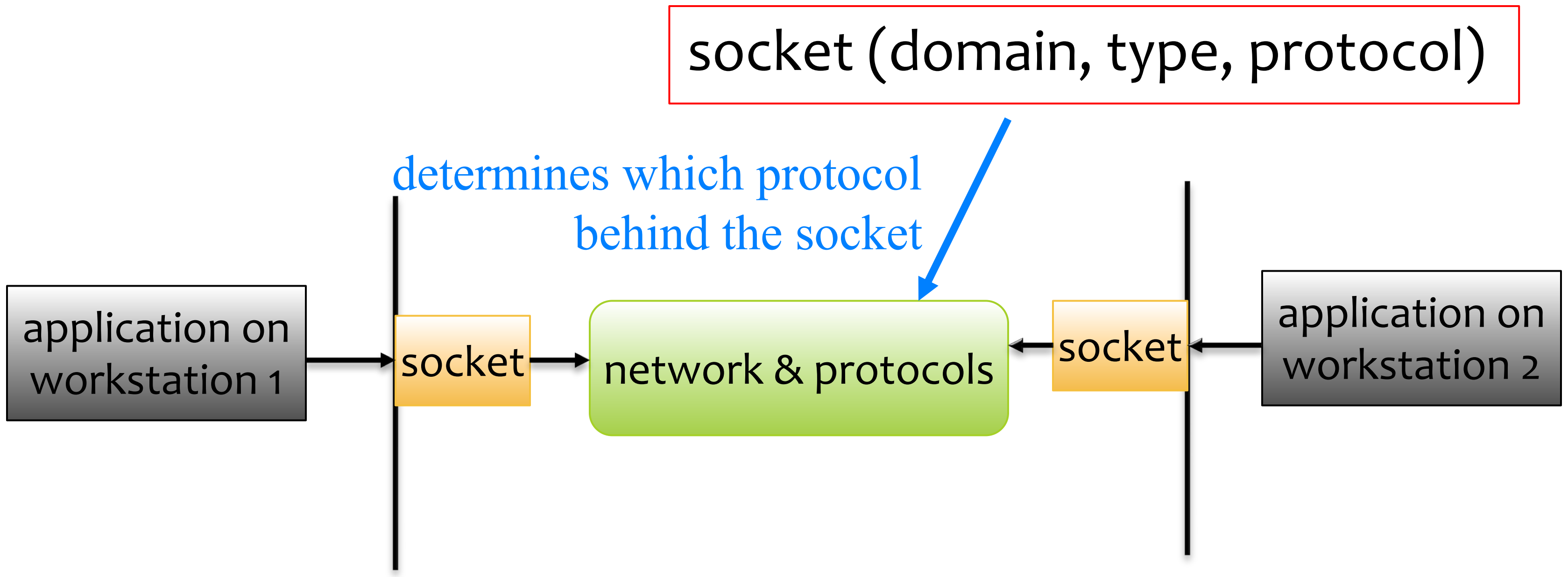
Overview on Socket API

Advisor: Cheng-Shang Chang

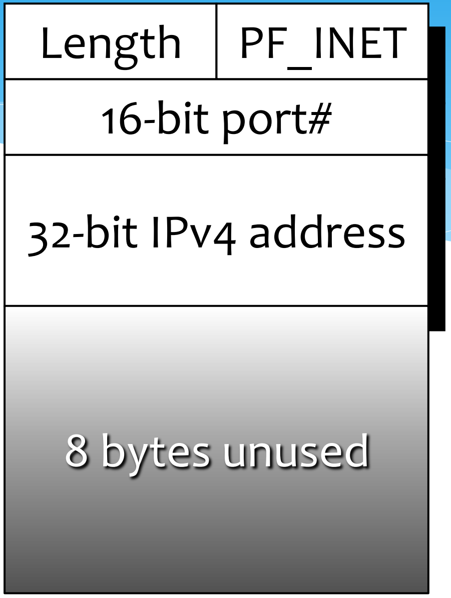
Socket Introduction

* A **network socket** is a network interface- an endpoint of an interprocess communication flow across a computer network.


socket (domain, type, protocol)
determines which protocol behind the socket
application on workstation 1
socket
network & protocols
socket
application on workstation 2

Socket Structure

```
Struct in_addr{
     in_addr_t s_addr;
}
struct sockaddr_in {
    uint8_t sin_len;
    sa_family_t   sin_family;
    in_port_t sin_port;
    struct  in_addr sin_addr;
    char    sin_zero[8];
};
```



