C/C++ Lectures

Daemons and Networking

Karel Kubat
e-tunity

karel@kubat.nl
http://www.kubat.nl

http://lectures.kubat.nl/2007-c/

Contents

- Daemons
  - Forking daemons
  - Multi-threaded daemons
  - Mixed forking / threaded
  - How should I daemonize?
  - How should I behave (signals)?

- Networking
  - OSI Model, TCP and UDP
  - Client-side Socket Handling
  - Socket IO and Timeouts
  - Server-side Socket Handling

- Loose Ends
  - Error handling and Messaging
  - DOS Protection
  - IP-Address based Access Control Lists

Daemons: Introduction

What is a daemon?

- Started once (at boot time)
- No command line or GUI
- No operator intervention
- Wait for an 'event', then handle it
- Event handling in parallel: multi-processing
- Often networked
- Forking or multi-threaded

Daemons: Forking vs. Threaded

Forking

- New program copy
- New resources
- Both copies run
- Copy may crash
- “Private” data
  - Easy modification
  - Difficult IPC

Threaded

- Same program copy
- New execution point
- One program runs
- Crashes are deadly
- “Shared” data
  - Careful modification
  - Easy PC

Daemons: Forking

Code sample of forking

```
pid = fork();
if (pid == -1)
      error("I can't fork!\n");

if (pid > 0) {
      // Parent branch: pid was started
      printf ("Event will be handled by pid %d\n", pid);
} else {
      // Child branch
      do_interesting_stuff();
      exit(0);
}
```

Daemons: Forking

Potential Problems

- Avoid “fire and forget”
  - What is the child's exit status?
- Avoid “zombies”
  - Non-tracked processes

```
pid_t child;
int status;

while (child = wait4 (-1, &status, WNOHANG, 0)) > 0) {
    // Pid 'child' exited, state saved in 'status'
}
```

Daemons: Threaded

POSIX Threads

- Portable Operating System Interface
- API to Unix variants (commands, library calls)
- POSIX Threads: about 1995

- pthread_create (&id, &attributes, startroutine, &data)
- pthread_cancel (id)
- pthread_exit (&retval) / return()
- pthread_join (other_id, &retvalptr)
- pthread_mutex_lock (&lck)
- pthread_mutex_unlock (&lck)

Daemons: Threaded

PThread's Startroutine and Data Access

- Startroutine
  `void *startroutine (void *thread_data)`
- Access to allocated `thread_data`: thread-safe
- Access to other data:
  `pthread_mutex_lock (&mutex);`
  `pthread_mutex_unlock (&mutex);`
- “Not Thread Safe”:
  - Static data
  - No locking provisions

Deamons: Mixed

Apache's mod_worker: forking *and* threaded!

StartServers	25
ThreadsPerChild	50
ServerLimit	100

- Initially 25 forks, with max 50 threads per fork
- Initially 1250 waiting threads
- Maximum 100 forks
- Maximum 5000 active threads

Daemons: How to daemonize

Fork to daemonize: parent vs. child branch

- Parent branch code: initialize, fork and exit
- Child branch code:
  - Free up mount points:

    chdir ("/");
  - Create new process group:

    setsid ();
  - Free file descriptors 0, 1, 2:

    close (0); close (1); close (2);

    Re-open file descriptors 0, 1, 2:

    open ("/dev/null", O_RDONLY);
    open ("/dev/null", O_WRONLY);
    open ("/dev/null", O_WRONLY);

Daemons: How should I behave

Signal handling

- void handler (int signum) { .... }
- signal (signum, handler);
- signal (signum, SIG_IGN);
- signal (signum, SIG_DFL);

Typically caught signals:

- SIGHUP (1): Re-read configuration and continue
- SIGINT, SIGQUIT, SIGTERM: Graceful stop
- SIGCHLD, SIGPIPE: Application-dependent

Typically not caught:

- SIGKILL (9), SIGBUS, SIGSEGV

Daemons: Signal Handler Protocol

Prototype

- void handler (int signum) { .... }

Use only reentrant functions

- no new, delete, malloc() and friends
- no stream IO, no file IO, no printf() and friends
- POSIX defined whitelist
- except when handler terminates program

