Section Notes
Use of select
Project

Scott Schremmer

Select and time outs

- Problem:
  - Reading/writing to a socket blocks while waiting for data
  - You might want to:
    - Wait for some amount of time, then give up
    - Wait for data on more than one socket at the same time
- Solution
  - Select!
  - Inputs a set of file descriptors and a time out
  - Returns when one or more of the files/sockets is no longer blocking

select()

- int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
  - nfds -- This should be one greater than the highest file descriptor considered
  - readfds, writefds, exceptfds
    - lists of file descriptors
      - ready for reading, writing or have an exceptional condition
    - fd_set format -- use macros to modify
    - can be set to NULL (if all set to NULL only timeout matters)

select()

- macros for fdset
  - FD_ZERO(&fdset)
    - initializes an fdset
  - FD_SET(fd,&fdset)
    - adds fd to the set
  - FD_CLR(fd,&fdset)
    - removes fd from the set
  - FD_ISSET(fd,&fdset)
    - is true (>0) when fd is in the set

timeout

- struct timeval *timeout
- struct timeval {

```
long tv_sec; /*seconds*/
long tv_usec; /*microseconds*/
}
```

- this is the amount of time to wait for a non-blocking state

Select

- Returns the number of file descriptors which are now non-blocking
  - reads / writes (depending on list) will not block
- the various lists are modified to contain only descriptors in a non-blocking state
- If time out is reached returns 0

Select

- sample call for sockfd from which we want to read

```
#define TIMEOUT 10
fd_set read_set;
struct timeval time_out;
...
time_out.tv_sec=TIMEOUT;
time_out.tv_usec=0;
FD_ZERO(&read_set);
FD_SET(sockfd,&read_set);
n=select(sockfd+1,&read_set,NULL,NULL,&time_out)
....
```

n=0 if time out 1 if data to be read

Network vs. host byte ordering

- Your machine might of might not agree with network ordering of bits (big-endian, little endian)
- A set of functions is provided to convert
- These have no effect on some machines.
  - Use anyway for code portability!
- examples
  - htons --> read “host to network short”
  - ntohs --> read “network to host short”
  - htonl,ntohl --> for long (32 bits)

Extracting port information

- From sample server:
  - serv_addr.sin_port=htons(portno);
    - convert the port from the host byte ordering to the network
- To extract port info from client:
  - ntohs(cli_addr.sin_port);
    - converts from network short to host short

Finding hostname

```
#include <netdb.h>

struct hostent *clientInfo;

....
....

clientInfo = gethostbyaddr(&(cli_addr.sin_addr),sizeof(cli_addr.sin_addr),AF_INET);
```

clientInfo.h_name now contains the hostname.

General Notes

- Use http version 1.0!
- Read returns number of bytes read, 0 indicates EOF
- Lines to be sent/received end in CRLF “\r\n”
- Blank line at end of header sent by both client and server important (“\r\n\r\n”)!
- Simple error handling ok
- Server should close connection when done
  - close(newsockfd);
- Simple is often best!