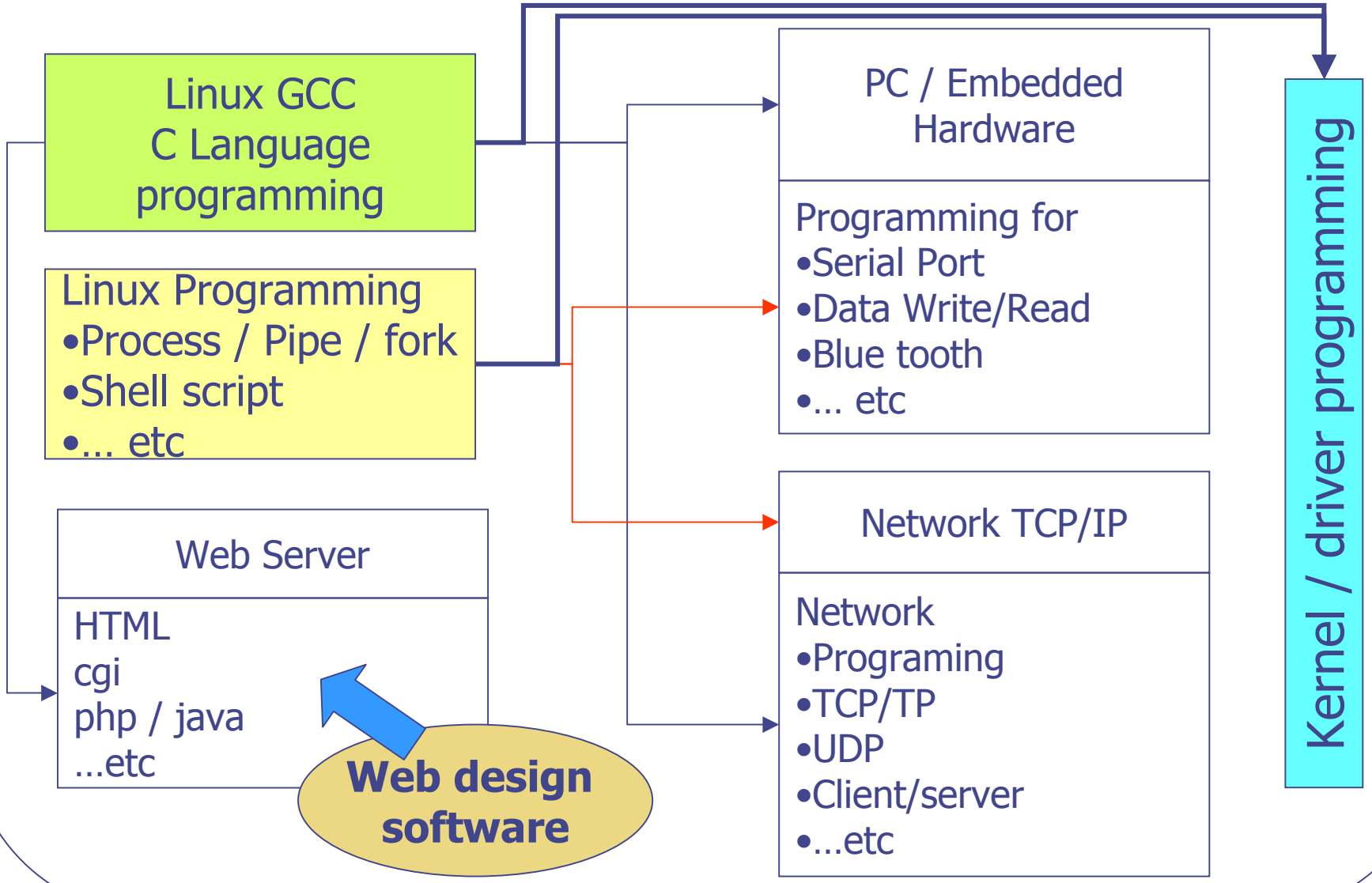
Part 3

Linux Programming in Brief

By : H K Sim
(Contact Email : simhkeng@yahoo.com)
March 2007

Areas Involved



Email : simhkeng@yahoo.com

Makefile

Email : simhkeng@yahoo.com

Compilation Process

Pre-processor

Pre-processor runs before the complier and add or manage the lines of code that will be sent for actual compilation.

Line of code that with the # (hash mark) is belongs to the pre-processor , not compiler. The common instructions (called directives) are :

#include

#define

#ifdef … #endif this Supports conditional compilation.

