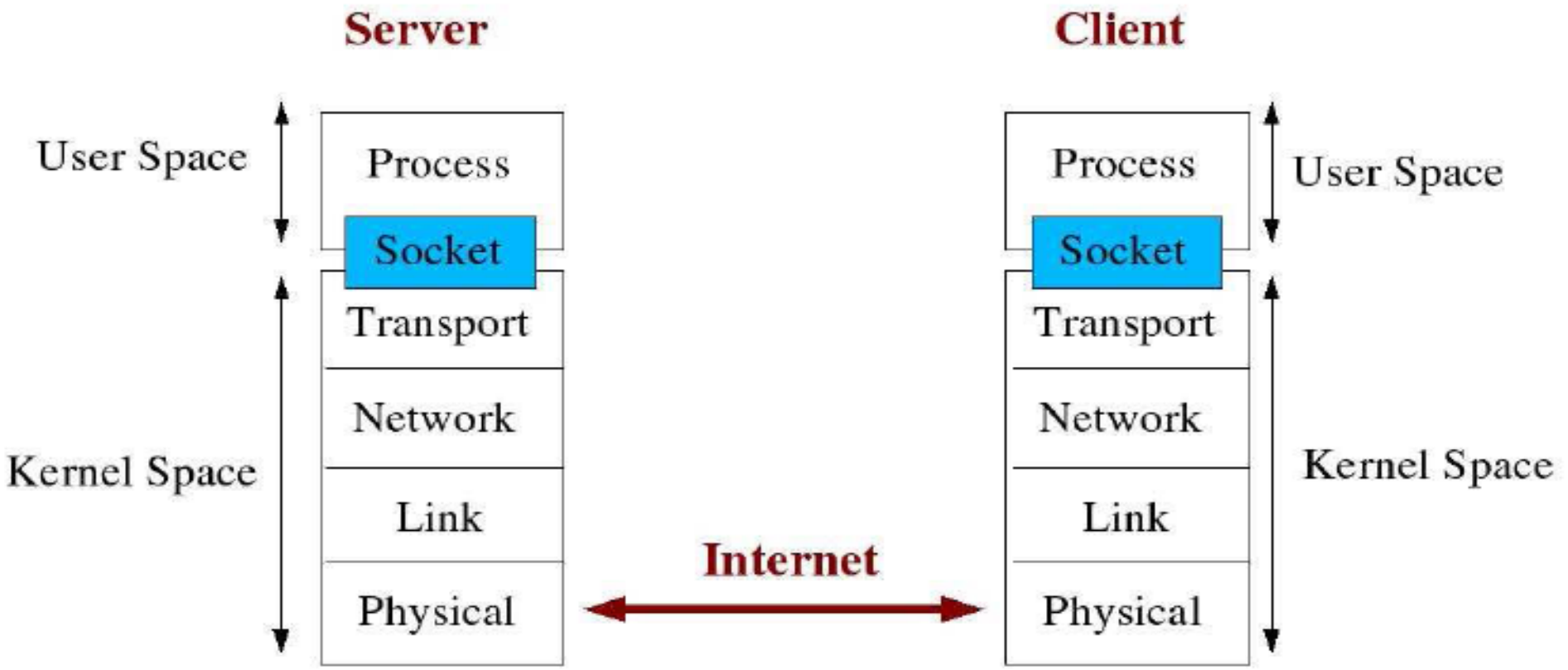
Basics of Socket Programming

Please check the referenced links for the further description and examples.


Server
Client
User Space
Process
Socket
Transport
Network
Link
Physical
Kernel Space
Internet
Process
Socket
Transport
Network
Link
Physical
User Space
Kernel Space

Procedures for Socket Implementation

1. Create the server application (e.g. a simple shellscript)
2. Test the server application at the command line
3. Decide on a port number and service name
4. Declare the port and name in /etc/services
5. Create a file for this service in /etc/xinetd.d or modify the file /etc/inetd.conf
6. Restart xinetd or inetd
7. Telnet into the service, and see the server app's output
8. Write your client program

