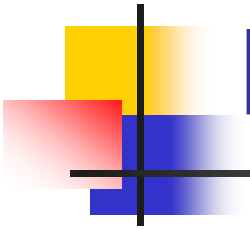
Network Socket Programming - I

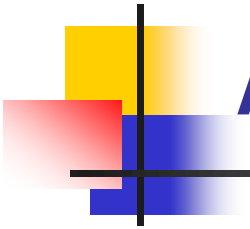
BUPT/QMUL

2021-03-11



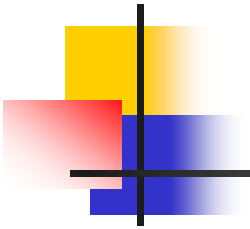
Review

- Introduction
  - What is the Internet?
  - How does it work?
  - When & how did it come about?
  - Who controls it?
  - Where is it going?
- Basic network definitions
  - Terms for Network Devices
  - Terms for Network Performance Parameters
  - Ways to connect to the Internet
  - Terms for Network Types
- Layered architecture



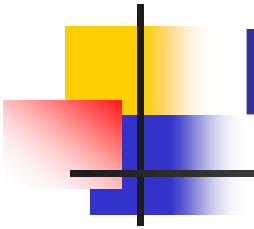
Agenda

- Week 2
  - Basic Concepts in Network Programming
  - Review of Some Helpful Points
- Week 4
  - Structures About IP Address and DNS
  - Sockets Interface
  - Major System Calls
  - Sample Programs



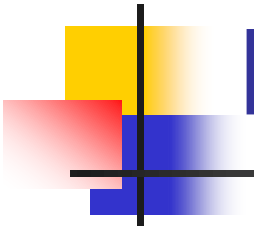
Basic Concepts in Network Programming

Basic Concepts in Network Programing



- *Introduction to Network Programming*
- Program Developing
- Basic Concepts
  - Process
  - File Descriptor
  - System Call
  - Signal

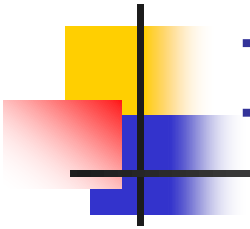
Introduction to Network Programming



- Network Programming encompasses various concepts, techniques and issues that are involved in writing programs which will communicate with other remote programs.
- Examples
  - Concepts – how to interact with the protocol stack
  - Techniques – what APIs (Application Programming Interface) to use …
  - Issues – how to handle reliability…

Introduction to NP - *classes*

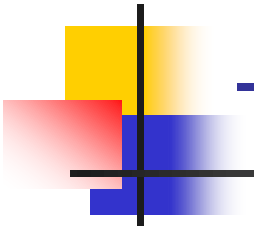
- Protocol Implementation
  - TCP/IP
  - IPX/SPX
  - ...
- Hiding the complexities
  - Sockets
  - RPC : Remote Procedure Call
- Programming language
  - C, C++, java, Perl, Virtual Basic, PHP, Python ...
- Applications for specific services
  - Mail server
  - Multimedia
  - Banking application
  - ...



Introduction to NP - *importance*

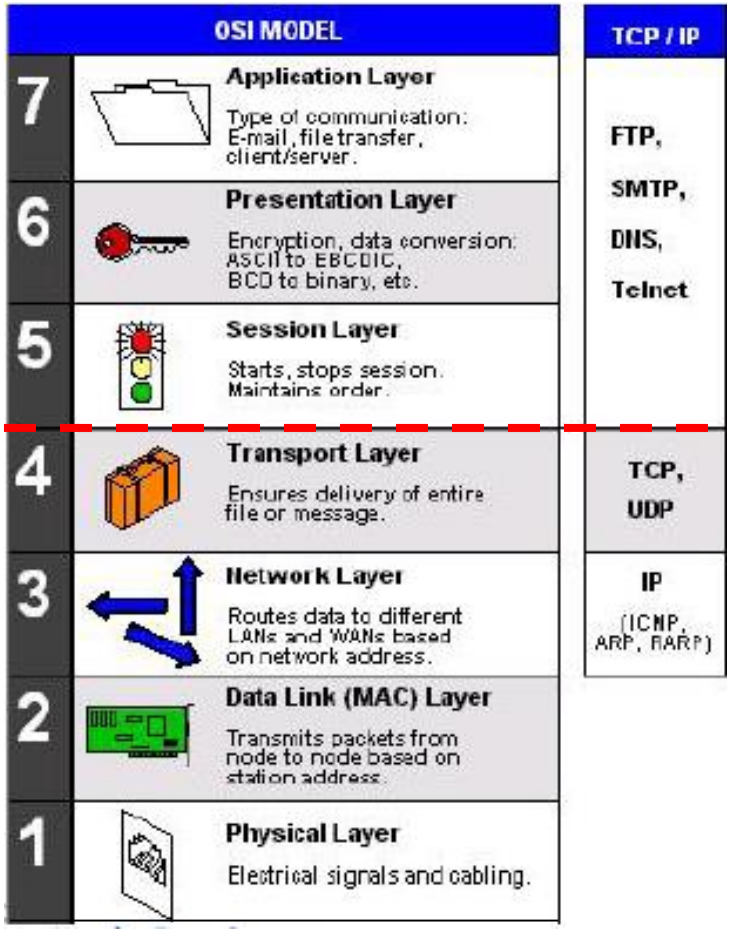
- Network Programming is a rather wide field
- The concepts and techniques learnt can be helpful in numerous application areas
  - Distributed applications
  - Intelligent/Remotely-managed Devices

Introduction to NP
– *environments in this course*



- TCP/IP nodes on Ethernet
- LINUX as the Operating System
- C language for most sample programs and assignments