Compilation

The compiler take in the xxx.c file and turn it into a object file with a .o extension. When compiling with a gcc, this file is usually deleted once the compilation is done successfully. Option extension can be use to compile into object file e.g. xxx.o

Email : simhkeng@yahoo.com

Compilation Process

Linker

The linker combines the .o files and the specified libraries into a single executable file. In linux ,by default, the library file /usr/lib/libc.so is also included.

e.g If a program includes #inlcude<math.h>, the maths library (/usr/lib/libm.so) will be linked to provide the final executable program.

Email : simhkeng@yahoo.com

Make Utility

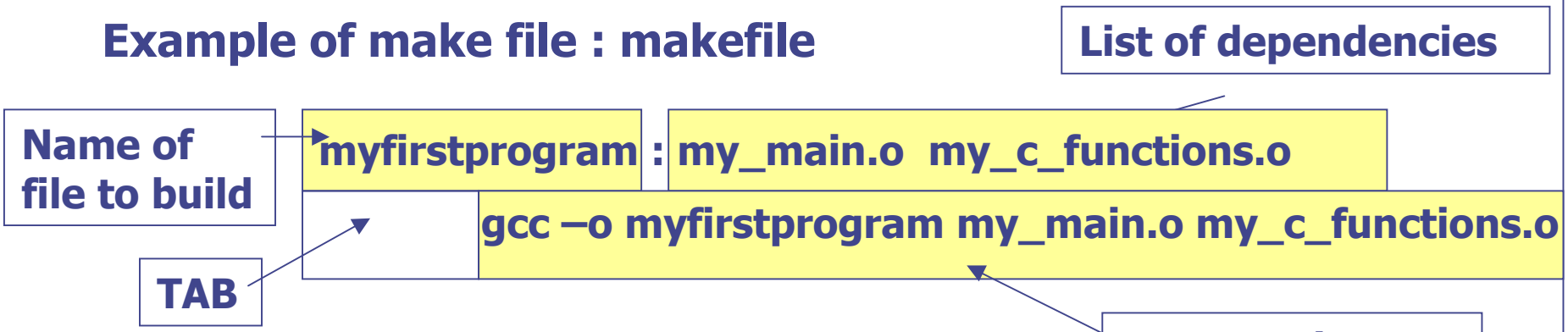
- Use in large development project involves many C code files and header files.
- The make file content must be created in the file name called “makefile” or “Makefile”.
- To run the make utilities, just type (@ command prompt) :

  …#] make.
- Make utility will look for the “makefile” or “Makefile” and execute it
- Makefile is very sensitive to the format sequence , carriage return and Tab. Tab cannot be replaced by Space.

Email : simhkeng@yahoo.com

Make Utility

Example of make file : makefile




```
my_main : my_main.c
        gcc –c my_main.c
my_c_function : myfunction.c
        gcc –c my_function.c
.PHONY : clean
 clean : rm *.o
```

For command that longer than one line, the next line can be continued using a "\"

Email : simhkeng@yahoo.com

Multi-file compile and link Example

main.c

```
#include<stdio.h>
#include "myheader.h"

int main(void)
{
int i = 10, k = 50;
int m, result;

m = add (i,k);
result = multiply( m, gInt);

printf("Final Result =
%d\n",result);

return 0;
}
```

myheader.h

```
int add ( int x, int y);
int multiply ( int x, int y);
int gInt = 100;  // declare global
integer
```

func1.c

```
#include<stdio.h>
int add( int x, int y)
{
printf("Print from func1.c->(x+y) =
%d\n",(x+y));
return (x+y);
}
```

func2.c

```
#include<stdio.h>
int multiply(int x, int y)
{
printf("Print from func2.c->(x*y) =
%d\n",(x*y));
return (x*y);
}
```

Compile , link and create a executable file called multi_files

Email : simhkeng@yahoo.com

Multi-file compile and link Example

Compile and "link" by hand

```
gcc –c main.c –o main.o
gcc –c func1.c –o func1.o
gcc –c func2.c –o func2.o
gcc main.o func1.o func2.o –o multi_files
```

Using makefile

```
multi_files: main.o func1.o func2.o
                    gcc -o multi_files   main.o    func1.o    func2.o
main.o :main.c
          gcc -c main.c
func1.o :func1.c
          gcc -c func1.c
func2.o :func2.c
          gcc -c func2.c

.PHONY:          clean
clean :
          rm -f *.o
```


Email : simhkeng@yahoo.com

Compilation extension

Common gcc compiler's interpretation of extension

-o filename - output to a file call "filename" If not specified, default is a.out