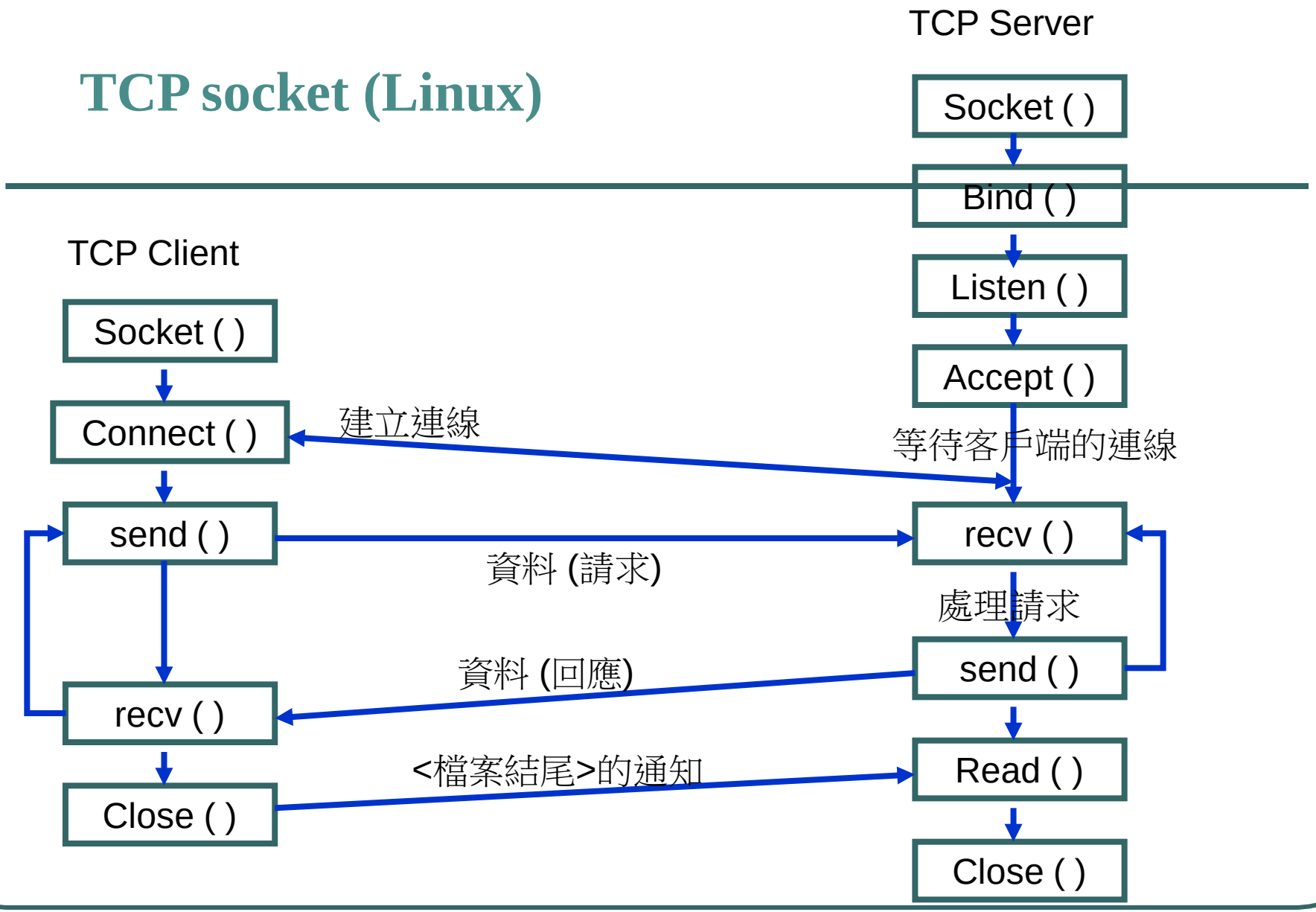
An example of a shell script on server

```
#!/bin/bash
/bin/echo -n "Hello World:" | /usr/bin/tee /tmp/log.log
/bin/date | /usr/bin/tee -a /tmp/log.log
```

Results from using telnet as a client to link to the server

```
[myuid@mydesk myuid]$ telnet 192.168.100.2 3333
Trying 192.168.100.2...
Connected to 192.168.100.2.
Escape character is '^]'.
Hello World:Tue Jan 23 13:33:55 EST 2001
Connection closed by foreign host.
[myuid@mydesk myuid]$
```

TCP socket (Linux)



Create a Socket

- 進行網路傳輸前，程式必須先呼叫socket函式
- Socket函式

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int family, int type, int protocol) ;
```

Parameters

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols

type	Description
SOCK_STREAM	stream socket (TCP)
SOCK_DGRAM	datagram socket (UDP)

protocol	Description
0	Normally set to 0

Connect function

- TCP客戶端使用connect函式來建立與TCP伺服端的連線

```
#include <sys/socket.h>
#include <sys/types.h>
int connect (int sockfd, const struct sockaddr_in *servaddr, int addrlen) ;
```

  - *sockfd* 是 socket 函式傳回的socket descriptor
  - 而第二、三個參數則是socket位址結構的指標及其長度

socket structures

- struct sockaddr: Holds socket address information for many types of sockets