Introduction to NP
– *environments in this course*

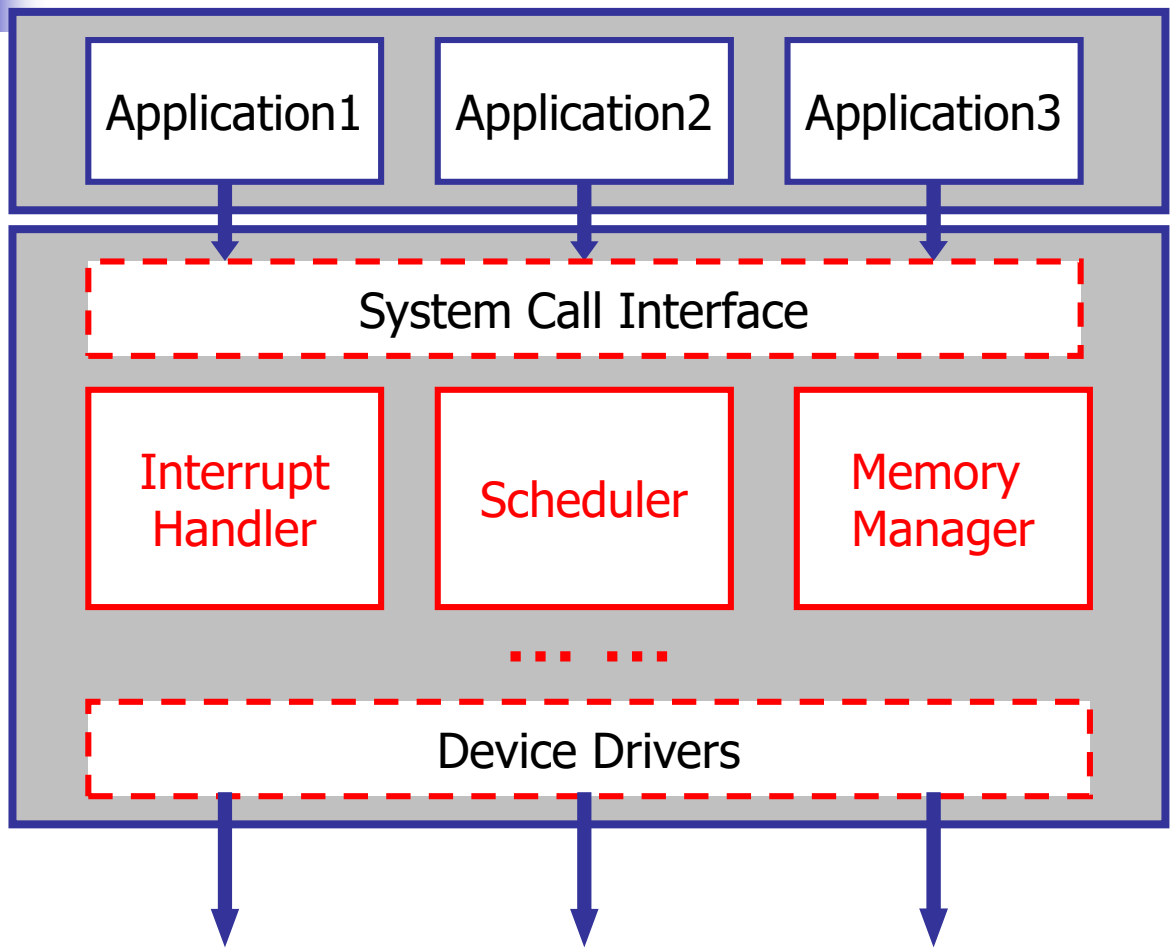


User and System Programs

Kernel Support

Hardware

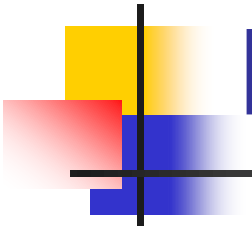
Introduction to NP
– *environments in this course*