-c - compile without linking

-Idirname - specific directory that gcc will search for include file

-Ldirname - specific directory that gcc will search for library file

-static - link the static library

-lmylib - link the mylib library

-g - include standard debugging information

-O - optimize the compiled code

-W - suppress all warning message

-Wall - display all the warning message that gcc can provide

-v - show the commands use in each step

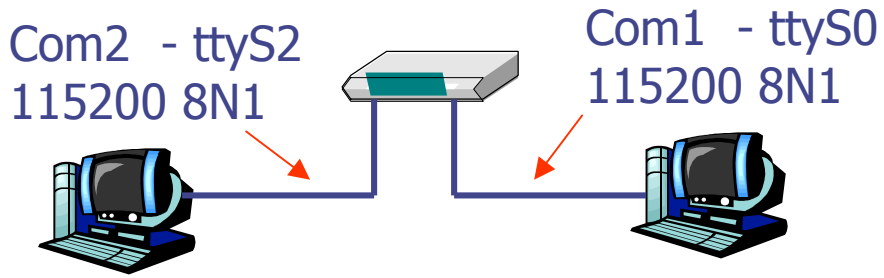
Email : simhkeng@yahoo.com

Serial Port Programming

Email : simhkeng@yahoo.com

Serial Server Demo Connection

- Connect two PC to the A-LimEmb Serial port server
- A-LinEmb is running the serial port server program
- What ever type on one PC will appear to another PC



Email : simhkeng@yahoo.com

A-LinEmb COMx

- By default, A-LinEmb COM1 is running the monitor program which is use to monitor or command the A-LinEmb Server.

  the default setting for COM1 is 115200 8N1

- Open 2nd COM2 port, ttyS2. for communication

```
//portname="/dev/ttyS2";  // for Axis 2nd serial port
  portname="/dev/ttyS0";  // for PC
  pf = open(portname, O_RDWR);
  if (pf < 0) {  printf("\n *** Serial Port Open Error ****\n");   }
```

- modify the port configuration

```
tcgetattr(pf, &pts);
 pots = pts;                    // note :  static struct termios pots;

/* # of data bits */
//pts.c_cflag |= CS5;  //CS8, CS7, CS6 , CS5
  pts.c_cflag |= CS8;  //CS8, CS7, CS6 , CS5
```


Email : simhkeng@yahoo.com

A-LinEmb COMx

❖ **Set number of stop bit**

```
pts.c_cflag |= 0;                // 1 stop bit
// pts.c_cflag |= CSTOPB;  // 2 stop bits
```

❖ **Set parity**

```
pts.c_cflag |= 0;     // No parity

 /* odd parity setting */
    // pts.c_cflag |= PARENB; // parity enable
    // pts.c_cflag |= PARODD;

/* even parity setting */
    // pts.c_cflag |= PARENB; // parity enable
```