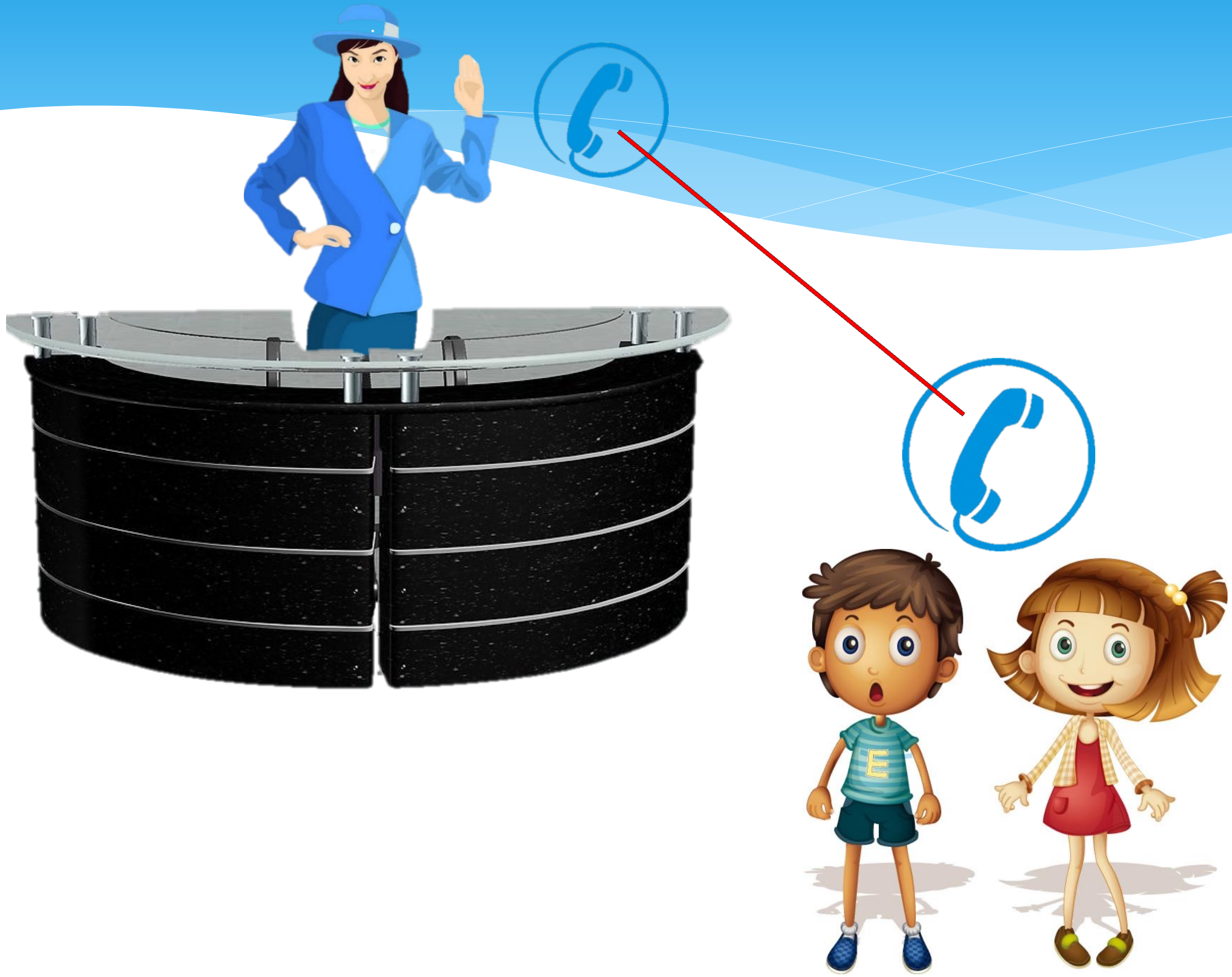
0-1023: well-know ports
1024-49151: registered ports
49152-65535: dynamic ports
http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

$$\begin{tikzcd} 140.114.26.111 \arrow[d] \\ 0\sim255: 8 \text{ bits} \end{tikzcd}$$

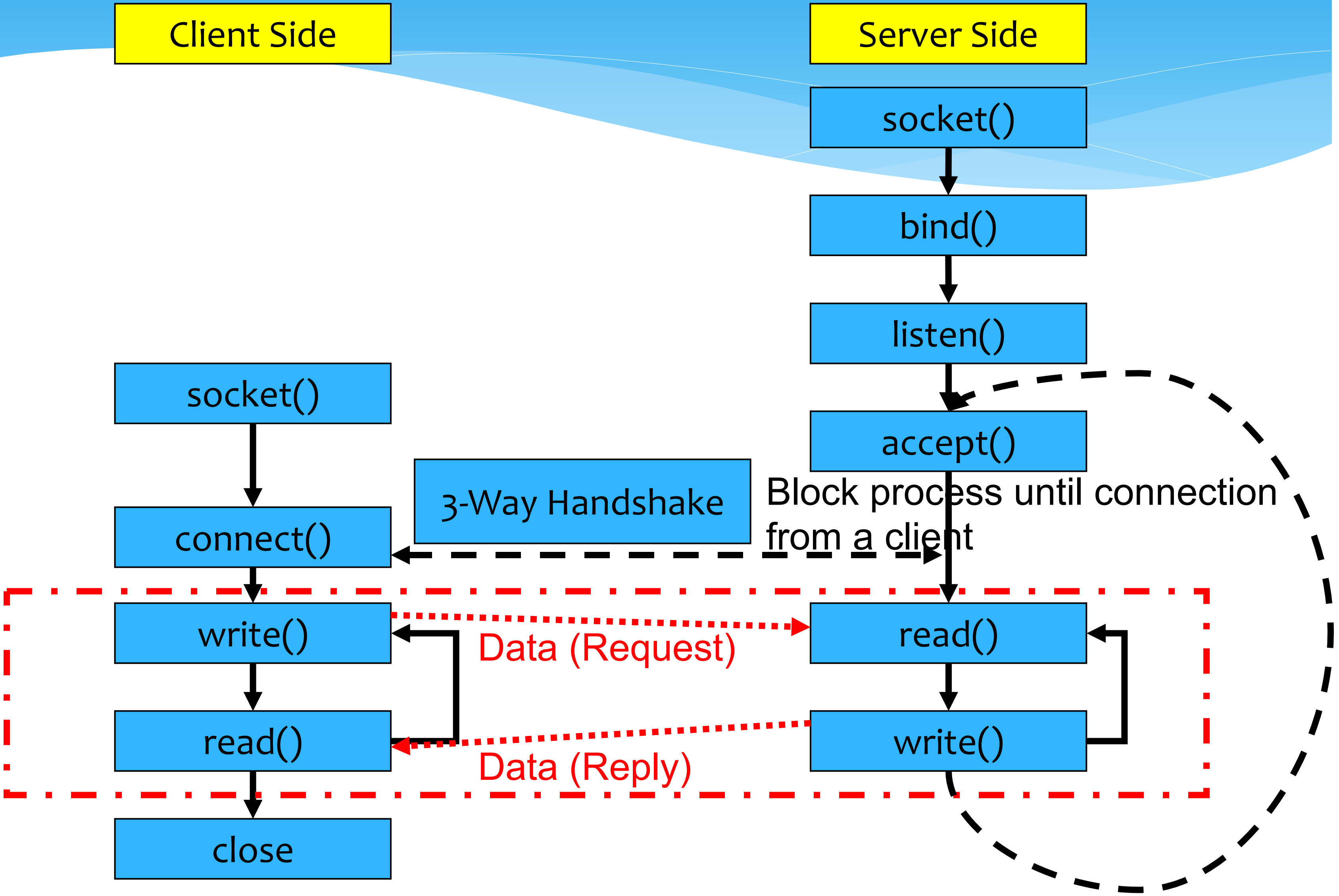
typedef uint16_t in_port_t;
typedef unsigned int uint16_t;
in_port_t : an unsigned integer type of exactly 16 bits.