Separate stack space

- no setjmp()/longjmp() and friends
- except when termination follows

Daemons: Signal Handler Wiring

Safe Implementation

```
int interrupted = 0;

void handler (int signum) {
    interrupted++;
}

/* Event loop */
signal (SIGINT, handler);
while (!interrupted) {
    /*
     * Do some useful stuff
     */
}
```

Daemons: Signal Handler Wiring

Topsafe Implementation

```
int interrupted = 0;

void handler (int signum) {
    interrupted++;
}

void (*prevhander)(int) = signal (SIGINT, handler);
while (!interrupted) {
    /*
     * Do some useful stuff
     */
}
signal (SIGINT, prevhandler);
```

C/C++ Lectures

Daemons and Networking

Karel Kubat
e-tunity

karel@kubat.nl
http://www.kubat.nl

http://lectures.kubat.nl/2007-c/

Contents

- Daemons
  - Forking daemons
  - Multi-threaded daemons
  - Mixed forking / threaded
  - How should I daemonize?
  - How should I behave (signals)?

- Networking
  - OSI Model, TCP and UDP
  - Client-side Socket Handling
  - Socket IO and Timeouts
  - Server-side Socket Handling

- Loose Ends
  - Error handling and Messaging
  - DOS Protection
  - IP-Address based Access Control Lists

Networking

OSI 7-Layer Model

Open Systems Interconnection basic reference

1. Physical layer:	cable, RS232 port, NIC
2. Data link: Ethernet:	PPP
3. Network layer:	ICMP, IP
4. Transport layer:	TCP, UDP
5. Session layer:	a connection, 'socket'
6. Presentation layer:	XDR, SSL
7. Application layer:	SMTP, POP, HTTP

Networking

OSI Layer 3-7 Analyzer: Ethereal

(Untitled) – Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
4	4.909198	192.168.2.2	194.50.163.139	TCP	54439 > http [SYN] Seq=0
5	4.919042	194.50.163.139	192.168.2.2	TCP	http > 54439 [SYN, ACK] S
6	4.919155	192.168.2.2	194.50.163.139	TCP	54439 > http [ACK] Seq=1
7	4.919775	192.168.2.2	194.50.163.139	HTTP	GET /2007-c/index03.html
8	4.931683	194.50.163.139	192.168.2.2	TCP	http > 54439 [ACK] Seq=1
9	4.933290	194.50.163.139	192.168.2.2	HTTP	HTTP/1.1 304 Not Modified
10	4.933363	192.168.2.2	194.50.163.139	TCP	54439 > http [ACK] Seq=7