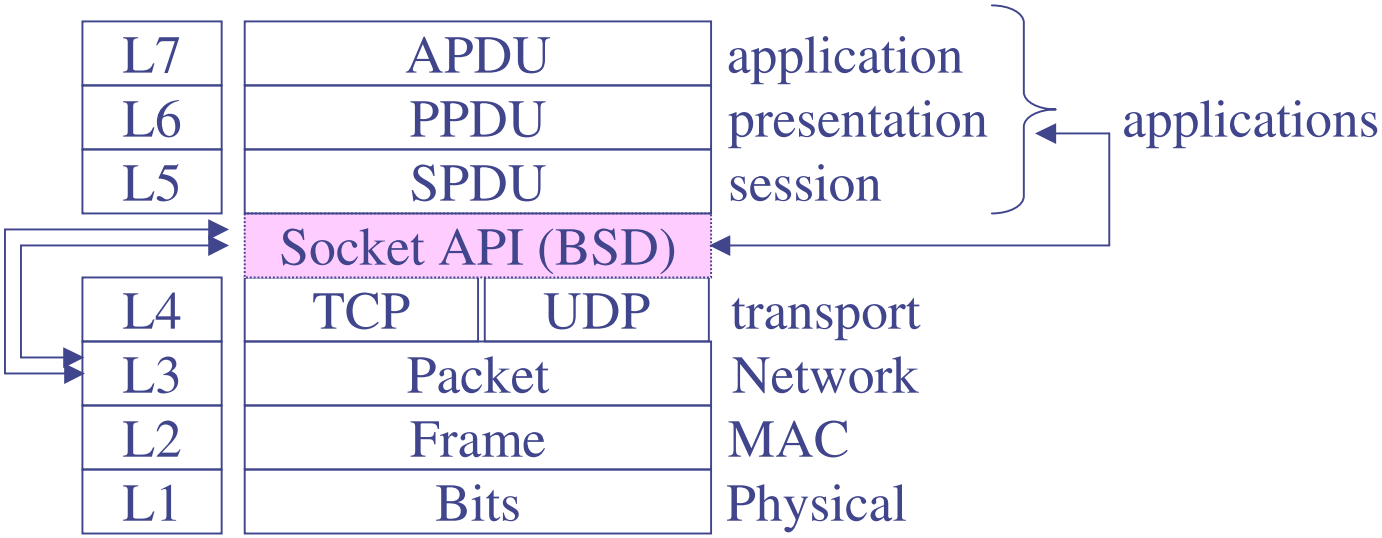

Email : simhkeng@yahoo.com

Network Socket Programming

Email : simhkeng@yahoo.com

Network Socket

- A socket is a communication connection point that one can named and addressed in a network
- Socket layer sits between transport layer and application layers.



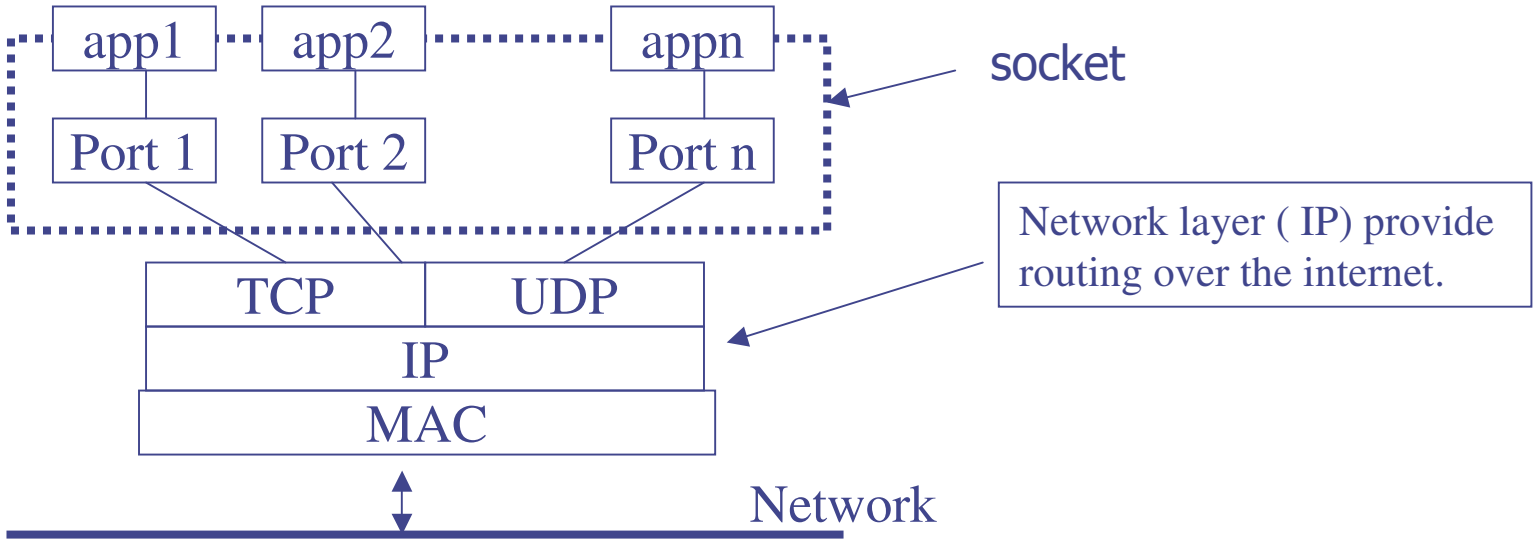
- Socket Application Program Interface (API) are the network standard for TCP/IP
- Socket programming shows how to use socket API to establish communication link between computers

Email : simhkeng@yahoo.com

Network Socket

How Socket Work?