typedef uint32_t in_addr_t;
typedef unsigned long int uint32_t;
in_addr_t : an unsigned long integer type of exactly 32 bits.

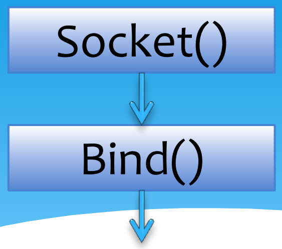
TCP Socket Programming



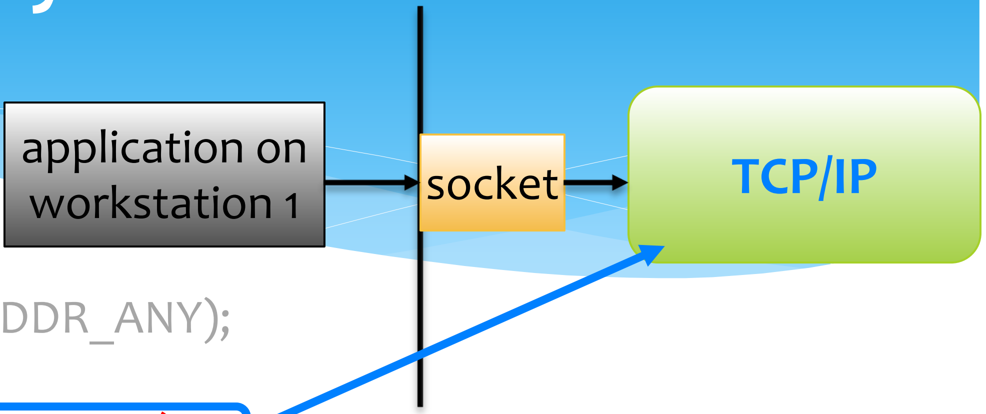
Flow Chart of TCP Setup



The Server Side

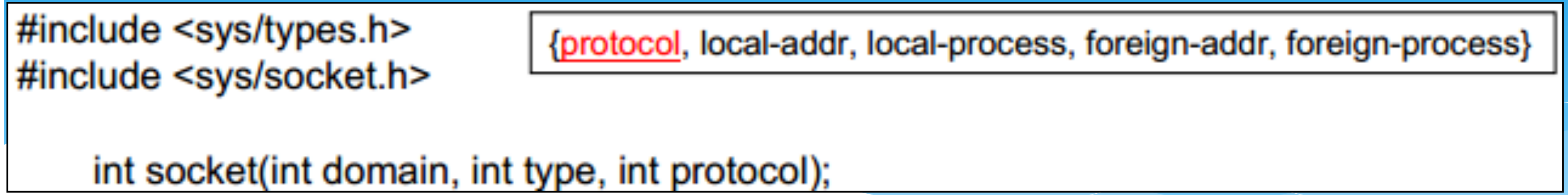


Socket System Call



```
myaddr.sin_family = PF_INET;
myaddr.sin_port = htons(5000);
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);

sockfd = socket (PF_INET, SOCK_STREAM, 0);
bind (sockfd, (struct sockaddr *) &myaddr, sizeof(struct sockaddr_in));
listen (sockfd, 10);
addr_size = sizeof (client_addr);
while(1){
   streamfd = accept (sockfd, (struct sockaddr *) &client_addr, &addr_size);

   status = read (streamfd, str_buf, 100);
   printf ("string from net: %s\n", str_buf);

   close(streamfd);
}
```