```
struct sockaddr {
        unsigned short  sa_family;    //address family AF_xxx
        unsigned short  sa_data[14]; //14 bytes of protocol addr
}
```

- struct sockaddr_in: A parallel structure that makes it easy to reference elements of the socket address

```
struct sockaddr_in {
        short int          sin_family;   // set to AF_INET
        unsigned short int sin_port;     // Port number
        struct in_addr     sin_addr;     // Internet address
        unsigned char      sin_zero[8];  //set to all zeros
}
```

Bind function

- bind 函式會將本機協定的IP和Port指定給socket；在Internet協定中的協定位址是32位元的IPv4位址、或128位元的IPv6位址，加上16位元的TCP或UDP埠號。

```
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr_in *myaddr, socklen_t
   addlen) ;
```

  - 第二個參數是特定協定位址的指標
  - 第三個參數則是該位址結構的長度
  - returns 0 if successful or -1 on error.

Example

```
struct sockaddr_in serv;

serv.sin_family = AF_INET;
serv.sin_addr.s_addr = htonl(“127.0.0.1”);
serv.sin_port = htons(3000);

bind(serv_sd, &serv, sizeof(serv));
```

Listen

- 只有TCP伺服端會呼叫listen函式
  - 函式的第2個參數會指定kernel應該為此socket暫存到佇列中的最大連線數目

```
#include <sys/socket.h>
int listen (int sockfd, int backlog) ;
```

Accept

- TCP伺服端會呼叫accept，從已完成連線佇列最前端傳回下個已完成的連線。如果已完成連線佇列是空的，行程就會進入睡眠

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr_in *cliaddr,
  socklen_t *addrlen) ;
```

  - *cliaddr*與*addrlen*參數是用來傳回連線客戶端的協定位址；*addrlen*是值－結果參數：在呼叫前，將**addrlen*指向的整數值設為*cliaddr*指向之socket位址結構的長度；在傳回時，整數值會包含kernel在socket位址結構中實際存放的位元組數目

accept styles

There are basically three styles of using accept:
1. Iterating server: Only one socket is opened at a time. When the processing on that connection is completed, the
socket is closed, and next connection can be accepted.
2. Forking server: After an accept, a child process is forked off to handle the connection. Variation: the child processes
are preforked and are passed the socketId.
Concurrent single server: use select to simultaneously wait on all open socketIds, and waking up the process only when
new data arrives.
3. Concurrent single server: use select to simultaneously wait on all open socketIds, and waking up the process only when
new data arrives.

Pro and Con of Accept styles

Iterating server is basically a low performance technique since only one connection is open at a time.

Forking servers enable using multiple processors. But they make sharing state difficult, unless performed with threads. Threads, however present a very fragile programming environment.

Concurrent single server: reduces context switches relative to forking processes and complexity relative to threads. But does not benefit from multiprocessors.

Send

- TCP伺服端或客戶端呼叫send傳送資料

```
#include <unistd.h>
int count=send( int sockfd, char *buffer, int buflen, int flags )
```

  - count:傳送資料的大小
  - buffer:資料傳送的指標
  - buflen:資料長度
  - flag:通常設為0
  - Or write
    - int write(int sockfd, char * buffer, int buflen);

recv

- TCP伺服端或客戶端呼叫recv接收資料

```
#include <unistd.h>

Int count=recv( int sockfd, char *buffer, int buflen, int flags )
```