User and System Programs

Kernel Support

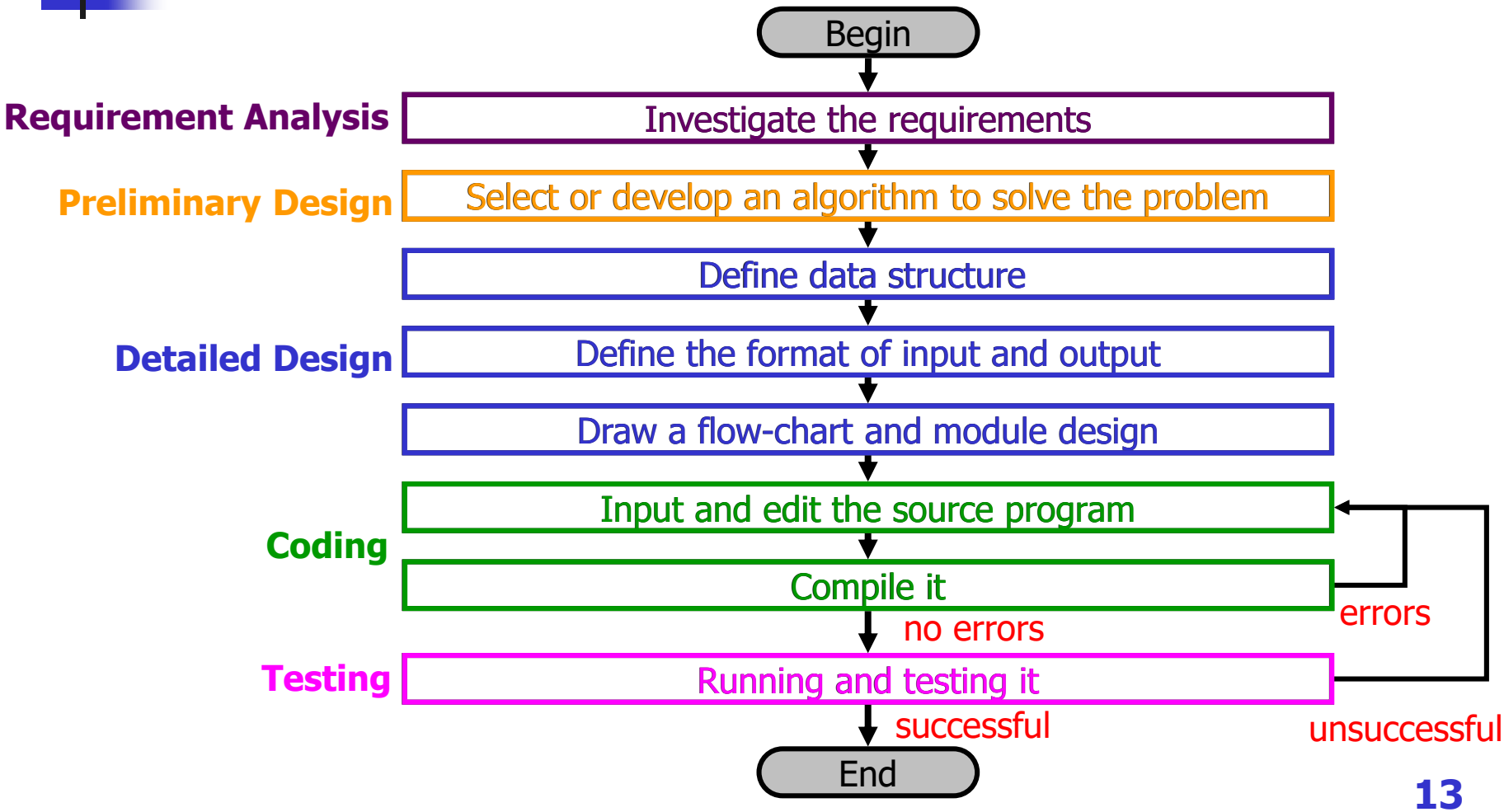
Hardware

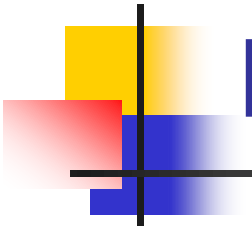


Basic Concepts in NP

- Introduction to Network Programming
- *Program Developing*
- Basic Concepts
  - Process
  - File Descriptor
  - System Call
  - Signal

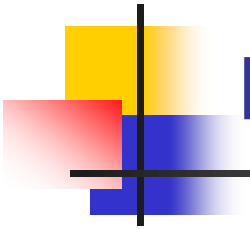
Program Developing - *Phases*





Program Developing - *skills*

- Programming style
  - ident, remarks, variable names
- Editor
  - vi, a very powerful full screen editor
  - pico, an utility with Linux
- Related Linux/Unix command
  - http://tech.sina.com.cn/2000-04-25/46/1.html
- Backup your program is important!

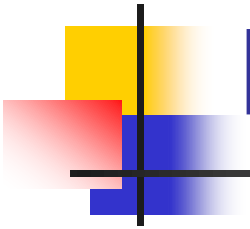


Program Developing – *C Compiler in Linux*

- cc
  - Example: % cc test1.c -o test
  - test1.c : program to be compiled
  - -o : specify the name for running program
- gcc
  - Example: % gcc test1.c –o test

Program Developing – *debugger in Linux*

- gdb [options] [executable-file [core file or process-id]]
- Example: % gdb test1
- gdb Command list
  - file : load the program for debugging
  - kill : stop the program for debugging
  - list : list the source code of the program for debugging
  - break : set a break point in the source program
  - run : run the program to be debugged
  - next : execute a single line of the program, but not go into it
  - step : execute a single line of the program, but go into it
  - quit : quit the gdb to shell
  - print : display the value of a variable
  - make : make a run-able program without quiting gdb
  - c : Continue running your program (e.g. at a breakpoint)
  - bt (backtrace) : display the program stack



Basic Concepts in NP

- Introduction to Network Programming
- Program Developing
- *Basic Concepts*
  - Process
  - System Call
  - File Descriptor
  - Signal

Basic Concepts - *definitions*

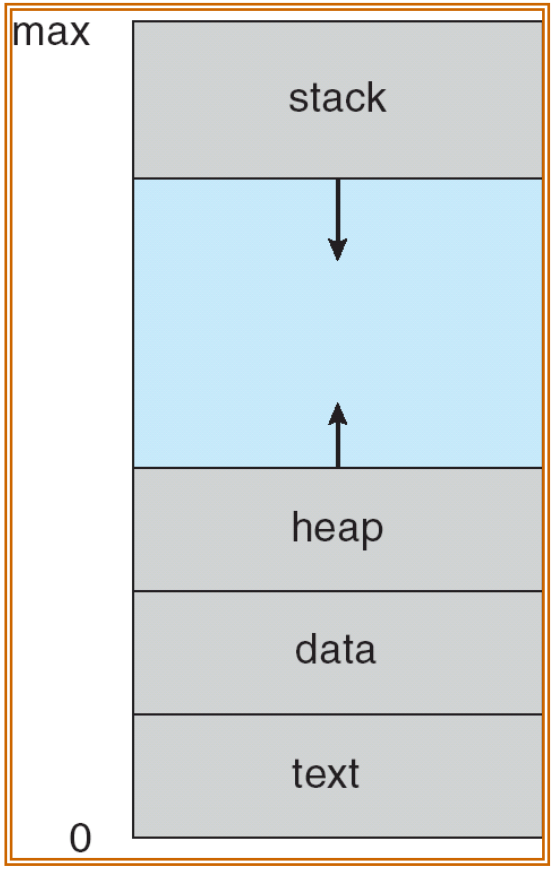
Process	▪ A process is an instance of a program that is being executed by the operating system.
System Call	▪ Linux/Unix kernel provides a limited number (typically between 60 and 200) of direct entry points through which an active process can obtain services from the Kernel.
File Descriptor	▪ A file descriptor is a small integer used to identify a file that has been opened for I/O operation.
Signal	▪ A signal is a notification to a process that an event has occurred.

