C Socket Programming

CS587X Lecture 3
Fall, 2002

C Socket Interface

- Generalized communication interface
- Available on most UNIX and UNIX-like systems
  - Similar interface, Winsock, available on Windows
- Protocol families include
  - Internet (PF_INET)
  - Unix (PF_UNIX)
  - ISO (PF_ISO)
- Socket types of interest
  - SOCK_STREAM
  - SOCK_DGRAM

Socket Types

- SOCK_DGRAM
  - Maps to UDP in the PF_INET family
  - Use send, sendto, recv, recvfrom calls
- SOCK_STREAM
  - Maps to TCP in the PF_INET family
  - Use send, write, recv, read calls

Using Sockets

- Open a socket of the desired type and family
  - s=socket(PF_INET, SOCK_DGRAM, 0);
- Optionally bind the socket's local address
  - bind(s, local_sockaddr, addrlen);
  - By default, the O/S will assign an ephemeral local port
  - Why use bind()?
    - To specify the local port for a server process
- listen(s, backlog) & accept(s)
  - If a streaming server socket

Using Sockets (2)

- Connect the socket's remote address
  - connect(s, remote_sockaddr, addrlen);
  - Not necessary with UDP
    - Can use sendto() and specify the remote's address along with the message
  - Why connect()?
    - For TCP, establish the virtual connection
    - For UDP, specify the default recipient

Using Sockets (3)

- read()/write()/recv()/send() using the socket descriptor
  - A read() returning 0 bytes indicates EOF
- shutdown() the stream socket when done sending or receiving
  - e.g., shutdown(sock, SHUT_WR)
  - For example, can be used to inform the receiver that a complete request has been sent
- close() the socket descriptor

Streaming Server Example

```
int sock, i, s_in_len;
struct sockaddr_in s_in, new_s_in;
char buf[1024];

memset(&s_in, 0, sizeof(s_in));
s_in->sin_port = htons(8080); /* Server port */
s_in->sin_addr.s_addr = INADDR_ANY; /* Any IP address */
sock = socket(PF_INET, SOCK_STREAM, 0);
s_in_len = sizeof(struct sockaddr_in);
bind(sock, (struct sockaddr *)s_in, s_in_len);
listen(sock, 1);
new_sock = accept(sock, &new_s_in, sizeof(new_s_in));
while ((i = read(new_sock, buf, sizeof(buf))) > 0)
  write(1, buf, i);
shutdown(new_sock, SHUT_RW);
close(new_sock);
close(sock);
```

Streaming Client Example

```
int sock, s_in_len, i;
struct sockaddr_in *s_in;
char buf[1024];

s_in = malloc(sizeof(struct sockaddr_in));
memset(s_in, 0, sizeof(*s_in));
s_in->sin_port = htons(80); /* HTTP */
inet_aton("129.186.3.78", /* www.cs.iastate.edu */
  &s_in->sin_addr.s_addr);
sock = socket(PF_INET, SOCK_STREAM, 0);
s_in_len = sizeof(struct sockaddr_in);
connect(sock, (struct sockaddr *)s_in, s_in_len);
write(sock, "GET / HTTP/1.0\r\n\r\n", 18);
shutdown(sock, SOCK_WR);
while ((i = read(sock, buf, sizeof(buf))) > 0)
  write(1, buf, i);
close(sock);
```

Datagram Server Example

```
int sock, i, s_in_len, from_len;
struct sockaddr_in s_in, from;
char buf[1024];

memset(&s_in, 0, sizeof(s_in));
s_in->sin_port = htons(8080); /* Server port */
s_in->sin_addr.s_addr = INADDR_ANY; /* Any IP address */
sock = socket(PF_INET, SOCK_DGRAM, 0);
s_in_len = sizeof(struct sockaddr_in);
bind(sock, (struct sockaddr *)s_in, s_in_len);
from_len = sizeof(from);
while ((i = recvfrom(new_sock, buf, sizeof(buf), 0, &from,
   &from_len)) > 0) {
  printf("Message from %s\n",
   inet_ntoa(from.sin_addr.s_addr));
  fflush(stdout);
  write(1, buf, i);
  from_len = sizeof(from);
}
close(sock);
```

Datagram Client Example

```
int sock, s_in_len, i;
struct sockaddr_in *s_in;
char buf[1024];

s_in = malloc(sizeof(struct sockaddr_in));
memset(s_in, 0, sizeof(*s_in));
s_in->sin_port = htons(7); /* Echo */
inet_aton("129.186.3.78", /* www.cs.iastate.edu */
  &s_in->sin_addr.s_addr);
sock = socket(PF_INET, SOCK_STREAM, 0);
s_in_len = sizeof(struct sockaddr_in);
connect(sock, (struct sockaddr *)s_in, s_in_len);
send(sock, "My Message\r\n", 12);
i = recv(sock, buf, sizeof(buf));
write(1, buf, i);
close(sock);
```

Waiting for Input

- What if input never comes?
- How to handle reading & writing from/to multiple sockets?
- select(max_fd, readfds, writefds, exceptfds, timeout)
  - max_fd: highest-numbered file descriptor in any of the bitmaps, plus 1
  - readfds, writefds, exceptfds: bitmap of descriptors to watch for available input, output, and exceptions
  - timeout: length of time, in seconds and microseconds, to wait

select() & Timeouts

```
struct timeval tv;
fd_set fds;
...
tv.tv_sec = 5;
tv.tv_usec = 0;
FD_ZERO(&fds);
FD_SET(&fds, sock);
result = select(sock + 1, &fds, NULL, NULL,
   &tv);
if (result == 1) { read_from_sock(sock); }
if (result == 0) { timed_out(sock); }
```

select() & Multiple Sockets

```
struct timeval tv;
fd_set fds;
...
tv.tv_sec = 5; tv.tv_usec = 0;
FD_ZERO(&fds);
FD_SET(&fds, sock1);
FD_SET(&fds, sock2);
result = select(MAX(sock1, sock2) + 1, &fds,
   NULL, NULL, &tv);
if (result > 1) {
  if (FD_ISSET(&fds, sock1)
    read_from_sock(sock1);
  if (FD_ISSET(&fds, sock2)
    read_from_sock(sock2);
}
if (result == 0) { timed_out(sock); }
```

Summary

- Sockets
  - PF_INET
  - SOCK_STREAM & SOCK_DGRAM
  - Reading & writing with stream sockets
  - Reading & writing with datagram sockets
  - Using select() for timeouts and multiple sockets
- Useful for comparison to Java's socket interface