Create Socket Descriptor

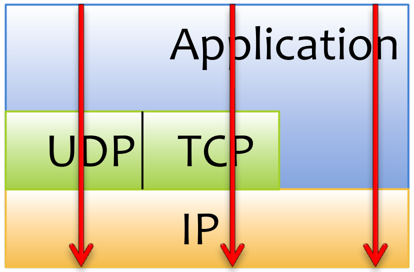


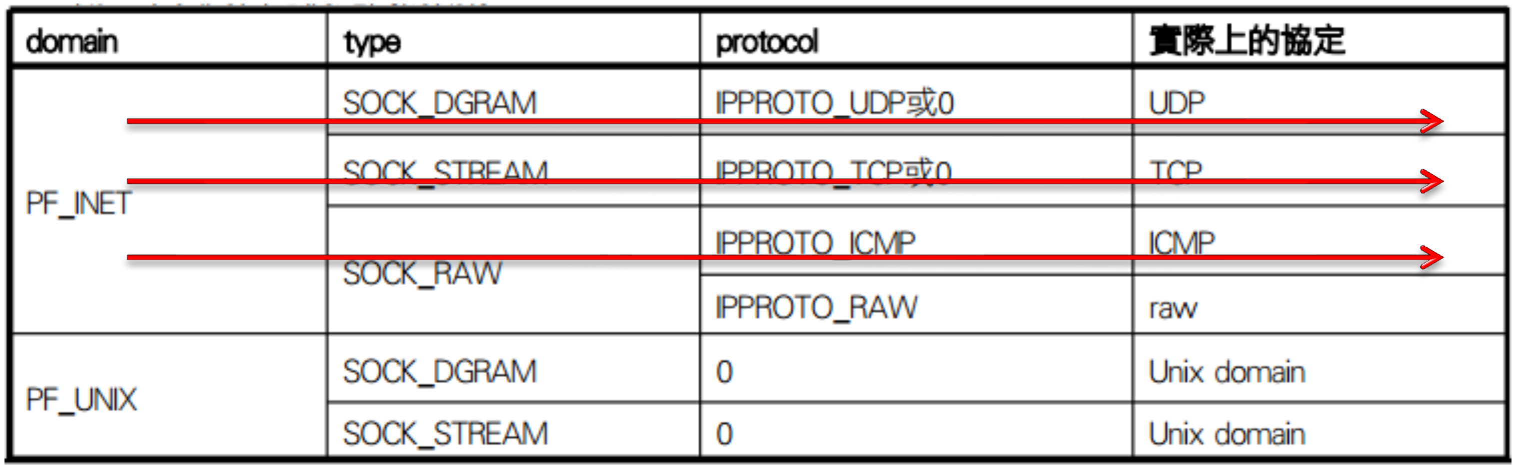
```
#include <sys/types.h>
#include <sys/socket.h>

        int socket(int domain, int type, int protocol);
```

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- socket() 建立socket，執行成功後回傳socket file descriptor
  執行失敗回傳-1
- socket() 三個參數介紹：

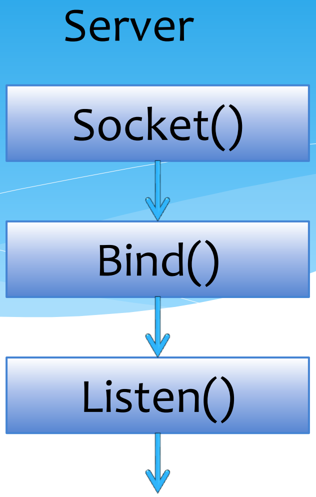
1.表示internet協定		**PF_INET**
2.連結的型態(TCP, UDP…)		**SOCK_STREAM**
3.通訊協定		**0**





domain	type	protocol	實際上的協定
PF_INET	SOCK_DGRAM	IPPROTO_UDP或0	UDP
	SOCK_STREAM	IPPROTO_TCP或0	TCP
	SOCK_RAW	IPPROTO_ICMP	ICMP
		IPPROTO_RAW	raw
PF_UNIX	SOCK_DGRAM	0	Unix domain
	SOCK_STREAM	0	Unix domain