Basic concepts - *process*

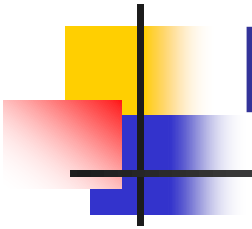
- One of the most basic abstractions in Unix (the other one is File)
- process ≠ program
  - Program：a file containing instructions to be executed, static
  - Process: an instance of a program in execution, live entity
  - One program can have multiple processes
  - One process can invoke multiple programs
- Alias: task, job

Basic concepts - *process*

- Process is the basic unit for resource allocation in operating system
- Process in memory
  - Text: program code
  - Data: global variables
  - Heap: dynamic allocated memory, malloc()
  - Stack: temporary data (local variable, function parameters, return addresses)



Basic concepts – *process*



- PID (Process ID): Every process has a unique PID. The PID is an integer, typically in the range 0 through 32,767.
- PPID (Parent PID): Every process has a parent process ID.
- Special process
  - PID = 1: init process
  - PID = 0: special kernel process (e.g., idle/swapper process)
  - PID = 2: special kernel process (e.g., page daemon process)

Basic concepts – *process*

- Linux command
  - ps –ef
  - To see every process on the system

```
[root@localhost ~]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Aug06 ?        00:00:00 init [5]
root         2     1  0 Aug06 ?        00:00:02 [migration/0]
root         3     1  0 Aug06 ?        00:00:00 [ksoftirqd/0]
```

Basic concepts – *process*

- Related system calls
  - fork(): to create a child process
  - getpid(): to obtain the PID of a process
  - getppid(): to obtain the PPID(Parent Process ID) of a process
  - exec(): often used after fork() to load another process
    - execl(), execv(), execle(), execve(), execlp(), execvp()
  - exit(): to terminate a process and release all the resources

Basic concepts – *process*

- Simple sample program of fork() – fork1.c

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

int main(void) {
        pid_t t;
        t=fork();
        printf("fork returned %d\n",t);
        exit(0);
}
```

```
$ gcc fork1.c -o fork1
$ ./fork1
fork returned 0
fork returned 22770
```

Basic concepts – *process*

- Complete sample program of fork() – fork2.c

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

int main (void) {

        pid_t t;

        printf("Original program, pid=%d\n", getpid());
        t = fork();
        if (t == 0) {
                printf("In child process, pid=%d, ppid=%d\n",
                        getpid(), getppid());
        } else {
                printf("In parent, pid=%d, fork returned=%d\n",
                        getpid(), t);
        }
}
```

```
Original program, pid=987
In child process, pid=988, ppid=987
In parent, pid=987, fork returned=988
```

Basic concepts – *process*

- Sample program of exec() – exec1.c

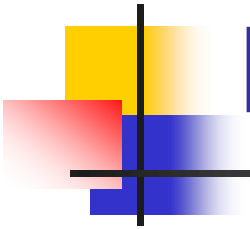
```
#include <unistd.h>
#include <stdio.h>

int main (void) {

        char *arg[] = { "/bin/ls", 0 };

        /* fork, and exec within child process */
        if (fork() == 0) {
                printf("In child process:\n");
                execv(arg[0], arg);
                printf("I will never be called\n");
        }
        printf("Execution continues in parent process\n");
}
```

```
[shiyan@localhost examples-for-ia]$ ./exec1
In child process:
Execution continues in parent process
[shiyan@localhost examples-for-ia]$ exec1  exec1.c
fork1  fork1.c  fork2  fork2.c
```



Basic concepts – *file descriptor*

- A file descriptor (an integer) is returned when a file is opened or created, and is used as an argument when later the file is read or written
- File descriptors are assigned by the kernel when the following system calls are successful
  - open
  - creat
  - dup
  - pipe
  - fcntl

Basic concepts – *file descriptor*

- There are two methods available under Unix for doing I/O (Input and Output)

method	Unix system calls for I/O	standard I/O library
concept	Working with file descriptors	Working with stream
header file	<unistd.h>	<stdio.h>
examples	open, read, write, lseek, …	printf, putc, getc, …

Basic concepts – *file descriptor*

- Related system calls
  - open
    - Be used to open or create a file
    - int open (char *pathname, int oflag, int mode);
  - close
    - Be used to close a file
    - int close (int filedes);
  - read
    - Be used to reading the data from an opened file
    - int read (int filedes, char *buff, unsigned int nbytes);
  - write
    - Be used to writing data into an opened file
    - int write (int filedes, char *buff, unsigned int nbytes);

Name of the file

O_RDONLY
O_WRONLY
O_RDWR
...
(<fcntl.h>)

File descriptor or -1

Only used when creating a file to indicate the access authority

0 or -1

Bytes of data that are read, 0 or -1

Bytes of data that are written or -1

Basic concepts – *file descriptor*

- Related system calls
  - lseek
    - Be used to locate in a file
    - long lseek (int filedes, long offset, int whence);
  - dup
    - Be used to duplicate a file descriptor
    - int dup (int filedes);
    - int dup2 (int filedes, int filedes2);
  - fcntl
    - Be used to change the properties of the file that is already open
    - int fcntl (int filedes, int cmd, int arg);

SEEK_SET
SEEK_CUR
SEEK_END

The new offset in the file or -1

The new file descriptor or -1

Depending on cmd or -1

Indicate the different functions of fcntl():
F_DUPFD,
F_GETFD/F_SETFD,
F_GETFL/F_SETFL,
F_GETOWN/F_SETOWN,
F_GETLK/F_SETLK/F_SETLKW

```
dup(filedes);  <==>  fcntl(filedes,F_DUPFD,0);
```

Basic concepts – *file descriptor*

- Sample program of lseek() – lseek1.c

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

char buf1[]="abcdefghij";
char buf2[]="ABCDEFGHIJ";
#define FILE_MODE 0644

int main(void)
{
        int fd;

        if ((fd=creat("file.hole",FILE_MODE))<0)
        {       printf("creat error\n");
                exit(1);}
```

Basic concepts – *file descriptor*

- Sample program of lseek() – lseek1.c