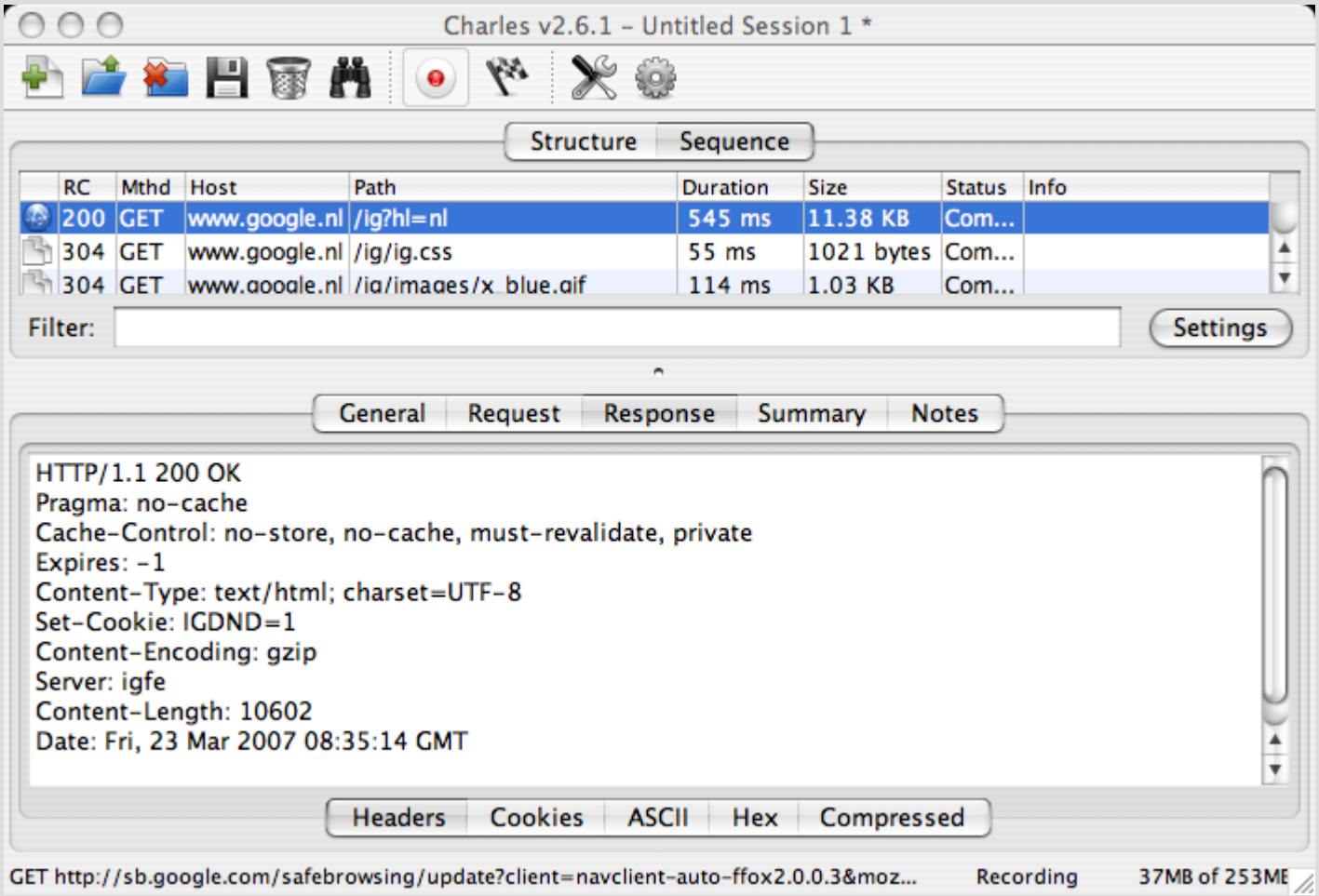
```
▷ Ethernet II, Src: 00:11:24:a3:b9:96, Dst: 00:30:bd:96:0a:00
▷ Internet Protocol, Src Addr: 192.168.2.2 (192.168.2.2), Dst Addr: 194.50.163.139 (194.50.163.139)
▽ Transmission Control Protocol, Src Port: 54439 (54439), Dst Port: http (80), Seq: 1, Ack: 1, Len: 730
    Source port: 54439 (54439)
    Destination port: http (80)
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 731    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
```

```
0020  a3 8b d4 a7 00 50 2d 00  60 f4 b9 42 74 a1 80 18   .....P-. `..Bt...
0030  ff ff 98 ef 00 00 01 01  08 0a 00 a7 2a 40 d8 33   ........ ....*@.3
0040  3d fe 47 45 54 20 2f 32  30 30 37 2d 63 2f 69 6e   =.GET /2 007-c/in
0050  64 65 78 30 33 2e 68 74  6d 6c 20 48 54 54 50 2f   dex03.ht ml HTTP/
```

Destination Port (tcp.dst) | P: 28 D: 28 M: 0

Networking

OSI Layer 6-7 HTTP(S) Analyzer: Charles



Networking

Unix command line

- Netstat: network connections
- Ping: icmp connectivity

```
ssh
[root@leonardo] root > netstat -nt |head -5
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  192.168.2.2.54910      10.120.34.111.8080     SYN_SENT
tcp4       0      0  192.168.2.2.54908      194.50.163.139.80      ESTABLISHED
tcp4       0      0  127.0.0.1.8888         127.0.0.1.54892        ESTABLISHED
[root@leonardo] root > ping -c3 www.kubat.nl
PING www.kubat.nl (194.50.163.138): 56 data bytes
64 bytes from 194.50.163.138: icmp_seq=0 ttl=57 time=8.847 ms
64 bytes from 194.50.163.138: icmp_seq=1 ttl=57 time=10.410 ms
64 bytes from 194.50.163.138: icmp_seq=2 ttl=57 time=9.836 ms

--- www.kubat.nl ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 8.847/9.698/10.410/0.646 ms
[root@leonardo] root >
```