- Socket is represented by an integer called socket descriptor
- Socket are commonly used for client / server interaction
- Server program running on one PC (server) and client program running on another PC (client).Client connects to server, exchange information
- Socket exists as long as the process maintains an open link to the socket
- When process completed, socket is disconnected



Email : simhkeng@yahoo.com

Network Socket

3 elements for setting up network socket

- **Host**
  - Host are identified by IP address.
  - e.g IPv4 address 192.168.1.20 – a 32 bits address
- **Protocol**
  - Specifies the detail of communication over socket
  - TCP – Transmission Control Protocol
  - UDP – User Datagram Protocol
- **Port**
  - End point for a given process (interface)
  - Port number < 1024 are reserved for well-known services
  - Port number > 1024 are reserved for application

Note : Socket interface has a set of predefined symbolic constants and data structure declaration. (/usr/include/bit/)

AF_xxx - Address family constant in <socket.h>

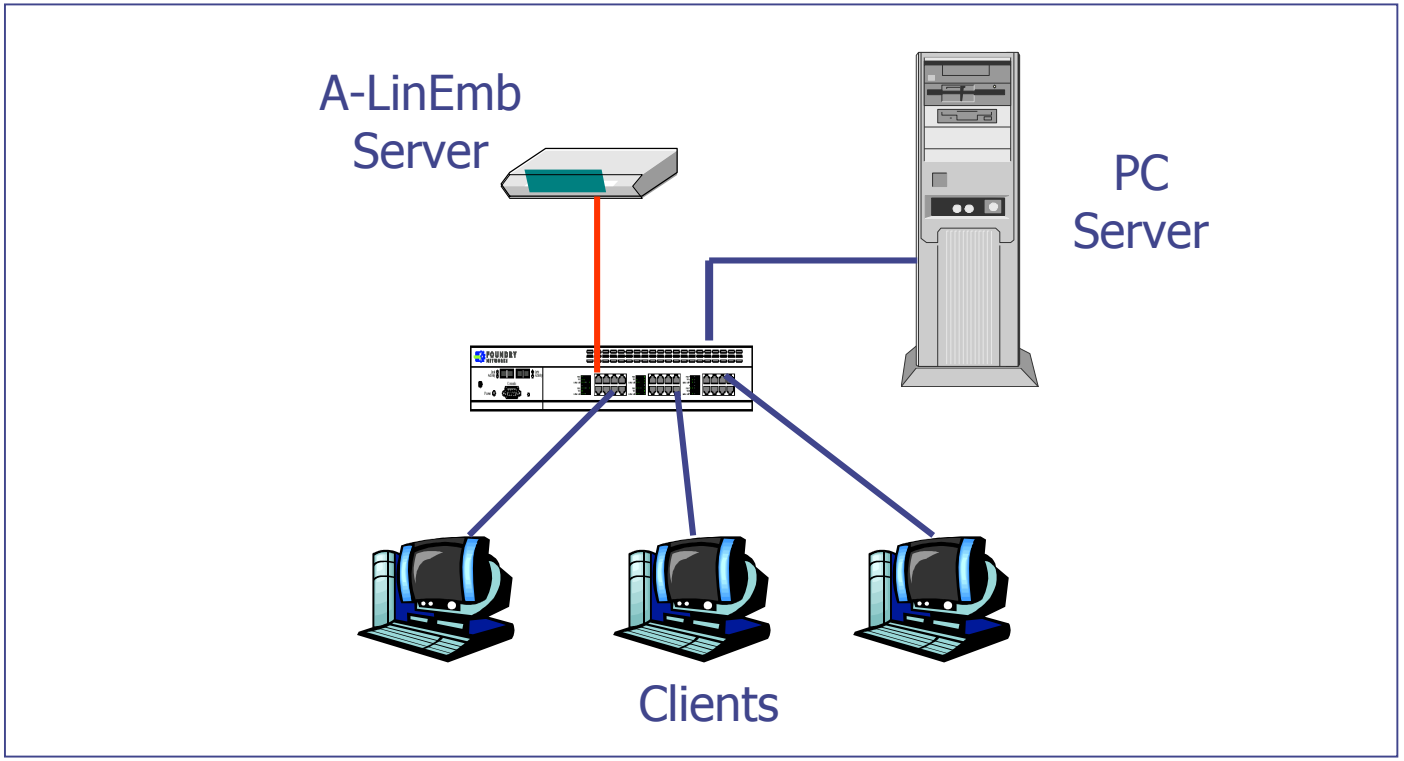
PF_xxx - protocol type constant in <types.h>

Email : simhkeng@yahoo.com

Client/Server Model

Role of client / server model

- Server – response to request
- Client – make request.
- Socket API provides functions that are specific to client and server
- Two client / server models, connection-oriented and connectionless