Bind System Call and IP-address Setup



```
struct sockaddr_in myaddr,  struct sockaddr_in client_addr;
                ...
myaddr.sin_family = PF_INET;
myaddr.sin_port = htons(5000);
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

setup address

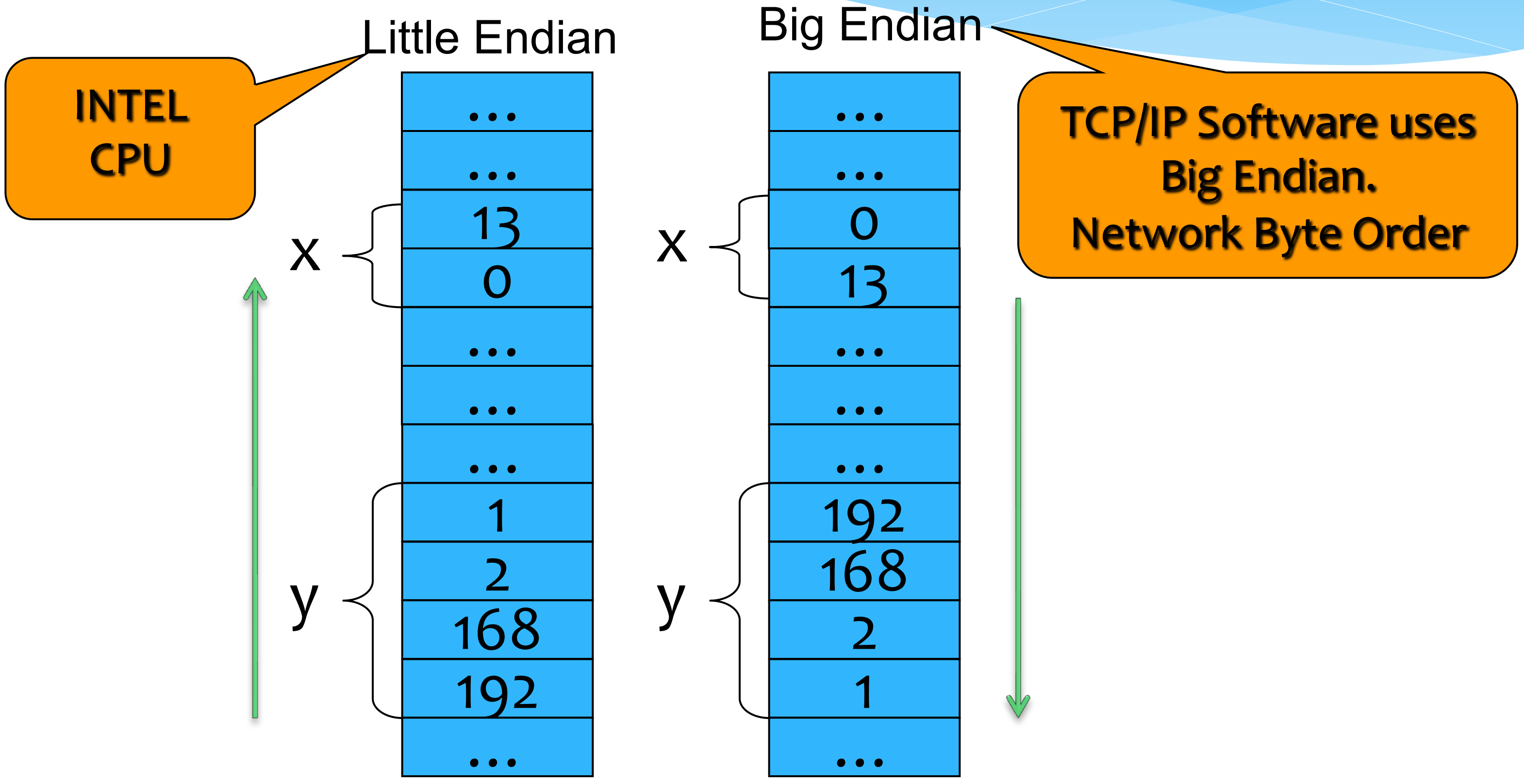
```
sockfd = socket (PF_INET, SOCK_STREAM, 0);
bind (sockfd, (struct sockaddr_in *) &myaddr, sizeof(struct sockaddr_in));
listen (sockfd, 10);
addr_size = sizeof (client_addr);
while(1){
   streamfd = accept (sockfd, (struct sockaddr *) &client_addr, &addr_size);

   status = read (streamfd, str_buf, 100);
   printf ("string from net: %s\n", str_buf);

   close(streamfd);
}
```

bind the server port and address

Byte Ordering Functions

short x=13; //16-bit integer

long y = $192 \times 256^3 + 168 \times 256^2 + 2 \times 256^1 + 1 \times 256^0$



Byte Ordering Function

struct sockaddr_in myaddr;

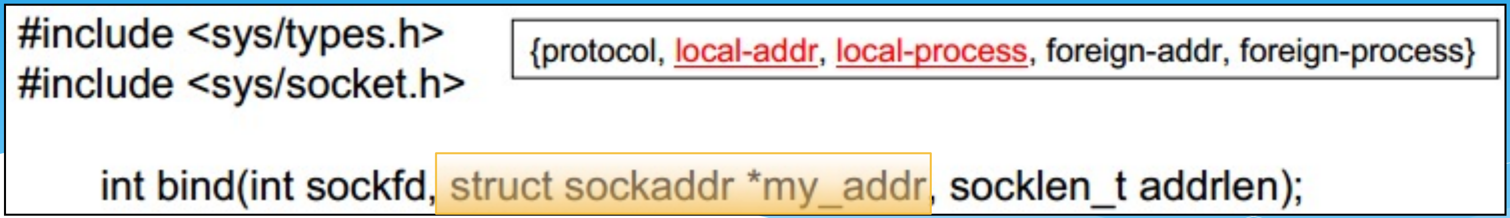
myaddr.sin_port = 13; this is not correct;

Network recognize the port number as 13*256+0*1;

- Byte ordering function
  - uint16_t htons(uint16_t *host16bitvalue*);
  - uint32_t htonl(uint32_t *host32bitvalue*);
    - Return value in network byte order
  - uint16_t ntohs(uint16_t *net16bitvalue*);
  - Uint32_t ntohl(uint32_t *net32bitvalue*);
    - Return value in host byte order

myaddr.sin_port = htons(13);

bind()



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

        int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
```

