15-441

Advice on Programming
Sept. 4, 2003

Topics

- Robust Programming
- Version Control
- Using scripting languages

About This Lecture

Version control / Source control

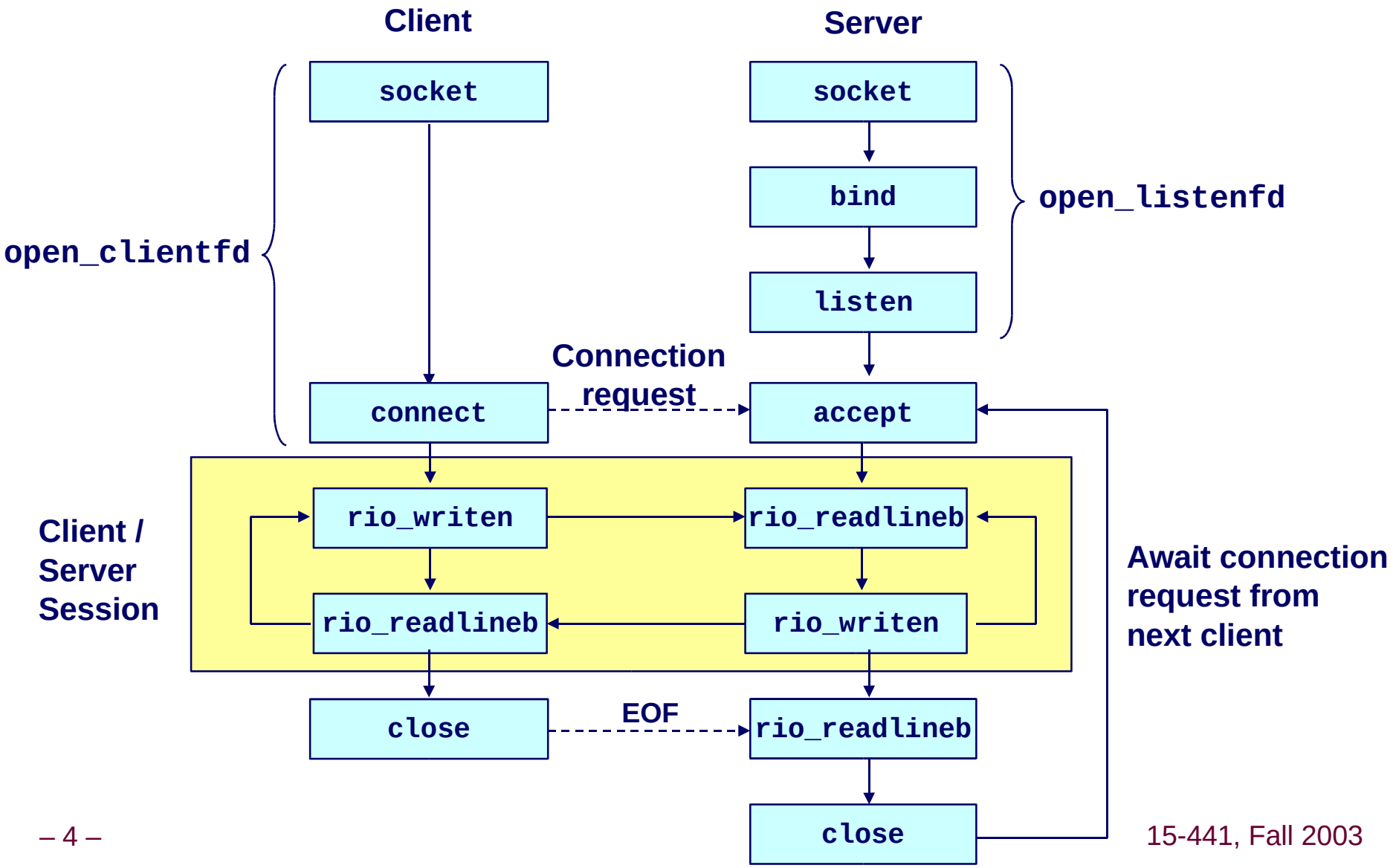
- Presented today: RCS
  - (by a rogue instructor)
- Rogue instructor's opinion of RCS: *extremely* obsolete
  - (Though it can be used effectively as a building block)
- Why will we discuss RCS today?
  - “Received wisdom” says...
    - » It's really hard to get students to use source control.
    - » Many refuse.
    - » The quickest thing to explain is RCS.
    - » Maybe if we describe something simple people might use it.