```
        if (write(fd,buf1,10)!=10)
        {       printf("buf1 write error\n");
                exit(1);}
         /*offset now =10 */
        if (lseek(fd,40,SEEK_SET)==-1)
        {       printf("lseek error\n");
                exit(1);}
         /*offset now =40 */
        if (write(fd,buf2,10)!=10)
        {       printf("buf2 write error\n");
                exit(1);}
         /*offset now =50 */
        exit(0);
}
```

Basic concepts – *file descriptor*

- Sample program of lseek() – lseek1.c

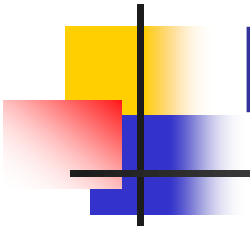
```
[shiyan@localhost examples-for-ia]$ ls -l file.hole
-rw-r--r--  1 shiyan shiyan 50 10ÔÂ  7 10:57 file.hole
[shiyan@localhost examples-for-ia]$ od -c file.hole
0000000   a   b   c   d   e   f   g   h   i   j  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0   A   B   C   D   E   F   G   H
0000060   I   J
0000062
```

- od command
  - be used to display the content of the file
  - -c: display in character format
- When using lseek, the offset of the file can be larger than the length of the file. So, the next writing operation will extend the file and a hole will be made inside the file.

Basic concepts – *file descriptor*

- Sample program of read() and write() – readwrite1.c

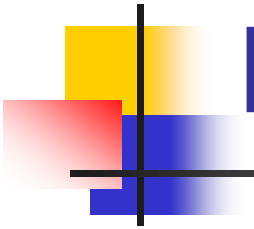
```
#include <fcntl.h>
#include <unistd.h>
int main(void)
{
        char quit='.';
        char buf[10];
        int fd;
        if((fd = open("out.out",O_RDWR | O_CREAT,0))==-1)
                printf("Error in opening\n");
        while(buf[0]!=quit)
        {
                read(0,buf,1);
                write(fd,buf,1);
                write(1,buf,1);
        }
        close(fd);
}
```



Basic concepts – *system call*

- System call is the only method for the user space to access the kernel

Basic concepts – *signal*



- Signals are some time called “Software interrupts”
- Signals can be sent from one process to another or from kernel to a process
- Header file: <signal.h>
- The names of the signals begin with SIG
  - SIGALRM: alarm clock timeout
  - SIGINT: Interrupt character (Ctrl-C) is typed

Basic concepts – *signal*

Five conditions that generate signals

1. **Kill system call**
   - the system call ***kill*** allows a process to send a signal to another process or to itself
2. **Kill command**
   - the command ***kill*** is also used to send a signal
   - often used to terminate a background process out of control
3. **Certain terminal characters**
   - e.g., the interrupt character (typically control-C or Delete) terminates a process that is running - it generates a SIGINT signal
4. **Certain hardware conditions**
   - the hardware detects these conditions and then notifies the kernel. E.g., invalid storage access - SIGEGV
5. **Certain software conditions**
   - the kernel notices these conditions and generates the signal. E.g., SIGALRM

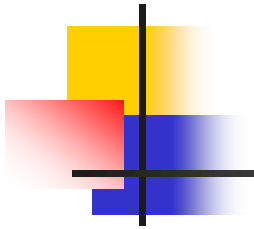
Basic concepts – *signal*

What can a process do with a signal?

1. **Catch the signal**
   - A process can provide a function that is called whenever a specific type of signal occurs. This function is called **handler**.

2. **Ignore the signal**
