Get matched with a DCS alum who wants to help you navigate your path through school and into your career.

NOTE: This program is open to undergraduate students in all years, including those in PEY.

REGISTRATION DEADLINE: Monday, November 17, 2014

www.cs.toronto.edu/dcs/mentorship
Reading Long Weekend

Will the help centre still be open?
Yes, It will be open the usual 4-6 on both days.

Will you still hold office hours?
Undecided. I’ll post on Piazza if I’m not holding them or if they are adjusted.
Where were we?

- created sockets
  - server listens on one socket then once a client connects, transfers that client to another socket
- reading/writing is MOSTLY like in files except:
  - can’t assume that you’ll get a whole line or entire string at once
  - explicitly send and receive everything

“write this many bytes starting from this address”
int soc;
struct hostent *hp;
struct sockaddr_in peer;

peer.sin_family = AF_INET;
peer.sin_port = htons(PORT);
/* fill in peer address */
hp = gethostbyname(argv[1]);
peer.sin_addr = *((struct in_addr *)hp->h_addr);
/* create socket */
soc = socket(PF_INET, SOCK_STREAM, 0);
/* request connection to server */
if (connect(soc, (struct sockaddr *)&peer, sizeof(peer))
    == -1) {
    perror("client:connect"); close(soc); exit(1);
}
write(soc, "Hello Internet\n", 16);
read(soc, buf, sizeof(buf));
printf("SERVER SAID: %s\n", buf);
close(soc);
Byte order

• Big-endian
  $91,329 = \begin{array}{cccc}
  A & A+1 & A+2 & A+3 \\
  00 & 01 & 64 & C1
\end{array}$

• Little-endian
  $91,329 = \begin{array}{cccc}
  A & A+1 & A+2 & A+3 \\
  00 & 01 & 64 & C1
\end{array}$

• Intel is little-endian, and Sparc is big-endian
Network byte order

• To communicate between machines with unknown or different “endian-ness” we convert numbers to network byte order (big-endian) before we send them.

• There are functions provided to do this:
  - unsigned long htonl(unsigned long)
  - unsigned short htons(unsigned short)
  - unsigned long ntohl(unsigned long)
  - unsigned short ntohs(unsigned short)
Peer Wise

• With a partner draft a peer wise question about something that you learned since the midterm.
I/O Multiplexing

Kerrisk 63.2
When reading from multiple sources, blocking on one of the sources could be bad. An example of denial of service.

One solution: one process for every client. What are the pros and cons of this solution?
Another way to look at the problem

Server
while(1)
    accept a new connection
    for each existing connection
    read
    write

Which of the system calls might block indefinitely?
read and accept
So what happens if there is only one connection?
int select(int maxfdp1,
    fd_set *readset,
    fd_set *writeset,
    fd_set *exceptset,
    const struct timeval *timeout);

A call to select returns when one of the file descriptors in one of the sets is ready for I/O.
If timeout is not NULL, then select returns when a descriptor is ready or timeout time has passed.
If timeout is 0, select returns immediately after checking descriptors.
Readiness

Ready to read when

there is data in the receive buffer to be read
end-of-file state on file descriptor
the socket is a listening socket and there is a connection pending
a socket error is pending

Ready to write when

there is space available in the write buffer
a socket error is pending

Exception condition pending when

TCP out-of-band data

We are typically interested in when bytes are available to be read,
but sometimes we use select on write or exception sets
select timeout

• The timeout specifies how long we're willing to wait for a fd to become ready

```c
struct timeval {
    long  tv_sec;     /* seconds */
    long  tv_usec;    /* microseconds */
};
```

– If timeout is NULL, wait forever (or until we catch a signal)
– If timeout is zero, test and return immediately
– Otherwise wait up to specified timeout

• select returns when a fd ready or we timeout
Descriptor sets

Typically implemented as an array of integers where each bit corresponds to a descriptor (except in Windows).

Implementation is hidden in the `fd_set` data type.

`FD_SETSIZE` is the number of descriptors in the data type.

`maxfdp1` specifies the number of descriptors to test.

Macros:

```c
void FD_ZERO(fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_CLR(int fd, fd_set *fdset);
int  FD_ISSET(int fd, fd_set *fdset);
```
Descriptor sets

<table>
<thead>
<tr>
<th>client1</th>
<th>server</th>
<th>client2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 4 6</td>
<td></td>
</tr>
<tr>
<td>fd0</td>
<td></td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>fd1</td>
<td></td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>fd2</td>
<td></td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>fd3</td>
<td></td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>fd4</td>
<td></td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>fd5</td>
<td></td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>fd6</td>
<td></td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

maxfd + 1 = 7

After select:

<table>
<thead>
<tr>
<th>allset</th>
<th>0 0 0 1 1 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>rset</td>
<td>0 0 0 1 0 0 0</td>
</tr>
</tbody>
</table>

maxfd + 1 = 7
select example

fd_set rfds;
struct timeval tv;
int retval;

FD_ZERO(&rfds); /* Watch stdin (fd 0) for input */
FD_SET(STDIN_FILENO, &rfds);
tv.tv_sec = 5;  /* Wait up to five seconds. */
tv.tv_usec = 0;
retval = select(1, &rfds, NULL, NULL, &tv);
if (retval == -1)
    perror("select()");
else if (retval > 0)
    printf("Data is available now.\n");
    /* FD_ISSET(0, &rfds) will be true, can use read() */
else
    printf("No data within five seconds.\n");