I/O Multiplexing – shutdown() and poll() functions

Lecture 6

Introduction

- The normal way to terminate a network connection is to call close( ) function

- But there are two limitations with close that can be avoided with shutdown
  1. close( ) decrements the descriptor's reference count and closes the socket only if the count reaches 0.
  2. close( ) terminates both directions of data transfer, reading and writing.

     Since a TCP connection is full duplex, there are times when we want to tell the other end that we have finished sending, even though that end might have more data to send us

shutdown function -1/2

- Syntax
  #include <sys/socket.h>
  int shutdown (int *sockfd*, int *howto*);

- Return value
  - 0 if OK
  - -1 on error

shutdown function -2/2

- The action of function depends on value of *howto* argument
  - SHUT_RD
    - The read-half of connection is closed
    - No more data can be received on the socket and any data currently in the socket receive buffer is discarded
  - SHUT_WR
    - The write-half of connection is closed
    - Any data currently in the socket send buffer will be sent, followed by TCP's normal connection termination sequence
  - SHUT_RDWR
    - Read-half and write-half of connection are closed

Sample program

Kindly go through the slides by yourself

poll() function – 1/5

- Syntax

  #include <poll.h>

  int poll (struct pollfd **fdarray*, unsigned long *nfds*, int *timeout*);

- Return value
  - Count of ready descriptors
  - 0 on timeout
  - -1 on error

poll() function – 2/5

- The first argument is pointer to first element of array of structures

- Each element of array is pollfd structure that specifies conditions to be tested for a given descriptor fd

```
struct pollfd
{
    int fd;  //descriptor to check
    short events; //events if interest on fd
    short revents; //events that occurred on fd
}
```

poll() function – 3/5

- The conditions to be tested are specified by *events* member, and function returns status for that descriptor in corresponding *revents* member
- Number of elements in array of structures is specified by *ndfs*
- The *timeout* specifies how long function will wait before returning
  - INFTIM : wait forever (negative value)
  - 0 : return immediately
  - >0 : wait specified no. of milliseconds

poll() function – 4/5

- If we are no longer interested on a particular descriptor, we set the fd member of pollfd structure to negative value
- Then *events* member is ignored, and *revents* member is set to 0 on return
- With poll( ), we must allocate an array of pollfd structures inorder to maintain the client information

poll() function – 5/5

- We handle the *fd* member of this array the same way we handled the client[ ] array
- Value of -1 means entry is not in use, otherwise it is the descriptor value
- Each of these two members is composed of one or more bits that specify a certain condition

pselect() function – 1/3

- Syntax

  #include <sys/select.h>

  #include <signal.h>

  #include <time.h>

  int pselect (int *maxfd*, fd_set **readset*, fd_set **writeset*, fd_set **exceptset*, const struct timespec **timeout*, const sigset_t **sigmask*);

- Return value
  - Count of ready descriptors
  - 0 on timeout
  - -1 on error

pselect() function – 2/3

- The pselect contains two changes from normal select
  - It uses the timespec structure
    - Difference in these two structures is with the second member
    - The tv_nsec of the newer structure specifies nanoseconds but older specifies microseconds

pselect() function – 3/3

- It adds a sixth argument: a pointer to signal mask
  - Allows the program to disable the delivery of certain signals, tests some global vars that are set by the handlers for these now-disabled signals, and then call pselect, telling it to reset the signal mask