Socket Programming

Christian Grothoff
christian@grothoff.org
http://grothoff.org/christian/

“Berkeley boys to the rescue” – Doug Comer
Overview

- UDP client and server
- TCP client and server
- UNIX client and server
- select and epoll
- DNS lookups
- Diagnostic tools
Keeping it short...

• No declarations of variables unrelated to networking
• No error handling code
• Minor details ignored

⇒ Read man-pages to easily fill the gaps
IPv4 UDP Sender Example

```
struct sockaddr_in addr;
int s = socket (PF_INET, SOCK_DGRAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin_family = AF_INET;
addr.sin_port = htons (5002);
addr.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
sendto (s, "Hello World", 12, 0,
       (const struct sockaddr*) &addr, sizeof (addr));
close (s);
```
IPv4 UDP Receiver Example

```c
struct sockaddr_in addr;
char buf[1024];
char sa[128];
socklen_t salen = sizeof(sa);
int s = socket (PF_INET, SOCK_DGRAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin_family = AF_INET;
addr.sin_port = htons (5002);
bind (s, (const struct sockaddr*) &addr, sizeof (addr));
recvfrom (s, buf, sizeof(buf), 0,
        (struct sockaddr*) &sa, &salen);
close (s);
```
FreeBSD

```c
memset (&addr, 0, sizeof (addr));
#if HAVE_SOCKADDR_IN_SIN_LEN
    addr.sin_len = sizeof (addr);
#endif
    addr.sin_family = AF_INET;
...
Example: minimal TCP client

Functionality:

- Connect to server on port 5002
- Transmit file to server
System Calls for TCP client

1. socket
2. connect
3. (recv|send)*
4. [shutdown]
5. close
IPv4 TCP Client Example

```c
struct sockaddr_in addr;
int s = socket (PF_INET, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sin_family = AF_INET;
addr.sin_port = htons (5002);
addr.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
connect (s, (const struct sockaddr*) &addr, sizeof (addr));
process(s);
close (s);
```
Client Example: processing

static void process (int s) {
    char buf[4092];
    int f = open (FILENAME, O_RDONLY);
    while ( (-1 != (n = read (f, buf, sizeof (buf)))) &&
            (n != 0)) {
        pos = 0;
        while (pos < n) {
            ssize_t got = write (s, &buf[pos], n - pos);
            if (got <= 0) goto END;
            pos += got;
        }
    }
    close (f);
}

END:
Example: minimal TCP server

Functionality:

- Listen to port 5002
- Write incoming TCP stream to disk
- Support multiple clients in parallel using pthreads
System Calls for TCP server

1. socket
2. bind
3. listen
4. “forever”:
   (a) accept
   (b) (recv|send)*
   (c) [shutdown]
   (d) close
5. close
Server Example: processing

static void * process (struct T * t) {
    ssize_t n;
    char buf[4092];

    int f = creat (filename, S_IRUSR | S_IWUSR);
    while ( (-1 != (n=read (t->a, buf, sizeof (buf)))) &&
            (n != 0) )
        write (f, buf, n);
    close (f);
    close (t->a);
    return NULL;
}
Server Example: struct T

struct T {
    int a;
};
Server Example: accepting

```c
struct sockaddr addr;
int s = socket (PF_INET, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
struct sockaddr_in * ia = (struct sockaddr_in*) &addr;
ia->sin_family = AF_INET; ia->sin_port = htons (5002);
bind (s, &addr, sizeof (struct sockaddr_in));
listen (s, 5);
while (1) {
    memset (&addr, 0, sizeof (addr));
    socklen_t alen = sizeof (struct sockaddr);
t->a = accept (s, &addr, &alen);
pthread_create (&pt, NULL, &process, t);
}
Threads?