Email : simhkeng@yahoo.com

Client/Server Model

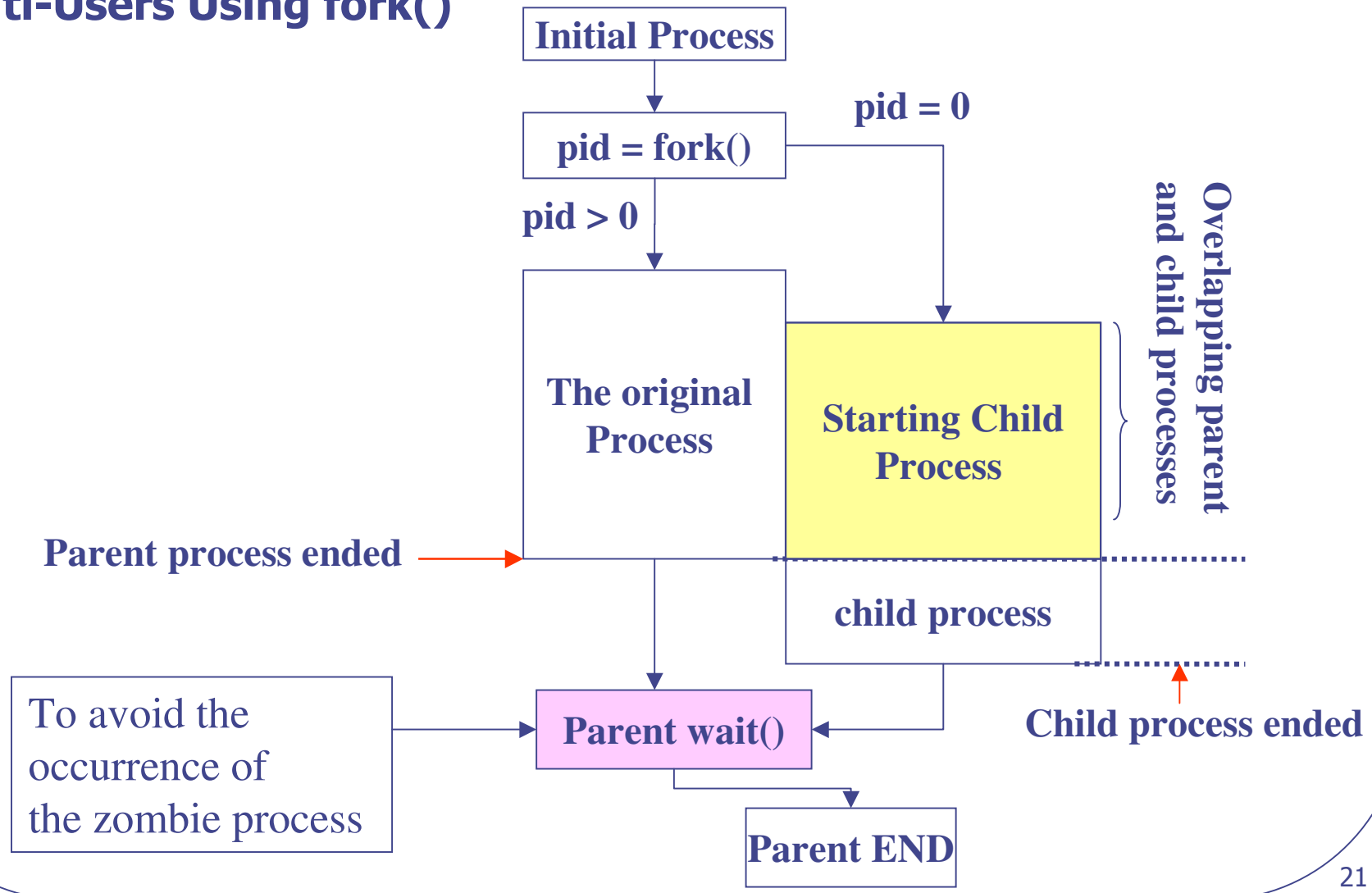
Connection-oriented

- After a server opened a network socket and bind the socket with the IP address and port number, the server waits for clients to request a service (client connect).
- After accepted the client's connection request, the client-to-server data exchange takes place until the task is completed. The client then disconnect from the server by closing the socket.
- Server uses one socket for listening to client's request and uses another socket for data exchange with the connected client.
- The dual socket model can be expended into multiple socket for data exchange with multiple clients ,using fork() function, to generate a child for each client's data exchange. The listen socket remain a one.