   - A process can choose to ignore a signal
   - Two signals that can not be ignored: SIGKILL, SIGSTOP

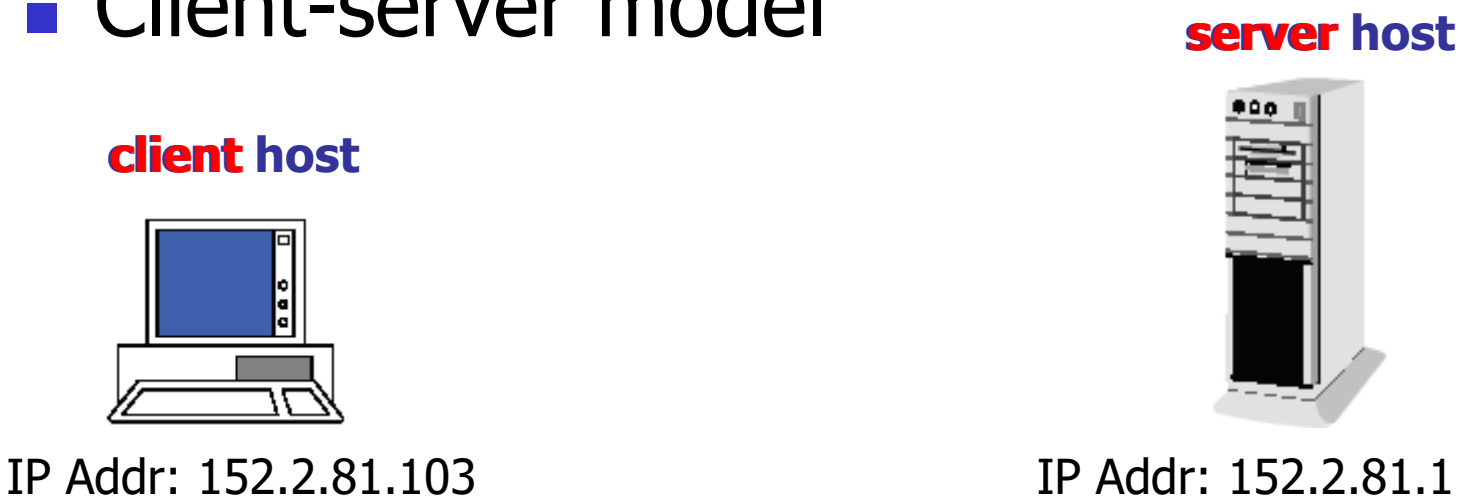
3. **Execute the default action**
   - A process can allow the default to happen
   - default actions of most signals are to terminate the process



Review of Some Helpful Points

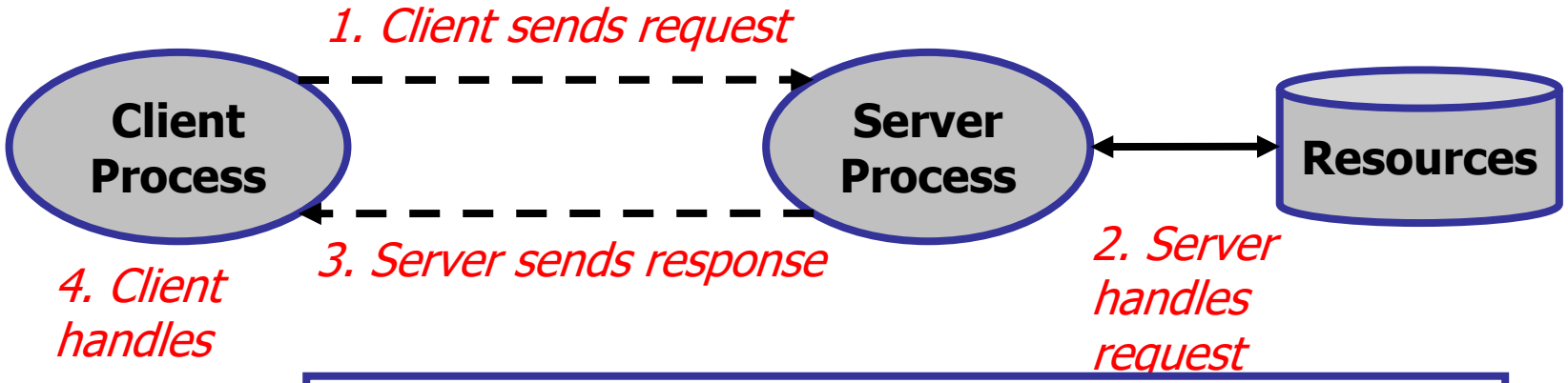
Reviews of Some Helpful Points

- Client-server model



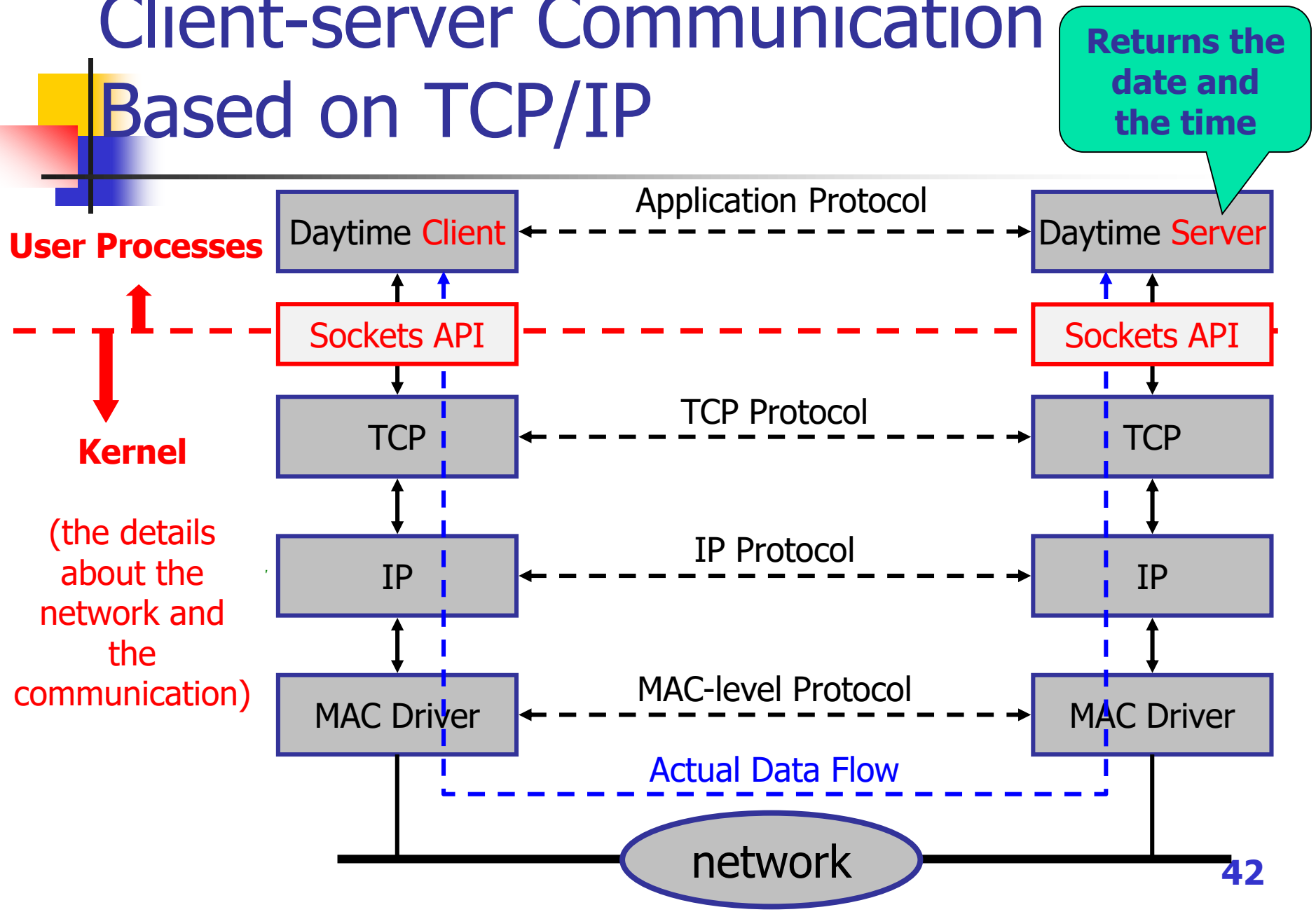
A Client-server Transaction

- Most of network applications are based on the client-server model:
  - A ***server*** process and one or more ***client*** processes
  - Server manages some ***resources***.
  - Server provides ***service*** by manipulating resources for clients.



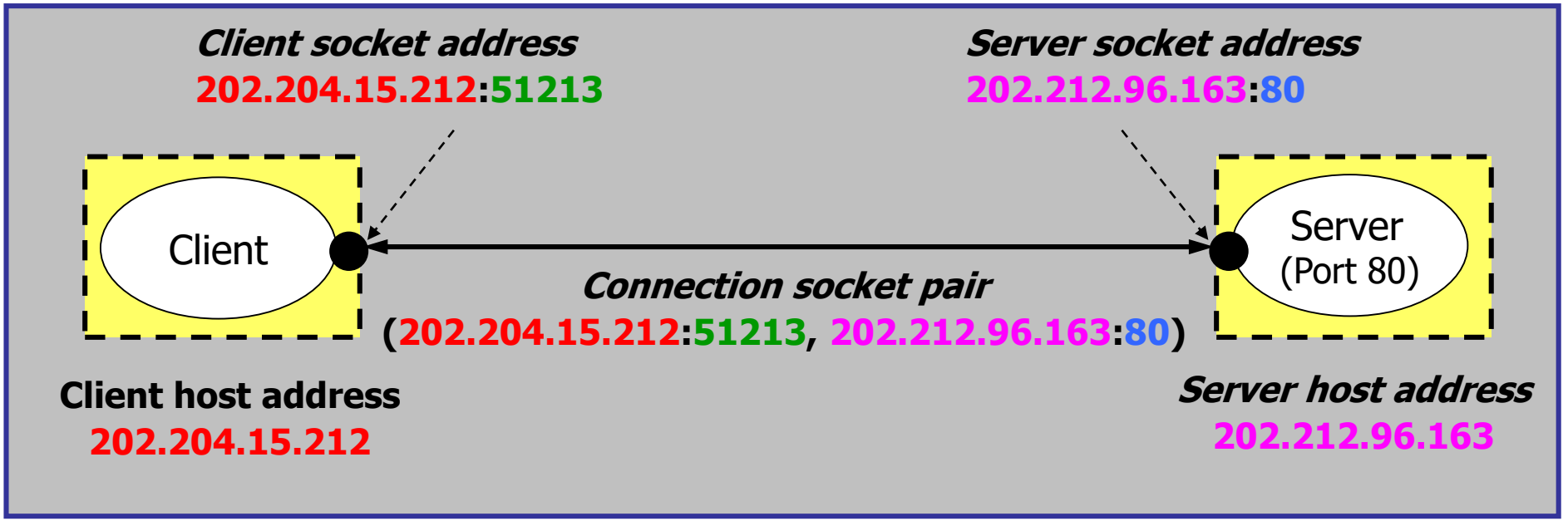
Note: *clients and servers are processes running on hosts (can be the same or different hosts).*

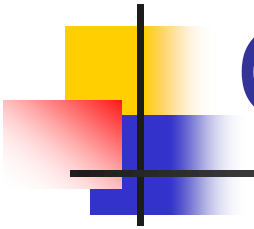
Client-server Communication Based on TCP/IP



Connections

- Clients and servers communicate by sending streams of bytes over *connections*.
- Connections are end-to-end, full-duplex (2- way communication), and reliable.

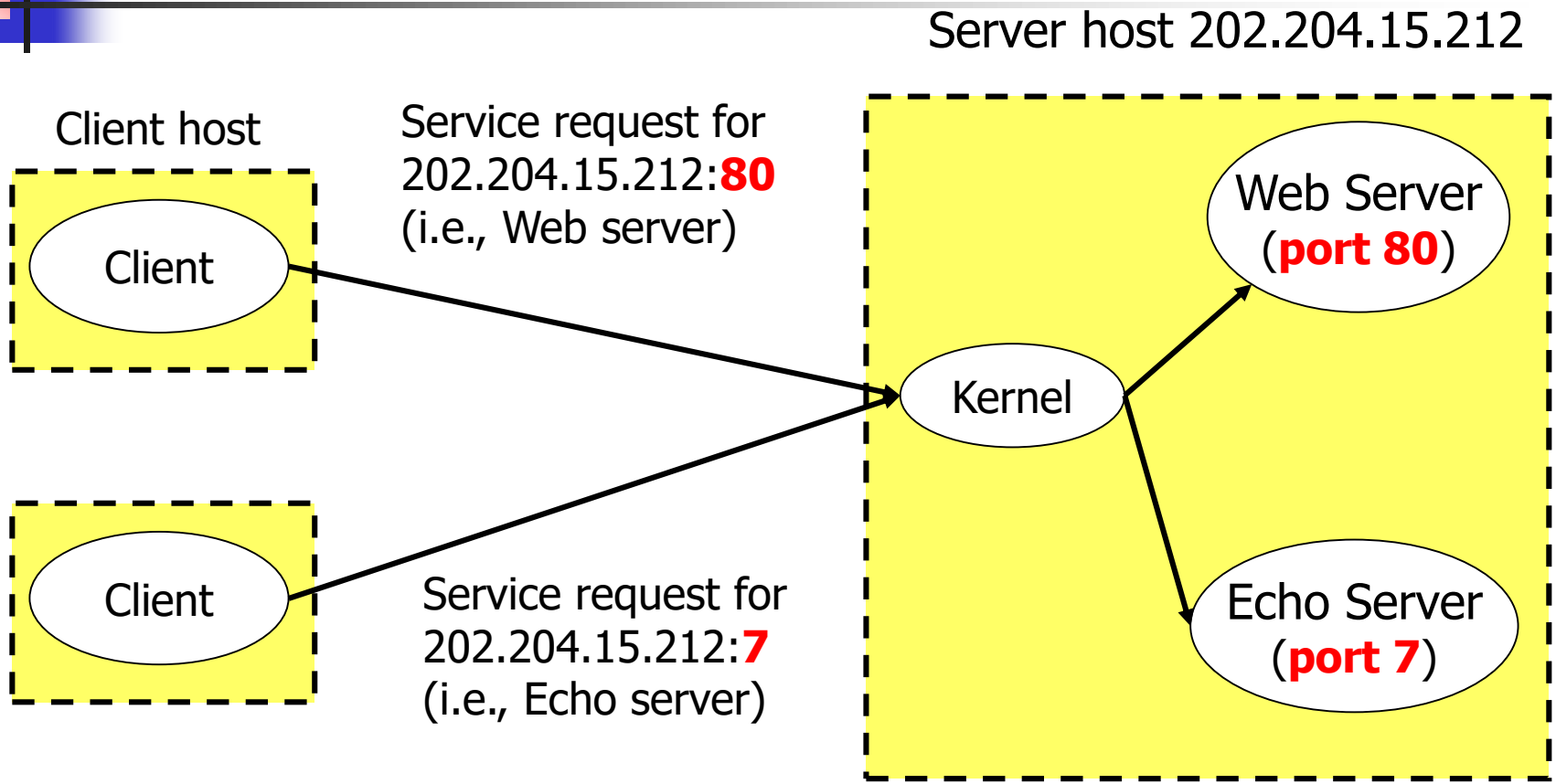


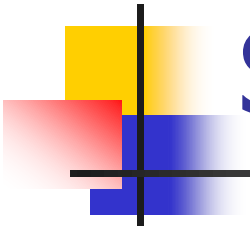


Clients

- Examples of client programs
  - Web browsers, ftp, telnet, ssh
- How does a client find the server?
  - The IP address in the server socket address identifies the host *(more precisely, an adapter on the host)*
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
  - Examples of well-known ports
    - Port 7: Echo server
    - Port 23: Telnet server
    - Port 25: Mail server
    - Port 80: Web server

Using ports to identify services



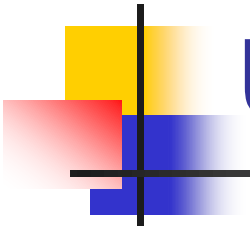


Servers

- Servers are long-running processes (daemons).
  - Typically created at boot-time by the **init process** (pid=1)
  - Run continuously until the machine is turned off
- Each server waits for requests to arrive on a well-known port associated with a particular service.
  - See ***/etc/services*** for a comprehensive list of the services available on a Linux machine
- A machine that runs a server process is also often referred to as a "server"

Server examples

Name	Port	Services	Resources
Web server	80	Retrieves files and runs CGI programs on behalf of the client	files/compute cycles (CGI programs)
FTP server	20, 21	stores and retrieve files	files
TELNET server	23	proxies a terminal on the server machine	terminal
Mail server	25	stores mail messages in spool file	email “spool” file



Useful Unix Commands

- netstat
- ifconfig
- ping

Useful Unix Commands - netstat

- Functions: prints information about the Linux networking subsystem, e.g., network connections, routing tables, interface statistics etc.
- netstat
  - Displays a list of open sockets.
- netstat -i
  - Display the information about the network interfaces
- netstat -ni
  - Display the information about the network interfaces using numeric addresses
- netstat -r
  - Display the kernel routing tables
- netstat -nr
  - Display the kernel routing tables using numeric addresses

Useful Unix Commands - netstat

- netstat