Networking: TCP vs. UDP

OSI Layer 4

- TCP: Transmission Control Protocol
  - Continuous flow, re-transmitted if necessary
  - Order of transmission == order of reception
  - Slower than UDP
  - Suitable for guaranteed data flows

- UDP: User Datagram Protocol
  - “Fire and forget” - not guaranteed
  - Possibly out of order
  - Faster than TCP
  - Suitable for one-shot messages (syslog, P2P, streaming)

Networking

OSI 7-Layer Model

Open Systems Interconnection basic reference

1. Physical layer
2. Data link
3. Network layer
4. Transport layer
5. Session layer (You are HERE, in tcp mode)
6. Presentation layer
7. Application layer

Networking: Client-side

Steps to take

- Establish a network socket
- Look up the server to connect to
- Prepare binding of socket and host info
- Connect the socket to the server
- Service the connection
- Close the connection

Networking: Client-side

Steps to take:

- Establish a network socket
- Look up the server to connect to
- Prepare binding of socket and host info
- Connect the socket to the server
- Service the connection
- Close the connection

Establish a network socket:

```
int sock;

if ( (sock = socket (PF_INET, SOCK_STREAM, 0)) < 0 )
      error ("Failed to create a network socket\n");
```

Networking: Client-side

Steps to take:

- Establish a network socket
- Look up the server to connect to
- Prepare binding of socket and host info
- Connect the socket to the server
- Service the connection
- Close the connection

Look up the server to connect to:

```
struct hostent *hent;
if (! (hent = gethostbyname ("my.server.org")) )
    error ("Failed to resolve hostname\n");
```

Networking: Client-side

Steps to take:

- Establish a network socket
- Look up the server to connect to
- Prepare binding of socket and host info
- Connect the socket to the server
- Service the connection
- Close the connection

Prepare binding between socket and host info:

```
struct sockaddr_in sname;

sname.sin_family = AF_INET;
sname.sin_port = htons (80);          // E.g. TCP port 80 / http
sname.sin_addr = hent->h_addr;
```

Networking: Client-side

Steps to take:

- Establish a network socket
- Look up the server to connect to
- Prepare binding of socket and host info
- Connect the socket to the server
- Service the connection
- Close the connection

Connect the socket to the server:

```
if (connect (sock, &sname, sizeof(sname)))
    error ("Failed to connect to server\n");
```

Service and close:

```
read() / write()
close (sock);
```

Networking: Client-side

connect() may fail upon refused connection

Check return value, must be zero

connect() may hang when packets are dropped

To catch time outs / **forking** daemon:

- connect () is interrupted upon signal
- signal(SIGALRM, timeout_handler)
- flag variable timed_out

To catch time outs / **threaded** daemon:

- put socket in nonblocking mode
- connect(), errno == EINPROGRESS
- select() for writing
- next read()/write() may give errno ECONNREFUSED

Networking: Client-side

```
int timed_out = 0;

void timeout_handler (int sig) {
	timed_out++;
}

signal (SIGALRM, timeout_handler);
alarm (10);
if (connect (sock, sname, sizeof(sname)) {
	if (timed_out)
		error (“Connection timed out\n”);
	else
		error (“Connection failed\n”);
}
signal (SIGALRM, SIG_DFL);
alarm (0);
```

Networking: Socket I/O

Sockets are most basic file descriptors

- Chunk-based, not 0-terminated, unbuffered
- Reading or writing a fixed size may require several read/write's

```
int socket_write (int fd, char const *buf, int len) {
    int totwritten = 0, nwritten = 0;
    while (totwritten < len) {
        nwritten = write (fd, buf + totwritten, len - totwritten);
        if (nwritten < 1)
            break;
        totwritten += nwritten;
    }
    return (totwritten == 0 && nwritten < 0 ?
            nwritten : totwritten);
}
```