About This Lecture

Hope, fear, loathing, ...

- We hope you will use ***something*** to safeguard your sanity
- Easiest thing to explain is RCS
  - RCS is fine for Project 1
- Your “friends” may have already introduced you to CVS
  - “Friends don't let friends branch CVS”
  - It's fine with us if you use CVS
    - » (necessary evolutionary step, like tube worms)
- For a conceptually clearer time...
  - Try PRCS
  - It's small (not as small as RCS, not as big as CVS)
  - Most-important features of source control are ***default*** behaviors!
  - See last semester's PRCS intro from 15-410, posted on 441 site

Client/Server Code



Robustness Principles

Client

- Nothing user does/types should make program crash
  - Must perform complete checking for user errors

Server

- Nothing a client does should cause server to malfunction
  - Possibly malicious clients

Things to Worry About

- Error return codes by system calls
- String overflows
- Malformed messages
- Memory/resource leaks
  - Especially for server

Echo Client Main Routine

```
#include "csapp.h"

/* usage: ./echoclient host port */
int main(int argc, char **argv)
{
    int clientfd, port;
    char *host, buf[MAXLINE];
    rio_t rio;

    host = argv[1];
    port = atoi(argv[2]);

    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);

    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout);
    }
    Close(clientfd);
    exit(0);
}
```

No checking of command line arguments

Wrappers exit on error

fgets does not insert \n when string too long

Robust Version of Echo Client (1)

```
#include <limits.h>

/* To demonstrate truncation */
#define LINELEN 20

/* Maximum number of errors to tolerate before exiting */
int errlimit = 5;

void errcheck(char *message, int fatal)
{
  if (--errlimit == 0 || fatal) {
    fprintf(stderr, "Error: %s.  Exiting\n", message);
    exit(1);
  }
  fprintf(stderr, "Error: %s.  Continuing\n", message);
}

void usage(char *progname) {
  fprintf(stderr, "Usage: %s host port\n", progname);
  exit(0);
}
```

Robust Version of Echo Client (2)

```
int main(int argc, char **argv)
{
  int clientfd, port;
  char *host, buf[LINELEN];
  rio_t rio;

  if (argc != 3)
    usage(argv[0]);

  host = argv[1];
  port = atoi(argv[2]);

  if (port <= 0 || port > SHRT_MAX)
    errcheck("Invalid Port", 1);

  clientfd = open_clientfd(host, port);
  if (clientfd < 0)
    errcheck("Couldn't open connection to server", 1);

  rio_readinitb(&rio, clientfd);
  ...
```

Robust Version of Echo Client (3)

```
...
while (fgets(buf, LINELEN, stdin) != NULL) {
  int n;
  if (strlen(buf) == LINELEN-1 && buf[LINELEN-1] != '\n')
    strcpy(buf+LINELEN-5, "...\n"); /* Truncate string */
  if (rio_writen(clientfd, buf, strlen(buf)) < 0) {
    errcheck("Failed to send message", 0);
    continue;
  }
  if ((n = rio_readlineb(&rio, buf, LINELEN) <= 0)) {
    if (n == 0)
      errcheck("Unexpected EOF from server\n", 1);
    else
      errcheck("Failed to receive reply from server", 0);
  }
  if (fputs(buf, stdout) < 0)
    errcheck("Couldn't print reply\n", 0);
}
...
```

Robust Version of Echo Client (4)

```
  ...
  if (close(clientfd) < 0)
    errcheck("Couldn't close connection to server", 1);
  exit(0);
}
```

Design Issues

Error Classification & Recovery

- Fatal vs. nonfatal errors
  - Server code should only have fatal error when something is wrong on server machine
- What to do when when encounter nonfatal error
  - Skip to next activity
  - Server might close connection to malfunctioning client