```
[root@localhost include]# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 192.168.1.253:telnet       192.168.1.27:3256           ESTABLISHED
tcp        0      0 ::ffff:192.168.1.253:ssh   ::ffff:192.168.1.27:2888    ESTABLISHED
tcp        0      0 ::ffff:192.168.1.253:ssh   ::ffff:192.168.1.27:3047    ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node Path
unix  10     [ ]         DGRAM                    5724   /dev/log
unix  2      [ ]         DGRAM                    6859   @/var/run/hal/hotplug_socket
unix  2      [ ]         DGRAM                    3351   @udevd
unix  2      [ ]         DGRAM                    927082
unix  2      [ ]         DGRAM                    926850
unix  3      [ ]         STREAM     CONNECTED     924266
unix  3      [ ]         STREAM     CONNECTED     924265
unix  3      [ ]         STREAM     CONNECTED     916866 /tmp/.X11-unix/X16
```

Useful Unix Commands - netstat

- netstat –ni

```
[root@localhost /]# netstat -ni
Kernel Interface table
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500   0 49922499      0      0      0 20779220      0      0      0 BMRU
lo        16436   0     1934      0      0      0     1934      0      0      0 LRU
```

- Ethernet interface is called eth0 or le0 depending on the machine
- Loop back interface is called lo and the common used IP address is 127.0.0.1

Useful Unix Commands - netstat

- netstat -nr