Email : simhkeng@yahoo.com

Client/Server Model

Multi-Users Using fork()



Email : simhkeng@yahoo.com

Client/Server Model

Connectionless

- Connectionless communication implies that no connection is established over which a dialog or data transfer takes place . The server program designates a name that identifies where to receive end send data.
- Server obtains the client's IP address from the packet it received

Email : simhkeng@yahoo.com

Client/Server Model

Steps involve in connection-oriented socket application

Server

- Create socket on server ,called passive socket, using socket()
- Use bind() to bind the server to an address and port
- call the listen() function to check if any client request for connection

If client is requesting for connection …

- use accept() function to accept client's connection request – single client
- use accept() function to accept client's connection request and create a new server side client socket using fork() for data exchange – multi-client model
- Data can send (using send()) or receive (using recv()) asynchronously between client and sever.
- When completed, the client will use then close() API to close the connection. The socket for this particular client is thus close too.

Email : simhkeng@yahoo.com

Client/Server Model

Steps involve in connection-oriented socket application

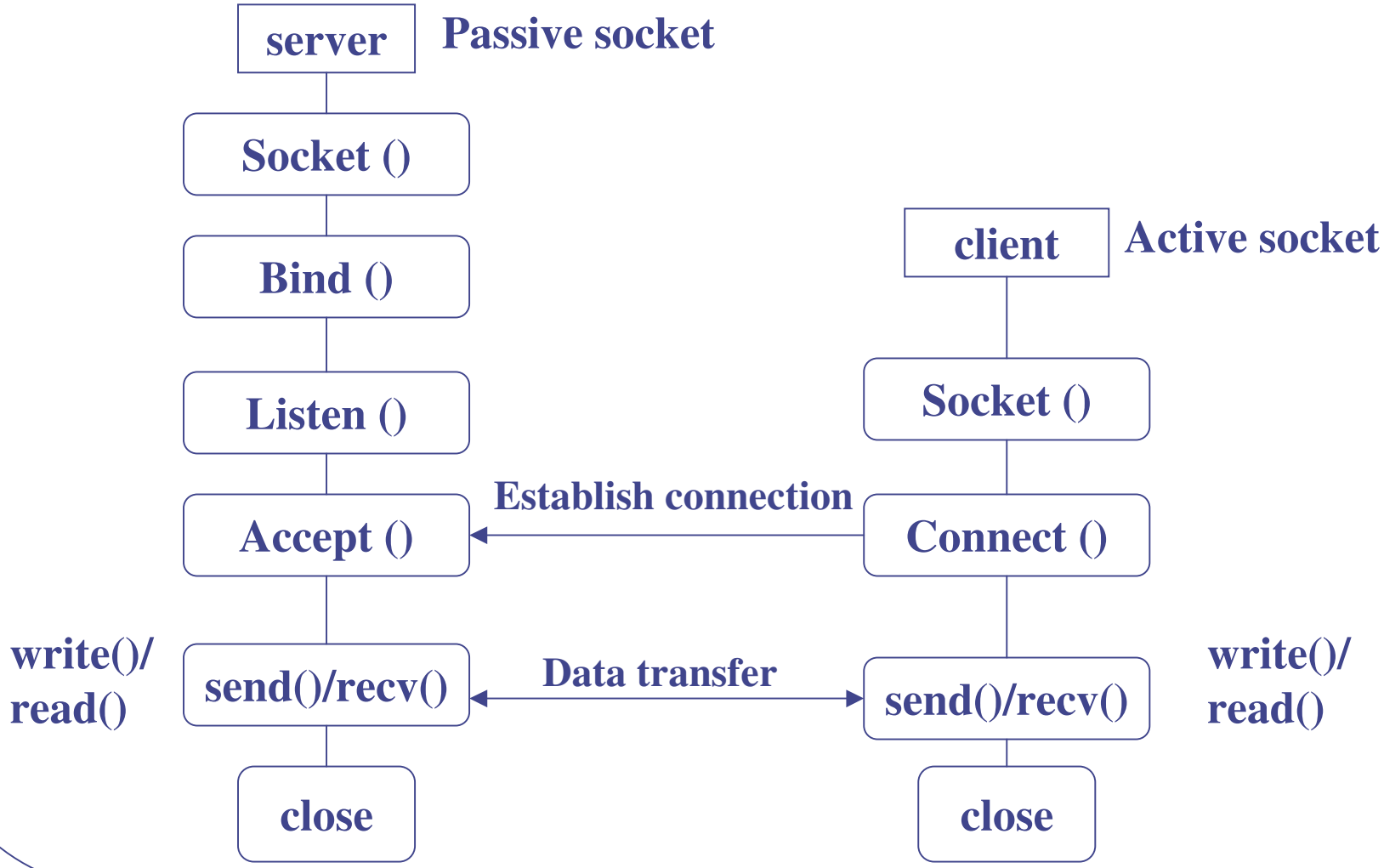
Client

- Create socket , call active socket, using socket()
- Using connect() to connect to a server (specify the IP address and Port)
- Perform data I/O using send()/recv() asynchronously
- close() socket

