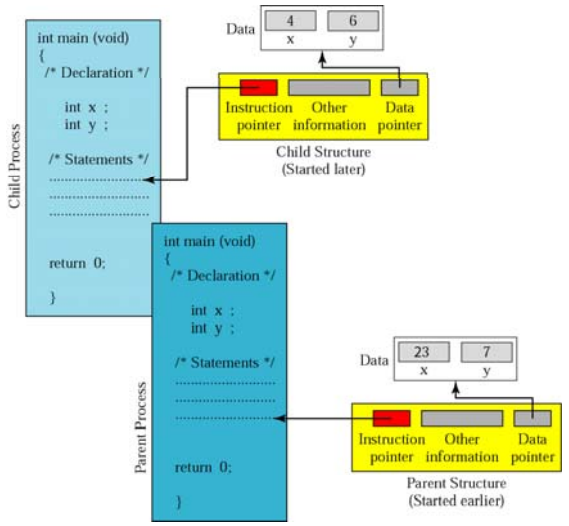
Programs & Processes

- In UNIX A program is different than process
- A *program* is an executable file (code).
- A *process* or *task* is an instance of a program that is being executed.
- A single program can generate multiple processes.

Programs & Processes

- The operating system associates a structure with a process that holds information needed to control a process.
- The structure holds:
  - Pointer to the line of the program being executed
  - Processid
  - Userid
  - Program name
  - Pointer to variable and data needed for the process

Programs and processes



Prototype for the getpid function

- Each process in UNIX is uniquely defined by an integer called the process identification number.
- The following function will return the pid number of a process.
- Pid_t is a data type that is often cast to long integer.

```
pid_t      getpid (void) ;
```


A program that prints its own processid

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{
  printf ("My process id  is %ld", (long) getpid());

  return  0;
}
```

Prototype for the fork function

- In UNIX a process can be created only by a parent process. The created process is called a child process.
- This requires some invocation in the program code to trigger the process.
- fork creates a child process that has the same image as its parents.
- Both parent and child executes all remaining lines of code

```
pid_t      fork (void) ;
```


A program with one parent and one child

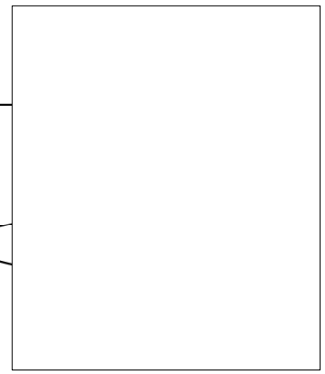
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{
  printf ("Hello World\n") ;

  fork () ;

  printf ("Bye World\n") ;

  return  0;
}
```



A program with two fork functions

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{

  printf ("Parent\n") ;
  fork () ;
  printf ("
  fork () ;
  printf ("

  return  0;
}
```


The output of the previous program

```
Parent

Parent and first child
Parent and first child

Parent, first child, second child, and grandchild
Parent, first child, second child, and grandchild
Parent, first child, second child, and grandchild
Parent, first child, second child, and grandchild
```

A program that prints the processids of the parent and the child

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main (void)
{
  pid_t  pid ;
  pid  = fork () ;
  if (pid ! = 0 )
     printf ("Parent process id  is %ld", (long) getpid());
  else
    printf ("Child process id  is %ld", (long) getpid());
 return  0;
}
```


Example of a server program with parent and child processes