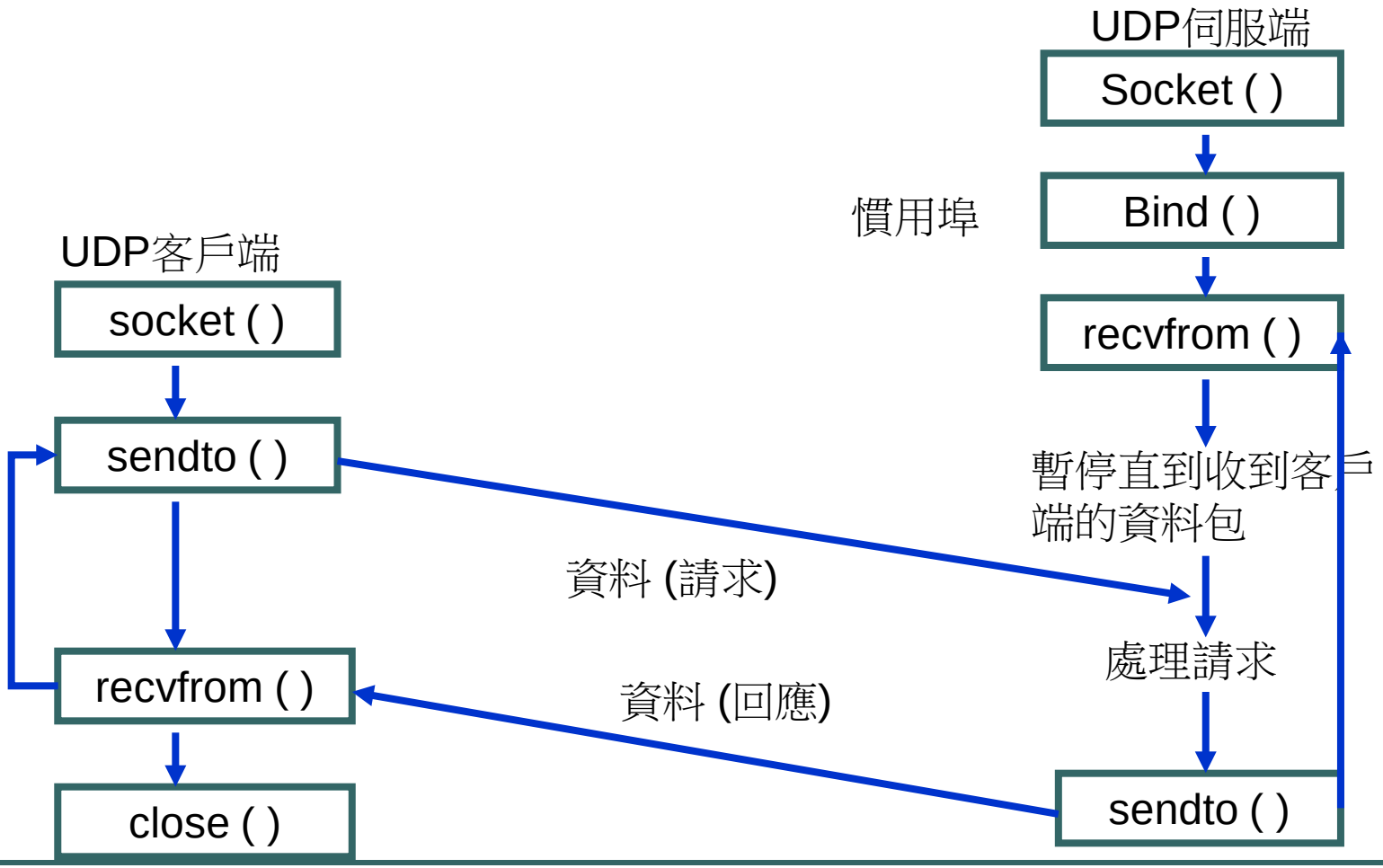
  - Count:接收到的資料大小
  - buffer:資料接收後儲存位址的指標
  - buflen:資料長度
  - flag:通常設為0
  - Or read
    - int read(int sockfd, char * buffer, int buflen);

Close

- 用來關閉socket，並且終止TCP連線

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

UDP socket (Linux)



Socket 在TCP 與 UDP 上的差異

- 由於TCP與UDP這兩個傳輸層本身的差異
  - UDP是非連線式、不可靠的資料包協定
  - TCP為連線導向式、可靠的位元組串流
- 因此使用TCP與使用UDP所撰寫的應用程式之間，有些基本的差異
- UDP並不會建立與伺服端的連線，而只是使用sendto函式將資料包傳送給伺服端：這函式只需目的位址為參數
- 而伺服端也不接受來自客戶端的連線，而是呼叫recvfrom函式，等待資料由客戶端送達，並傳回客戶端的協定位址和資料包，使伺服端傳送回應給正確客戶端。

revfrom與sendto 函式

- 這兩個函式類似標準的read與write函式

```
#include <sys/socket.h>
int recvfrom (int sockfd, const void *buff, size_t nbytes, int flags,
struct sockaddr *from, socklen_t *addrlen) ;
```

```
int sendto (int sockfd, const void *buff, size_t nbytes, int flags, const
struct sockaddr *to, socklen_t addlen) ;
```

  - 前三個參數*sockfd*、*buff*、*nbytes*與read及write的前三個參數相同，分別是描述子、指向讀取或寫入緩衝區的指標以及讀取或寫入的位元組收數目。
  - sendto 的*to*參數是socket位址結構，指定資料所要送往的協定位址， *addrlen*則是指定這個socket位址結構的長度;

1st Homework

- Go through the article for building a socket program as in the following web site:

  http://www.troubleshooters.com/codecorn/sockets/

- Refer to Socket Programming Tutorial & Introduction, such as

  http://www.rites.uic.edu/~solworth/sockets.pdf
  http://www.docstoc.com/docs/2156175/Socket-Programming-Tutorial

- Prepare for the demonstration on Oct. 13/14.