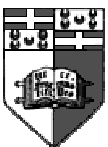




# Non-Blocking I/O

- To set all socket I/O as non blocking (including the *accept()* call), use the normal *fcntl()* function shown below.
- When reading or writing to a non-blocking socket, if the operation is not possible the *read()* and *write()* calls return  $-1$  with *errno* set to **EAGAIN**..

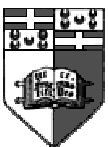
```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
int fcntl(int sockfd, F_SETFL,O_NONBLOCK);
or
int fcntl(int sockfd, F_SETFL,O_NDELAY);
returns  $-1$  on error
```





# Non-Blocking I/O (cont)

- Many implementations vary in their response to a non-blocking *connect()* and *accept()* system call.
- On SunOS systems a failed non-blocking *connect()* sets *errno* to EWOULDBLOCK, showing such a call would normally block, or sets *errno* to EINPROGRESS, showing the connection is not yet established but will be ready soon.
- On SunOs systems, a non-blocking *accept()* sets *errno* to EWOULDBLOCK if *accept* cannot establish a connection immediately.
- On Linux systems in non-blocking mode, a failed *connect()* always returns an error of EINPROGRESS in *errno* while a failed *accept()* returns an error of EAGAIN in *errno*.

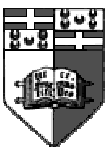




# Input Peeking

- *recv()* allows you to look at incoming data without removing it from the receive data buffer.
- Any data peeked will still be returned by the next *read()* system call.

```
int recv(int sockfd, char *buff, int nbytes,  
        MSG_PEEK);  
        returns -1 on error or amount  
        of bytes peeked
```



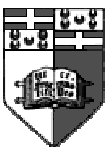


## Connection Information

- Sometimes, we might want to refresh our memory of who we are connected to.
- *getpeername()* gives us the details of the foreign part of the socket tuple.
- *addrlen* needs to be set to the size of peer and on return it will contain the amount of data used up in peer.

```
int getpeername(int sockfd, struct sockaddr *peer,  
                int *addrlen);
```

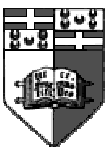
returns -1 on error





# I/O Multiplexing

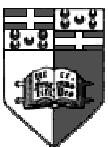
- Sometimes we want to check if a specific socket is available for reading or writing from a selection of open sockets.
- If we have several sockets open, we can have a child for each socket testing the availability of each socket and then passing the data to the parent using some form of IPC.
- Alternatively we can have the parent polling through each socket after setting them to non-blocking.
- A far better method is provided by 4.3BSD where a special system call can return the status of multiple system calls.
- Upon retrieving this status information, one can act on the specific available sockets.



# *Unix* File Descriptor Masks

- These are similar to socket masks used in `sigprocmask`.
- Each bit in the mask represents one of all file descriptors possible. (ex `fd=0` is bit 0, `fd=1` is bit 1, etc.).
- The type `fd_set` is guaranteed to have a bit for all possible file descriptors. (it is usually implemented as an array of `int`).
- Always call `FD_ZERO` on file descriptor masks before using.

```
FD_ZERO(fd_set *set);  
           clears all bits in set  
FD_SET(int fd, fd_set *set);  
           turn bit for fd on  
FD_CLR(int fd , fd_set *set);  
           turn bit for fd off  
FD_ISSET(int fd, fd_set *set);  
           test bit for fd (0 or 1)
```



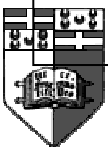


# select()

- *select()*'s behavior depends on the timeout value given:
  - *timeval* values equal to zero make *select()* return immediately. Result is 0 if no socket is available to read or write.
  - *timeval* values not equal to zero make *select()* block until a socket descriptor is available or until a timeout occurs (poll).
  - *timeout* equals to NULL, makes *select()* block until a socket descriptor becomes available to read or write.

```
struct timeval {  
    long tv_sec;  
    long tv_usec;  
}
```

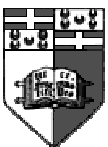
```
int select(int maxfdpl, fd_set *read,  
          fd_set *write, fd_set *exceptfd,  
          struct timeval *timeout);  
    returns number of descriptors set,  
    or 0 if timeout occurs  
    or -1 on error
```





## select() (cont)

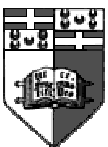
- If we want to ignore one of the parameters (read, write or except), just set these to NULL.
- *maxfdpl* should be set to the maximum file descriptor number being used + 1.
- *maxfdpl* can be set to `FD_SETSIZE` which is the maximum possible number assigned to file descriptors.
- After a successful return from *select()* (return value larger than zero), we can test the file descriptors using `FD_ISSET`.
- If we want to check what sockets are available to open a connection so that *accept()* won't block we can also use *select()*.
- A socket which has a connection pending on it, is marked as ready for reading.





# inetd

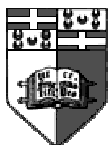
- *inetd* is a special background process which waits for a connection on several ports.
- The ports it listens on are listed in */etc/services*.
- When a connection is opened, it forks a new child and then calls *exec* to run the service. The details of each service are listed in */etc/inetd.conf*.
- One way of implementing *inetd* is to use *select()* which can listen on multiple ports for a connection.



# *Unix* *bind()* addendum

- When calling *bind()*, we might want the system to allocate the IP address of the local host automatically.
- `INADDR_ANY` is a constant which is allocated to the IP address of the local host.
- For a host with multiple IP addresses (multihomed), `INADDR_ANY` will be allocated to all available addresses resulting in a socket bound to more than one address.
- Prior to calling *bind()*, a server can use this as follows:

```
struct sockaddr_in serv_addr;  
serv_addr.sin_addr.s_addr =  
    htonl(INADDR_ANY);
```



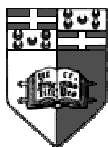


# DNS access

- *gethostbyname()* gives you a *hostent* structure containing the network address of the given name.
- *hostname* can be a hostname or an IP address in dot format.
- On some systems, if the name cannot be matched with an IP address, *gethostbyname()* returns  $-1$  with *errno* set to `HOST_NOT_FOUND`.
- Returned addresses are in network byte order.

```
#define <netdb.h>
struct hostent {
    char *h_name; //official name of host
    char **h_aliases; // alias list
    int h_addrtype; //AF_INET
    int h_length; // for IP = 4
    char **h_addr_list; //NULL terminated
                        // list of addresses
};
#define h_addr h_addr_list[0] //first addr

struct hostent *gethostbyname(char *hostname);
                        returns NULL on error
```





# Exercises

- Build a server that communicates on a socket and when the client sends strings ending with the @ character, it spawns off a child that outputs the string to screen.
- Using non-blocking I/O, build a server that polls multiple ports for a connection and when a client connects on one of them, spawn off a child to deal with it.
- Modify the above server such that it continuously outputs the number of established connections to screen.
- Implement the second question above using a blocking version of *select()*.
- Make the above server exit if no connections occur during a span of 1 minute.
- Find out the IP address of *yahoo.com* and *lycos.com*.