Other Types of Errors

- Client dormant too long
  - Add timeouts to code
  - Gets very messy
- Denial of service attacks
  - Difficult to detect and/or handle

Version Control

Typical Problems in Managing Software Project

- Multiple people simultaneously edit single file
  - Want to prevent this or have some way to merge updates
- Bug appears in new version that was not detected in earlier version
  - Want to run tests on older version
- Customer reports problem with program. Turns out he/she has old version of code
  - Want to back up to earlier version of program
- Code evolves along incompatible paths by two groups
  - Want to reconcile into common version

Solution

- Implement some form of automatic version control

RCS

Revision Control System

- Basic Unix program(s) for managing software project
- Written by Walter Tichy, CMU PhD 1980

Basic Idea

- Code file foo.c has RCS version foo.c,v
  - Complete history of all versions of file
  - Stored in compacted form
- User can “check out” copy of file
  - Either read-only or writable
  - Even when writable, only single user can do so
  - Can check out older versions of program
- When file modified, can “check in” file
  - Increments version number
  - Becomes available for other users to check out

RCS Example

Code for Echo Server

```
% ls
Makefile  csapp.c  csapp.h  echoserver.c

% mkdir RCS # Directory for RCS files

% ci Makefile csapp.c csapp.h echoserver.c
# RCS prompts for descriptions of files
RCS/echoserver.c,v  <--  echoserver.c
enter description, terminated with single
'.' or end of file:
NOTE: This is NOT the log message!
>> Sequential echo server
>> .
initial revision: 1.1
done

%ls
RCS   # Files are gone!
```

RCS Example (cont.)

```
% ls RCS
Makefile,v  csapp.c,v  csapp.h,v  echoserver.c,v

% co RCS/*,v # Check out read-only copies of all files

% ls
Makefile csapp.c csapp.h echoserver.c

% co -l echoserver.c # Check out writable version
# Now edit echoserver.c to make it concurrent

$ ci -r2.1 echoserver.c # Check in with major version change
RCS/echoserver.c,v  <--  echoserver.c
new revision: 2.1; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Concurrent echo server using I/O multiplexing
>> .
done
```

Getting Revision History

```
% rlog echoserver.c
RCS file: RCS/echoserver.c,v
Working file: echoserver.c
head: 2.1
branch:
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 2;    selected revisions: 2
description:
Sequential echo server
----------------------------
revision 2.1
date: 2003/09/03 21:08:58;  author: bryant;  state: Exp;  lines:
+118 -27
Concurrent echo server using I/O multiplexing
----------------------------
revision 1.1
date: 2003/09/03 21:00:07;  author: bryant;  state: Exp;
Initial revision
```

Retrieving Earlier Version

```
% co -r1.1 echoserver.c
RCS/echoserver.c,v  -->  echoserver.c
revision 1.1
done
```

For More Information

Unix Man Pages

- rcsinfo
  - Overview of RCS
- ci
  - Check-in program
- co
  - Check-out program
- rcs
  - Overall control

RCS has lots of other features. These are only the basics.

Scripting Languages

General Features

- Easy to write “quick & dirty” code
  - Minimal type checking
  - Interpretive
- Good support for strings, regular expressions, invoking other programs

Scripting Languages

Examples

- awk, shell code
  - Developed originally at Bell Labs. Not very popular
- tcl
  - Developed by John Ousterhout (CMU PhD 1980)
  - Nice integration with tk graphics interface package
- perl
  - Developed by Larry Wall to aid system administration
  - Big & messy, but very powerful
- python
  - Developed by Guido van Rossum
  - Indentation is significant
    - » (ouch)

Echo Client in Perl

```
#!/usr/bin/perl -w

use sigtrap;
use IO::Socket;

$host = $ARGV[0];
$port = $ARGV[1];

$socket = IO::Socket::INET->new("$host:$port")
  || die("Couldn't connect to $host:$port: $!\n");


while (<STDIN>) {
  $line = $_;
  print $socket $line;
  $reply = <$socket>;
  print $reply;
}
```