- Need to “clean up” handle pt (use struct T)
- Can cause dead-locks, data races
- Do not exist on all platforms
- Use at least one page of memory per thread, often more
- How scalable is your thread-scheduler?
select

- Do everything in one “thread”, no parallel execution needed
- Event-based $\Rightarrow$ tricky API, but fewer tricky bugs!
- Exists on pretty much all network-capable platforms
- Has some issues with UNIX signals, but mostly “safe”
- Scales with $O(n)$
select API

- FD_ZERO(fd_set *set)
- FD_SET(int fd, fd_set *set)
- FD_ISSET(int fd, fd_set *set)
- int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)

Homework: Read select_tut man-page and try it!
Example (1/3)

```c
int pi[2];
pipe(pi);
if (fork() == 0) {
    close(pi[0]);
    close(0); close(1); close(2);
    while (1) { write(pi[1], "Hello\n", 6); sleep(5); }
} else {
    close(pi[1]);
    while (1) { merge(pi[0], 0, 1); }
}
```
Example (2/3)

```c
#define MAX(a,b) ((a) > (b) ? (a) : (b))

void merge(int in1, int in2, int out) {
    fd_set rs, ws;
    FD_ZERO(&rs); FD_ZERO(&ws);
    FD_SET(in1, &rs); FD_SET(in2, &rs);
    select(1 + MAX(in1, in2), &rs, &ws, NULL, NULL);
    if (FD_ISSET(in1, &rs)) copy(in1, out);
    if (FD_ISSET(in2, &rs)) copy(in2, out);
}
```
Example (3/3)

```c
void copy(int in, int out) {
    size_t num;
    char buf[1024];

    num = read(in, buf, sizeof(buf));
    write(out, buf, num);
}
```
epoll

- Select scales with $O(n)$
- Can (theoretically) do the same with $O(1)$
- Linux does this using epoll
- Key different to select: you must have drained the buffers before epoll will trigger again!
epoll API

• int epoll_create(int size)

• int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)

• int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout)

Homework: Read epoll man-page and try it!
Other possibilities

- Forking
- Pre-Forking
- Multi-threaded with select or epoll
- kqueue (FreeBSD, NetBSD, OS X)
- Asynchronous IO (W32, z/OS), Signals (Linux)

connect revisited

- select works fine for read and write
- connect also blocks!

⇒ Need non-blocking connect!
Non-blocking `connect`

```c
struct sockaddr_in addr;
int s = socket (PF_INET, SOCK_STREAM, 0);
int ret;
int flags = fcntl (s, F_GETFL);
flags |= O_NONBLOCK;
fcntl (s, F_SETFL, flags);
ret = connect (s, (const struct sockaddr*) &addr, sizeof (addr));
if ( (ret == -1) && (errno == EAGAIN) )
    { /* wait in "select" for "write" */ }
```
The UNIX Address Family

• BSD Sockets used for *local* communication (IPC)

• No TCP, no network $\Rightarrow$ much faster

• Local only — can use UNIX access permissions!

• Special functions allow passing of file-descriptors and credentials between processes
UNIX Client Example

```c
struct sockaddr_un addr;
int s = socket (PF_UNIX, SOCK_STREAM, 0);
memset (&addr, 0, sizeof (addr));
addr.sun_family = AF_UNIX;
strcpy (addr.sun_path, "/tmp/my-unix-socket");
unlink (addr.sun_path);
connect (s, (const struct sockaddr*) &addr,
        strlen(addr.sun_path) + sizeof(addr.sun_family));
process(s);
close (s);```

Credentials Passing on Linux

As the UNIX server, you can get the client’s credentials using:

```c
struct ucred credentials;
int ucred_length = sizeof(struct ucred);
getsockopt(s, SOL_SOCKET, SO_PEERCRED,
    &credentials, &ucred_length))
// use ucred.pid, ucred.uid, ucred.gid
```
File Descriptor Passing on Linux

See

http://www.wsinnovations.com/softeng/articles/uds.html

for example code.
UNIX Datagram Family Example

UNIX Sockets can also be used for UDP-like communication:

```c
struct sockaddr_un addr;
int s = socket (PF_UNIX, SOCK_DGRAM, 0);
// ... (see UDP Sockets)
```
Java

- `java.net.Socket` for client sockets: bind, connect
- `java.net.ServerSocket` for client sockets: bind, accept
- `java.nio.channels.SocketChannel` for IO: read, write
- `java.nio.channels.Selector` for IO: select
IP-based access control

- Bind socket to limited IP addresses
- Check that connection is from trusted network
- Check that IP matches certain DNS names
Server Example: loopback only