Networking: Socket I/O

System calls `read()` / `write()` are blocking

- Handling time outs, e.g. while reading:
  - Preparation:
    - Timeout specifier (sec / microsec)
    - File descriptor sets: readset, exceptionset
  - Peeking before the system call: `select()`
  - System call itself: `read()` / `write()`

Networking: Socket I/O Timeouts

Handling time outs while reading:

- Preparation: timeout specifier, readset, exceptset
- Peeking before the system call: `select()`
- System call itself: `read()` / `write()`

```
struct timeval tv;
fd_set readset, exceptset;

// 10 sec timeout
tv.tv_sec = 10;                    tv.tv_usec = 0;

// file descriptors sets
FD_ZERO (&readset);                FD_SET (sock, &readset);
FD_ZERO (&exceptset);              FD_SET (sock, &exceptset);
```

Networking: Socket I/O Timeouts

Handling time outs while reading:

- Preparation: timeout specifier, readset, exceptset
- Peeking before the system call: select()
- System call itself: read() / write()

```
int res;

if ( (res = select (FD_SETSIZE,
                    &readset, 0, &exceptset,
                    &tv)) < 0 )
    error ("Select failed\n");
if (!res)
    error ("IO timed out\n");
if (FD_ISSET (sock, &exceptset))
    error ("Exception on socket %d\n", sock);
```