```
[root@localhost /]# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0          0 eth0
```

Useful Unix Commands - ifconfig

- Functions: configure the network interfaces, and usually be used to print the configuration of the network interfaces

```
[root@localhost /]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:13:72:4F:9D:3A
          inet addr:192.168.1.253  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::213:72ff:fe4f:9d3a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49923781 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20779648 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:647355456 (617.3 MiB)  TX bytes:2713364 (2.5 MiB)
          Base address:0xecc0 Memory:fe6e0000-fe700000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1934 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1934 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:266858 (260.6 KiB)  TX bytes:266858 (260.6 KiB)
```

Useful Unix Commands - ping

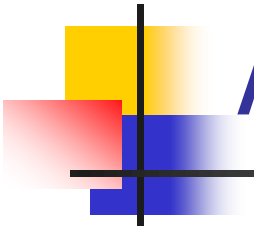
- Functions: Sends a packet to the host specified by destination and prints out the roundtrip time (Using ICMP messages)

```
[root@localhost etc]# ping 192.168.1.27
PING 192.168.1.27 (192.168.1.27) 56(84) bytes of data.
64 bytes from 192.168.1.27: icmp_seq=0 ttl=128 time=0.261 ms
64 bytes from 192.168.1.27: icmp_seq=1 ttl=128 time=0.219 ms
64 bytes from 192.168.1.27: icmp_seq=2 ttl=128 time=0.181 ms

--- 192.168.1.27 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.181/0.220/0.261/0.034 ms, pipe 2
```

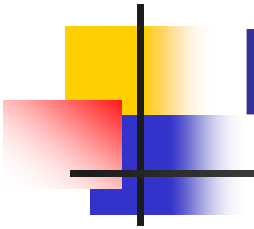
```
[root@localhost etc]# ping www.baidu.com
PING www.a.shifen.com (202.108.22.5) 56(84) bytes of data.
64 bytes from xd-22-5-a8.bta.net.cn (202.108.22.5): icmp_seq=0 ttl=57 time=363 ms
64 bytes from xd-22-5-a8.bta.net.cn (202.108.22.5): icmp_seq=1 ttl=57 time=177 ms
64 bytes from xd-22-5-a8.bta.net.cn (202.108.22.5): icmp_seq=2 ttl=57 time=172 ms

--- www.a.shifen.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 172.446/237.748/363.698/89.081 ms, pipe 2
```



Abbreviations

API	Application Programming Interface
IP	Internet Protocol
IPX	Internetwork Packet Exchange
Perl	Practical Extraction and Report Language
PID	Process Identifier
PPID	Parent Process Identifier
RPC	Remote Process Call
SPX	Sequenced Packet Exchange
TCP	Transport Control Protocol
UDP	User Datagram Protocol



Reference books

- W. Richard Stevens, ***Advanced Programming in the UNIX Environments***. 中译本: 尤晋元译, 机械工业出版社.
- W. Richard Stevens, ***UNIX Network Programming, Volume 1***. 中译本: 施振川等译, 清华大学出版社.
- Robert Love, ***Linux Kernel Development***. 中译本: 陈莉君 康华 张波 译, 机械工业出版社.