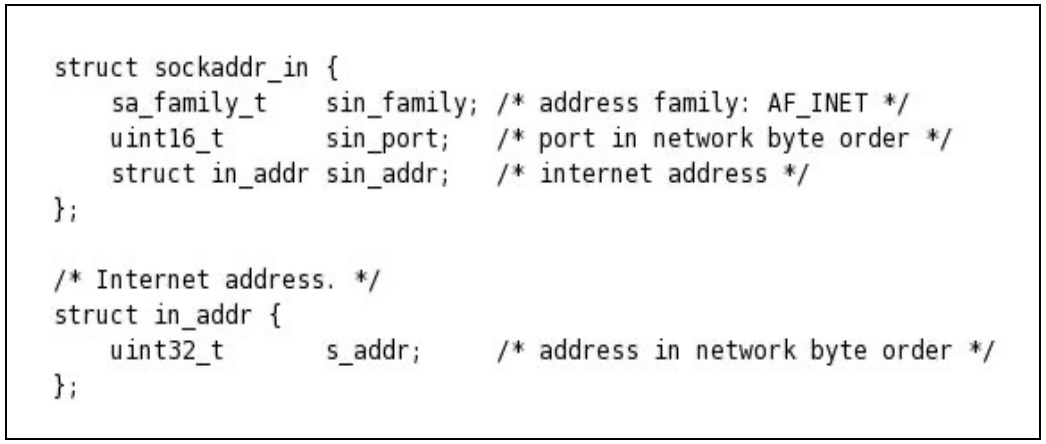
{protocol, local-addr, local-process, foreign-addr, foreign-process}

- bind() 將sockaddr結構連結到所建立的socket，當有封包抵達網路介面時，Linux核心便會將此封包導向到其連結的socket。
- bind() 三個參數介紹：
  1. socket()執行後傳回的socket descriptor
  2. 指向socket sockaddr_in結構的指標，用來存放連結的sockaddr位址結構
  3. 第二個參數的位址結構長度

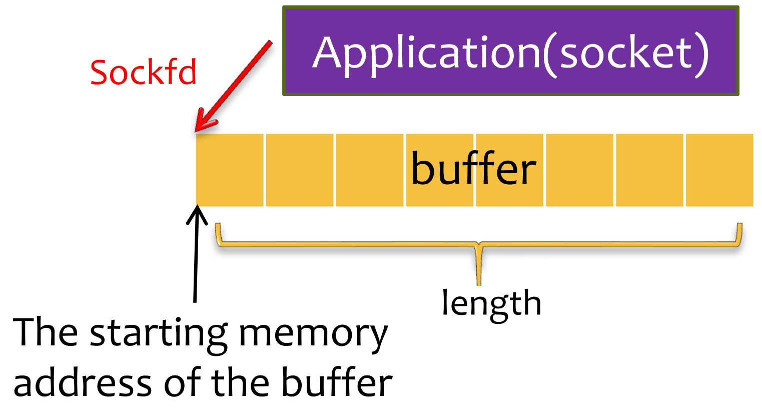
IP address設定為INADDR_ANY，
表示可 以接受任何Client主機的服務要求

$\longrightarrow$ Sockaddr_in

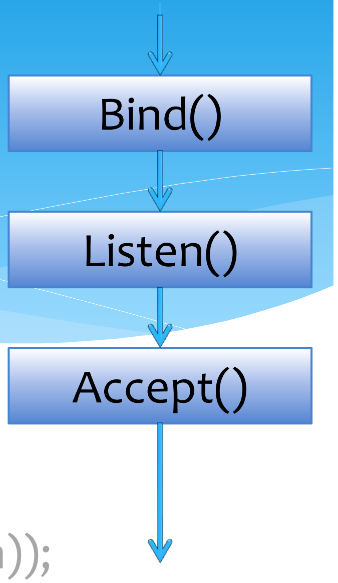
sa_family (e.g. PF_INET)	port#(e.g. 13)	address (e.g. INADDR_ANY)



```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    uint16_t       sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```



Listen Connection



```
myaddr.sin_family = PF_INET;
myaddr.sin_port = htons(5000);
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);

sockfd = socket (PF_INET, SOCK_STREAM, 0);
bind (sockfd, (struct sockaddr *) &myaddr, sizeof(struct sockaddr_in));
listen (sockfd, 10);
addr_size = sizeof (client_addr);
while(1){
   streamfd = accept (sockfd, (struct sockaddr *) &client_addr, &addr_size);

   status = read (streamfd, str_buf, 100);
   printf ("string from net: %s\n", str_buf);

   close(streamfd);
}
```

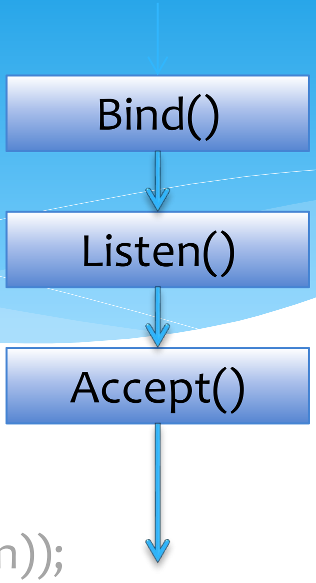
listen()開始監聽連線請求