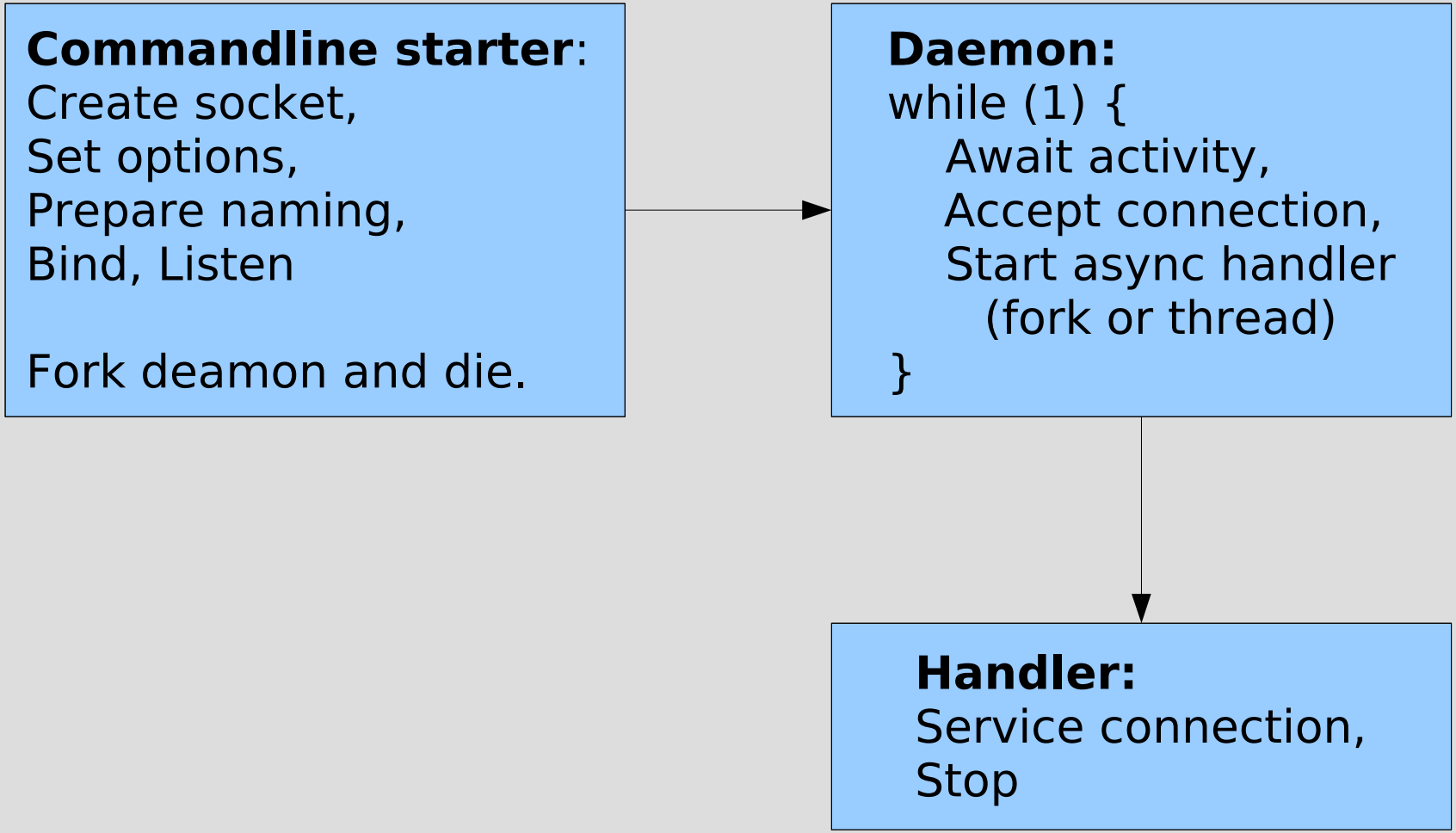
Networking: Server-side

Comparable to client-side socket handling,
few specific steps:

- Socket creation using `socket()`
- Setting socket options using `setsockopt()`
- Prepare naming to localhost or specific IP address
- Binding to address using `bind()`
- Inform kernel using `listen()`
- Daemonize
- Await activity using `select()`
- Accept a new TCP connection using `accept()`
- Serve client in separate fork or thread

Networking: Server-side

Steps:



Networking: Server-side

Server-side steps:

- Socket creation and options
- Naming, binding, listening
- Waiting for a client's connection
- Servicing a client

```
int sock, val = 1;

if ( (sock = socket (PF_INET, SOCK_STREAM, 0)) < 0 )
      error ("Failed to create socket\n");

if (setsockopt (sock, SOL_SOCKET, SO_REUSEADDR, &val,
                        sizeof(val)))
      error ("Failed to set socket options\n");
```

Networking: Server-side

Server-side steps:

- Socket creation and options
- Naming, binding, listening
- Waiting for a client's connection
- Servicing a client

```
struct sockaddr_in servername;

servername.sin_family = AF_INET;
servername.sin_port = htons (80);          // E.g. http port
servername.sin_addr.s_addr = htonl (INADDR_ANY);

if (bind (sock, &servername, sizeof(servername)) < 0)
      error ("Cannot bind socket\n");
if (listen (sock, 5) < 0)
      error ("Cannot listen to socket\n");
```

Networking: Server-side

Server-side steps:

- Socket creation and options
- Naming, binding, listening
- Waiting for a client's connection
- Servicing a client

```
fd_set readset, exceptset;

FD_ZERO (&readset);          FD_SET (sock, &readset);
FD_ZERO (&exceptset);        FD_SET (sock, &exceptset);
if (select (FD_SETSIZE, &readset, 0, &exceptset, 0) < 0)
      error ("Select failed\n");
if (FD_ISSET (serversock, &exceptset))
      error ("Exception on network socket\n");
```

Networking: Server-side

Server-side steps:

- Socket creation and options
- Naming, binding, listening
- Waiting for a client's connection
- Servicing a client

```
int clientsock, size;
struct sockaddr_in clientname;

if ( (clientsock = accept (serversock, &clientname,
                                    &size)) < 0)
      error (“Failed to accept client's connection\n”);
msg (“Got a connection from %s\n”,
        inet_ntoa (clientname.sin_addr));
// Next, serve the client, and then..
close (clientsock);
```

Contents

- Daemons
  - Forking daemons
  - Multi-threaded daemons
  - Mixed forking / threaded
  - How should I daemonize?
  - How should I behave (signals)?

