### Lecture 8
Error Handling, Servers
January 31, 2005

---

#### One More Thing About Raw Sockets
- Typically stay away (bug-ridden, unportable, limited in use)!
- Who uses them? traceroute, ping, arp, Internet security tools.
- You can implement your own user-level protocols.
- Fill in the headers (TCP, UDP) yourself and make handshake protocol yourself.
- Datagram protocol: no use of listen/accept/...

---

#### Error Handling
- In general, systems calls return a negative number to indicate an error:
  - we often want to find out what error
  - servers generally add this information to a log
  - clients generally provide some information to the user

---

#### extern int errno;
- Whenever an error occurs, system calls set the value of the global variable errno.
  - you can check errno for specific errors
  - you can use support functions to print out or log an ASCII text error message

---

#### errno
- errno is valid only after a system call has returned an error.
  - system calls don’t clear errno on success
  - if you make another system call you may lose the previous value of errno
  - printf makes a call to write!

---

#### Error Codes
#include <errno.h>
- Error codes are defined in errno.h

<table>
<thead>
<tr>
<th>EAGAIN</th>
<th>EBADF</th>
<th>EACCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBUSY</td>
<td>EINTR</td>
<td>EINVAL</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Support Routines
In stdio.h:
void perror(const char *string);

In string.h:
char *strerror(int errno);

Strategies
- Include code to check for errors after every system call.
- Develop "wrapper functions" that do the checking for you.

Wrapper
int Socket(int f, int t, int p) {
    int n;
    if ( (n=socket(f,t,p)) < 0 )) {
        perror("Fatal Error");
        exit(1);
    }
    return(n);
}

What is Fatal?
- How do you know what should be a fatal error (program exits)?
  - common sense.
  - if the program can continue – it should.
  - example – if a server can’t create a socket, or can’t bind to it's port - there is no sense in continuing...

Wrappers
- Wrappers like those used in the text can make code much more readable.
- Sometimes system calls are "interrupted" (EINTR) – this is not always a fatal error!

Last Advice
- Make sure you check all system calls for errors!
  - not checking can lead to security problems!
  - not checking can lead to bad grades on assignments!
Now Back to Servers...

- Today we’ll discuss different server strategies...

Quiz

- Match these ports:
  - 80: ?
  - 21: ?
  - 23: ?
  - 22: ?
  - 79: ?
  - 42: ?
  - 13: ?
  - 7: ?
- Hint: echo, finger, daytime, ssh, telnet, ftp, http, nameserver

Quiz

- Match these ports:
  - 80: http
  - 21: ftp
  - 23: telnet
  - 22: ssh
  - 79: finger
  - 42: nameserver
  - 13: daytime
  - 7: echo
  - /etc/services

Server Algorithm

- Create a socket.
- Bind the socket to well-known port.
- Declare ‘listening’ socket.
- Infinite loop: accept requests and handle them.
- Problem?

Iterative vs. Concurrent

- Iterative servers: process one request at a time.
- Concurrent server: process multiple requests simultaneously.
- Concurrent: better use of resources (service others while waiting) and incoming requests can start being processed immediately after reception.

Web Server Statistics

Source: ACS/Utah
Server Structures
- Depending on server type, a server can receive between zero and many thousand requests per second.
- Possible server structures:
  - iterative
  - one child per client/request
  - pre-fork
  - select loop
  - ...

Connection-Oriented/Less
- TCP/IP: TCP (connection-oriented), UDP (connection-less) servers.
- TCP:
  - point-to-point communication: 2 endpoints
  - reliable connection establishment (tests network connectivity)
  - reliable delivery: same order, no loss, no duplication
  - flow-controlled transfer: fast-slow computers
  - full-duplex transfer: simultaneous in either direction
  - stream paradigm: byte streams, no message boundaries.

Connectionless
- UDP:
  - many-to-many connection: multiple senders/receivers, multicast, broadcast
  - unreliable service: can be lost, duplicated, out of order, no re-transmission, no signalling
  - no flow control: if too fast, packets are discarded
  - message paradigm: sender specifies data size, UDP places data in single outgoing message

Connection-Oriented
- Ease of programming: TCP takes care of most potential problems.
- Separate socket for each connection (resource utilization, 3WHS).
- Crashing client: server keeps connection up, resources cannot be reclaimed.

Basic Server Types
- Iterative connectionless.
- Iterative connection-oriented.
- Concurrent connectionless.
- Concurrent connection-oriented.