```
#include <sys/socket.h>

    int listen(int s, int backlog);
```

- listen()：監聽socket connections 和限制接收連線
  queue的個數。
- listen()兩個參數介紹：
  1. socket()執行後傳回的socket descriptor
  2. 可以接受對該socket進行連線請求的個數(queue
size)

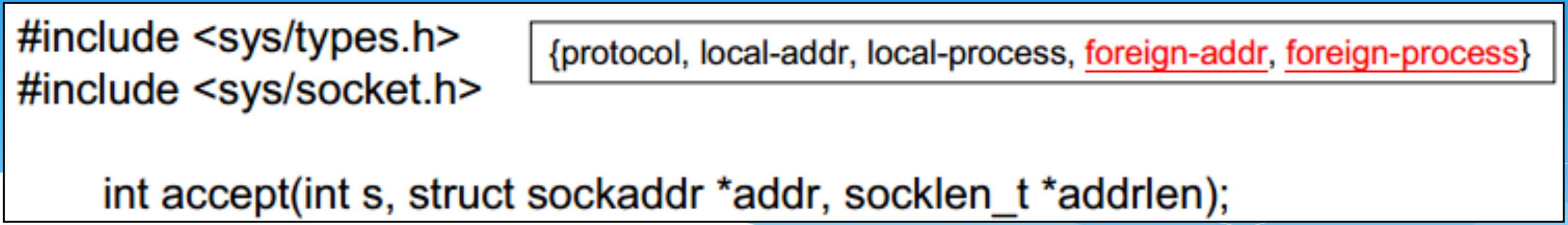
- Return value: 0 success, -1 error
- 每一個connection request會被accept()處理，尚未處理的connection request 會放入queue中等待，當queue滿時會產生connection refused。

Accept Connection



```
myaddr.sin_family = PF_INET;
myaddr.sin_port = htons(5000);
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);

sockfd = socket (PF_INET, SOCK_STREAM, 0);
bind (sockfd, (struct sockaddr *) &myaddr, sizeof(struct sockaddr_in));
listen (sockfd, 10);
addr_size = sizeof (client_addr);
while(1){
   streamfd = accept (sockfd, (struct sockaddr *) &client_addr, &addr_size);

   status = read (streamfd, str_buf, 100);
   printf ("string from net: %s\n", str_buf);

   close(streamfd);
}
```

accept()處理新連線



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

    int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
```

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- accept()用來處理連線請求，只有在TCP的server端呼叫。

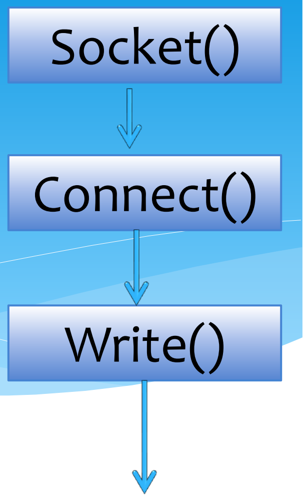
- accept() 三個參數介紹：
  1. socket()執行後傳回的socket descriptor
  2. 指向struct sockaddr_in結構的指標，用來存放client端的IP address
  3. 第二個參數的長度

- 當連線成功傳回client的socket descriptor，失敗則回傳 -1。

- 會從queue中拿取第一個connection request來處理，他會建立一個新的socket descriptor為之後用來操作該連線所用。
- 當queue中沒有connection request時，該函式預設會讓process block。

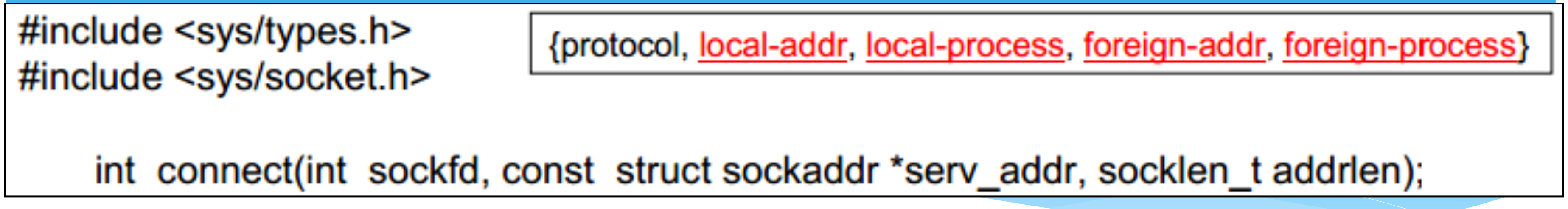
The Client Side

Connect to Server



```
struct sockaddr_in server_addr;
int sockfd, status;
//setup the server address
server_addr.sin_family = PF_INET;
server_addr.sin_port = htons(5000);
server_addr.sin_addr.s_addr = inet_addr ("127.0.0.1");
//connect to the server
sockfd = socket (PF_INET, SOCK_STREAM, 0);
connect (sockfd, (struct sockaddr *) &server_addr, sizeof(struct sockaddr_in));

status = write (sockfd, "Hello!", strlen("Hello")+1);

close(sockfd);
```