- Networking
  - OSI Model, TCP and UDP
  - Client-side Socket Handling
  - Socket IO and Timeouts
  - Server-side Socket Handling

- Loose Ends
  - Error handling and Messaging
  - DOS Protection
  - IP-Address based Access Control Lists

Loose Ends: Messaging

Once daemonized:

- File descriptors 0, 1 and 2 are gone
- No more stdin/cin, stdout/cout, stderr/cerr
- Instead:
  - Errors: syslog (LOG_ALERT, ...)
  - Warnings: syslog (LOG_WARNING, ...)
  - Messages: syslog (LOG_NOTICE, ...)

- Report sufficient details:

  errno/strerror() are your friends

```
if ( (pid = fork()) == -1 )
    error("I can't fork: %s!\n", strerror(errno));
```

Loose Ends: DOS Protection

Keep count of concurrent TCP connections

```
while (nconnections > 100)
    sleep (1);
clientsock = accept (...);
if (nconnections > 120)
    close (clientsock);
```

- Forking daemon:
  - nconnections++ upon forking
  - nconnections-- when child terminates
- Threaded daemon:
  - nconnections++ upon thread start
  - nconnections-- when thread terminates

Loose Ends: IP-Based ACL's

- Most network connections are to localhost!
  Use iptables/ipchains whenever possible.

- Bind server to specific IP address, instead of:

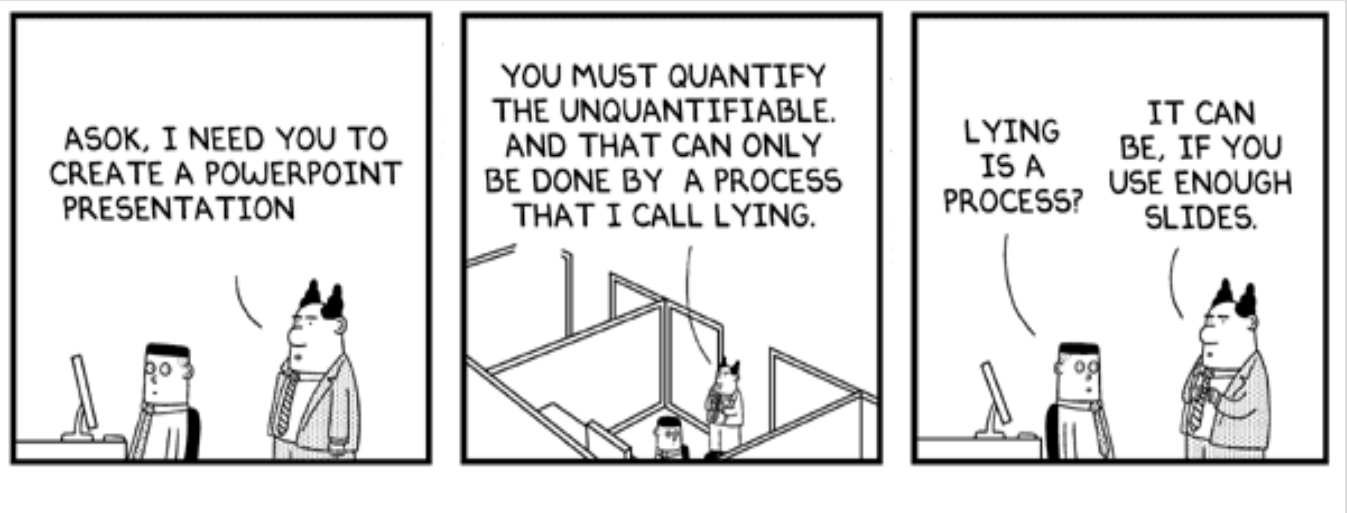
```
sockaddr.sin_addr.s_addr = htonl (INADDR_ANY);
```

- When a client connects, verify originator's address

```
int clientsock, size;
struct sockaddr_in clientname;

if ( (clientsock = accept (serversock, &clientname,
                                    &size)) < 0)
    error ("Failed to accept client's connection\n");
msg ("Got a connection from %s\n",
        inet_ntoa (clientname.sin_addr));
```

C/C++ Lectures



Daemons and Networking

Karel Kubat
e-tunity

karel@kubat.nl / http://www.kubat.nl