Email : simhkeng@yahoo.com

Client/Server Model

Steps involve in connection-oriented socket application



Email : simhkeng@yahoo.com

Connection-oriented communication send() / recv()

Server

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int main ( int argc , char *argv[])
{       • • •
serverSocket = socket(AF_INET, SOCK_STREAM,0);
        • • •
myServer.sin_family=AF_INET;
myServer.sin_port=htons(appPort);
//myServer.sin_addr.s_addr=inet_addr(myServerIP); // or
myServer.sin_addr.s_addr=htonl(INADDR_ANY);
        • • •
returnStatus = bind(serverSocket, (struct sockaddr *)&myServer,
                    sizeof(myServer));
        • • •
returnStatus=listen(serverSocket, listenBacklog);
        • • •
```


Email : simhkeng@yahoo.com

Connection-oriented communication send() / recv()

Server

```
while(1)
{
 struct sockaddr_in clientName={0};
 int clientSocket=0;
 int clientNameLength = sizeof(clientName);
    •••   //wait here
clientSocket = accept(serverSocket, (struct sockaddr *)&clientName,
                &clientNameLength );
    •••
 byteSent=write(clientSocket, welcomeMsg,strlen(welcomeMsg));
    •••
 // get messgage from client
 byteReceived = read(clientSocket, buffer, sizeof(buffer));
    •••
 close(clientSocket);
 }
```


Email : simhkeng@yahoo.com

Connection-oriented communication send() / recv()

Client

```
clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    • • •
/* retrieve the port number for connecting */
appPort = atoi(argv[2]);
serverToConnect.sin_family = AF_INET;
serverToConnect.sin_addr.s_addr=inet_addr(argv[1]);
serverToConnect.sin_port = htons(appPort);
    • • •
/*  connect to the address and port with our socket  */
returnStatus = connect(clientSocket, (struct sockaddr *)&serverToConnect,
                sizeof(serverToConnect));
     • • •

byteReceived = read(clientSocket, buffer, sizeof(buffer));
byteSent=write(clientSocket, ackMsg,strlen(ackMsg));
      • • •
```


Email : simhkeng@yahoo.com

Write/Read Data to file

Email : simhkeng@yahoo.com

Write Binary Format to File

```
struct devicePktData
 {
   char name[20];
   char IPAddr[20];
   char date[20];
   char time[20];
   char data1[20];
   char data2[20];
   char data3[20];
 };
```

```
struct devicePktData Device1,Device2;
```

```
// simulated data
strcpy(Device1.name,"Device1");
strcpy(Device1.IPAddr,"192.168.1.10");
strcpy(Device1.data1,"DATA1:1234");
strcpy(Device1.data2,"DATA2:5678");
strcpy(Device1.data3,"DATA3:9012");
```

```
// pass the struct and filename for writing
if ( writeBinaryFile(Device1,&fileName[0]) != -1)
{ return (0);} else  { return (-1);}
```

Next page

Email : simhkeng@yahoo.com

Write Binary Format to File

```
int writeBinaryFile(struct devicePktData myDataStructure,   char *fileName)
{
FILE *fp;
int is;

fp=fopen(fileName,"a+b");
  if ( fp == NULL )
  {  printf("\n+++ Error : file open error +++\n");
     return(-1);  }

Is  =   fwrite(&myDataStructure,sizeof(myDataStructure),1,fp);
fclose(fp);
return(is);
}
```


Email : simhkeng@yahoo.com

Read Data from Binary File

```
int readBinaryFile(struct devicePktData *s,  char *fileName, int maxRecord)
{
   FILE *fp;
   int is;
   fp=fopen(fileName,"rb");
   if ( fp == NULL )
  {  printf("\n+++ Error : file open error +++\n");
     exit(-1);  }
      is =1;
    while( (fread(s,sizeof(*s),1,fp)>0) && (is <= maxRecord) )
    {
      //printf("Display from function= %d %s\n",is,s->name);
       is++;
       s++;    }
      fclose(fp);
      printf("Total record read =%d\n",is-1);
      return (is-1); }
```


Email : simhkeng@yahoo.com

Thank you

For Future Contact :

Sim H K

simhkeng@yahoo.com