connect()連線到另一端的Socket

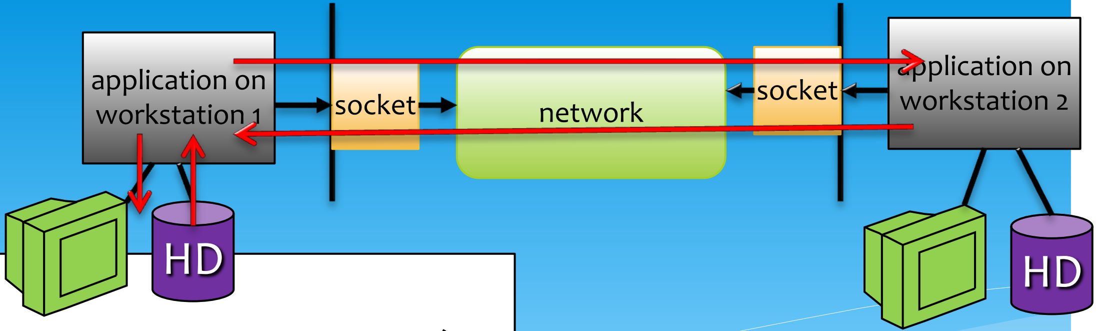


```
#include <sys/types.h>
#include <sys/socket.h>

    int  connect(int  sockfd, const  struct sockaddr *serv_addr, socklen_t addrlen);
```

{protocol, local-addr, local-process, foreign-addr, foreign-process}

- TCP client端建立socket後，可用connect()向TCP server端要求建立連線，在確定連線後，client端和sever端就能開始互相傳送資料。
- connect() 三個參數介紹：
  1. socket()執行後傳回的socket descriptor
  2. 指向struct sockaddr_in結構的指標，用來存放server的address
  3. 第二個參數的位址結構長度
- Return value: 0 success, -1 error

read/write()



```
#include <unistd.h>
        ssize_t read(int sockfd, void *buf, size_t count);
```

- **read**() ： attempts to read up to *count* bytes from socket file descriptor sockfd into the buffer starting at *buf*.
  If *count* is zero, **read**() returns zero and has no other results.
  If *count* is greater than SSIZE_MAX, the result is unspecified.

```
#define SSIZE_MAX LONG_MAX
#define LONG_MAX 0x7FFFFFFFL
```

- read() 三個參數介紹：
  1. (server side) accept()成功傳回client的socket descriptor
     (client side) socket()執行後傳回的socket descriptor
  2. 指向字元暫存器的指標，用來存放讀取到的資料
  3. 欲接收的資料量長度

- Success: the number of total bytes, failed: -1

```
#include <unistd.h>
        ssize_t write(int sockfd, const void *buf, size_t count);
```

- **write**() writes up to *count* bytes to the server/client referenced by the socket file descriptor *sockfd* from the buffer starting at *buf*.

- write() 三個參數介紹：
    1. (server side) accept()成功傳回client的socket descriptor
       (client side) socket()執行後傳回的socket descriptor
    2. 儲存資料的暫存器
    3. 欲傳送資料量的長度

- Success: the number of total bytes, failed: -1

close()關閉socket

```
#include <unistd.h>
            int close(int sockfd);
```

close()當應用程式不再使用socket出入口當做資料傳送時，需關閉socket

close() ：參數是socket()執行後傳回的socket descriptor

Return value: 0 success, -1 error

TCP Server Template

```
Int main (){
        struct sockaddr_in myaddr, sockaddr_in client_addr;
        int sockfd, streamfd, port, status, int addr_size;;
        char str_buf[100];
        bzero (&myaddr, sizeof(myaddr));
        myaddr.sin_family = PF_INET;
        myaddr.sin_port = htons(5000);
        myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        sockfd = socket (PF_INET, SOCK_STREAM, 0);
        bind (sockfd, (struct sockaddr *) &myaddr, sizeof(struct sockaddr_in));
        listen(sockfd, 10);
        addr_size = sizeof (client_addr);
        while(1){
          streamfd = accept (sockfd, (struct sockaddr *) &client_addr, &addr_size);

          handle_client(connfd);    //Call the procedure you wish to perform

          close(streamfd);
        }
  return 0;
}
```

TCP Client Template

```
int main (){
        struct sockaddr_in server_addr;
        int sockfd, status;
        //setup the server address
        server_addr.sin_family = PF_INET;
        server_addr.sin_port = htons(5000);
        server_addr.sin_addr.s_addr = inet_addr ("127.0.0.1");
        //connect to the server
        sockfd = socket (PF_INET, SOCK_STREAM, 0);
        connect (sockfd, (struct sockaddr *) &server_addr, sizeof(struct sockaddr_in));

        Server_Request_Procedure(sockfd); //Call procedure you wish to perform


        close(sockfd);
        return 0;
}
```

Final Project

Please implement a server program that waits a connection request, and a client program that connects to server.
Implement two commands：

- DNS: A client sends an URL address, and server returns an ip address
- QUERY: A client sends a student ID, and server returns the email of the student.

Submission

1. Source code(including readme)--70%
2. Report: --30%
   1. Screenshot the results.
   2. Experience

- Upload to eLearn, No paper Report
- Deadline：**6/24 23:59**

Reference

http://www.cis.nctu.edu.tw/~gis88507/course/linux/10_socket.pdf
http://en.wikipedia.org/wiki/Network_socket
http://www.tutorialspoint.com/index.htm
http://163.25.101.87/wiki/doku.php?id=course:2010_fall:unix_programming
http://www.vr.ncue.edu.tw/esa/EmbeddedSystemProgramming2010/ch07.htm