```c
struct sockaddr_in ia;
int s = socket (PF_INET, SOCK_STREAM, 0);
memset (&ia, 0, sizeof (ia));
ia.sin_family = AF_INET;
ia.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
ia.sin_port = htons (5002);
struct sockaddr * addr = (struct sockaddr *)&ia;
bind (s, addr, sizeof (struct sockaddr_in));
// ...
```
Parsing addresses

```c
int parse(const char * in,
           struct in_addr * out) {
    int ret = inet_pton(AF_INET, in, out);
    if (ret < 0)
        fprintf(stderr, "AF_INET not supported!\n");
    else if (ret == 0)
        fprintf(stderr, "Syntax error!\n");
    else
        return 0;
    return -1;
}
```
Network check

```c
int test_in_network (const struct in_addr * network, 
                      const struct in_addr * mask, 
                      const struct in_addr * addr) {
    return ( (addr->s_addr & mask.s_addr) 
            == network.s_addr & mask.s_addr)
}
```
```c
int
resolve_old (const char * hostname,
            struct in_addr * addr) {
    struct hostent * he;
    struct sockaddr_in *addr;
    he = gethostbyname(hostname);
    assert (he->h_addrtype == AF_INET);
    assert (hp->h_length == sizeof (struct in_addr));
    memcpy (addr, hp->h_addr_list[0], hp->h_length);
    return OK;
}
```
gethostbyname issues

• Synchronous
• IPv4 only

⇒ gethostbyname2
gethostbyname issues

- Synchronous
- IPv4 only

⇒ gethostbyname2

- Not reentrant

⇒ both are obsolete!
IPv4 DNS request with `getaddrinfo`

```c
int resolve (const char * hostname,
             struct sockaddr_in * addr) {
    struct addrinfo hints;
    struct addrinfo *result;
    memset (&hints, 0, sizeof (struct addrinfo));
    hints.ai_family = AF_INET;
    getaddrinfo (hostname, NULL, &hints, &result);
    assert(sizeof(struct sockaddr_in) == result->ai_addrlen);
    memcpy (addr, result->ai_addr, result->ai_addrlen);
    freeaddrinfo (result);
    return OK;
}
```
Reverse Lookup: `getnameinfo`

```c
char *
reverse_resolve (const struct sockaddr_in *ip) {
    char hostname[256];
    if (0 != getnameinfo ((const struct sockaddr*) ip,
                           sizeof (struct sockaddr_in),
                           hostname, sizeof(hostname),
                           NULL, 0, 0))
        return NULL;
    return strdup (hostname);
}
```
Data Transmission

All well-designed protocols transmit data in **network byte order**:

```c
uint32_t data;

data = htonl (42);
do_transmit ((const char*) &data, sizeof (data));
```

TUM
Receiving Data

When receiving data, it must be converted back:

```c
char buf[2];
uint16_t *nbo_data;
uint16_t sdata;

do_receive (buf, sizeof (buf));
nbo_data = (uint16_t *) buf;
sdata = ntohs (*nbo_data);
```
Fun with TCP

TCP windows can cause partial reads:

```c
void do_receive (char *buf, size_t num) {
    size_t off;
    ssize_t ret;
    off = 0;
    while (off < num) {
        ret = read (sock, &buf[off], num - off);
        if (ret == -1) abort ();
        off += ret;
    }
}
```
Fun with TCP

TCP windows can cause partial writes:

```c
void do_transmit (const char *buf, size_t num) {
    size_t off;
    ssize_t ret;
    off = 0;
    while (off < num) {
        ret = write (sock, &buf[off], num - off);
        if (ret == -1) abort (
        off += ret;
    }
}
```
Diagnostics

On a GNU/Linux system, run:

- $ netstat -nl
- $ netstat -nt
- valgrind --track-fds=yes

“Happy hacking.” – RMS
Questions

?
Copyright

Copyright (C) 2008, 2009 Christian Grothoff

Verbatim copying and distribution of this entire article is permitted in any medium, provided this notice is preserved.