Programming Tools

Henning Schulzrinne
Dept. of Computer Science
Columbia University

9-Mar-02  Advanced Programming
Spring 2002

What are tools for?

- Creating code modules
  - compiler
- Creating program from modules
  - linker
- Compiling groups of programs (dependencies)
- Debugging code
  - tracer, debugger, code checker
- Profiling and optimization
- Documentation: derive from code
- Coordination and "memory"
- Testing
- User installation
- User feedback

Compiler

- Convert source code to object modules
- .o: external references not yet resolved
- $ nm

Linker

- Combine .o and .so into single a.out executable module
- .so/.dll: dynamically loaded at run-time
- see "dl"
- $ ldd a.out

Creating a static library

- static library for linking: libsomething.a
  - create .o files: gcc --help.c
  - ar rlv libsomething.a *.o
  - ranlib libsomething.a
  - use library as gcc -L/your/dir -lsomething

Creating a dynamic library

- Details differ for each platform
  - gcc -shared -fPIC -o libhelper.so *.o
  - use same as for static (-library)
  - also LD_LIBRARY_PATH
Testing

- Every module and functionality needs to have an (automated) test
- Regression testing: change -> test old functionality
- Easy for simple functions
- Screen input/output?
- Complicated “test harness”

Program tracing

- Simple debugging: find out what system calls a program is using
- truss on Solaris, strace on Linux
- does not require access to source code
- does not show stdio calls, but can use -u Tlbc
- -f: follow children
- -p: attach to existing process (e.g., truss -p 27878 to see what process is doing when doing certain action)

strace

- Similar to truss, for Linux
- -T for timing

```
$ strace -t -T cat foo
```

Memory utilization: top

- Show top consumers of CPU and memory

```
| Task  | CPU  | Allocated | Memory
|-------|------|-----------|--------
| ProcA | 2.5% | 128MB     | 512MB  |
| ProcB | 3.2% | 256MB     | 1024MB |
| ProcC | 1.8% | 64MB      | 256MB  |
```
Debugging

- Interact with program while running
- Step-by-step execution
- Instruction
- Source line
- Procedure
- Inspect current state
- Call stack
- Global variables
- Local variables

$\text{gdb}$

```c
(gdb) \n
$\text{printf}("%d", i);$
(gdb) where
\#0 \text{loop} (i=3) at loop.c:4
\#1 \text{main} at main.c:1
(gdb) \n$s = 0$
(gdb) \n\text{breakpoint} 2 \text{ at } 0x1000; \text{ filp loop, line } 9.$
(gdb) continue
\text{Main}:
140
141
\ldots$
\text{breakpoint} 2, \text{ main} (\text{arg} = 1, \text{ argv} = \text{0x00001f09}) \text{ at loop.c:9}$
\text{frames } 0$
```

$\text{gdb hints}$

- Make sure your source file is around and doesn’t get modified
- Does not work (well) across threads
- Can be used to debug core dumps:
  \$ \text{gdb a.out core}$
  \#0 0x01003 in \text{main} (arg1=1, arg2=0x00000000) at loop.c:14
  "s = 0;"
  \text{(...)}
  \$s = 10$

$\text{gdb - execution}$

- `run `run program``
- `call `call function in program``
- `step `step N times into functions``
- `next `step N times over functions``
- `up `select stack frame that called current one``
- `down `select stack frame called by current one``

$\text{gdb - break points}$

- `break `set break point``
- `break `set break at function``
- `clear `delete breakpoint``
- `info `show breakpoints``
- `delete `delete break point 1``
- `display `display variable at each step``
Graphical interface: DDD

Installation
- Traditional:
  - tar (archive) file
  - compile
  - distribute binaries, documentation, etc.
- InstallShield
- Linux RPM
- Solaris pkg

Building programs
- Programs consist of many modules
- Dependencies:
  - if one file changes, one or more others need to change
  - .c depends on .h -> re-compile
  - .o depends on .c -> re-compile
  - executable depends on .o's -> link
  - library depends on .o -> archive
  - recursive

make
- make maintains dependency graphs
- based on modification times
- Makefile as default name
- make [-f makefile] [target]
- if node newer than child, remake child
  target ...: dependency
  command
  command

make
all: hello clean
clean:
rm -f *.o
helper.o: helper.c
OBJ = helper.o \\
hello.o
hello: $(OBJ)
$(CC) $(CFLAGS) $(LDFLAGS) -o $0 $(OBJ)

make variables
- $@ name of current target
- $? list of dependencies newer than target
- $< name of dependency file
- $^{"} base name of current target
- $% for libraries, the name of member
- implicit rules, e.g., a .c file into .o
  .c.o:
  $(CC) $(CFLAGS) $<
make depend

depend: $(CFILES) $(HFILES)
$(CC) $(CFLAGS) -M $(CFILES) > .state
# works for GNU make and BSD make
#if 0
#include .state
#else
#include ".state"
#endif

make depend - alternative

- can also use makedepend program
- appends and replaces rules after
  # DO NOT DELETE
- e.g.,
  SRCs = file1.c file2.c ...
  CFLAGS = -O -DNACK -I../foobar -xyz
  depend:
  makedepend -- $(CFLAGS) -- $(SRCS)

make environment

- Environment variables (PATH, HOME, USER, etc.) are available as $(PATH), etc.
- Also passed to commands invoked
- Can create new variables (gmake):
  export FOOBAR = foobar

User feedback - bug tracking

- Automatically capture system crash information
- non-technical users
- privacy?
- e.g., Netscape Talkback
- User and developer bug tracking
- make sure bugs get fixed
- estimate how close to done

Bug tracking

- Bugzilla

Development models

- Integrated Development Environment (IDE)
  - integrate code editor, compiler, build environment, debugger
  - graphical tool
  - single or multiple languages
    - VisualStudio, Creator, Arke, ...
  - Unix model
    - individual tools, command-line
Source code management

- problem: lots of people working on the same project
- source code (C, Perl, ...)
- documentation
- specification (protocol specs)
- mostly on different areas
- versions
- released – maintenance only
- stable – about to be released, production use
- development, beta
- different hardware and □ versions

---

cvs: overview

- version control system
- see also RCS or SCCS
- collection of directories, one for each module
- release control
- concurrent revisions: “optimistic”
- network-aware
- single master copy (repository) □ local (developer) copies
- see http://www.cs.columbia.edu/~hgs/cvs

---

What cvs isn’t/doesn’t...

- build system
- project management
- talking to your friends
- change control:
  - all changes are isolated vs. single logical change
  - bug fix tracking
  - track change verification
- testing program (regression testing)
- work flow or process model

---

cvs: setting up a repository

1. create directory (e.g.) cvsroot -> environment variable or -d
2. cvs -d /usr/local/cvsroot init
   □ creates CVSROOT directory for maintenance files
cvsversion
- CVSROOT
  - history, logininfo, modules, passwd, ...
  - testcvs
  - hello.c,v
  - Makefile.c,v

---

cvs: adding a module to a repository

Source files in src/testcvs
1. setenv CVSROOT ./src/cvsroot/# or
   not a browseable root
2. cd testcvs to your working directory
3. cvs import rdr vendortag releasetag:
   create rdr under $CVSROOT/ repository
   from current directory, with tag vendortag
   for branch, tag releasetag for release
   (generally, “start”); creates branch 1.1.0
   with cvsroot/testcvs/hello.c,v

---

cvs: adding a module

$ cvs -g import -o "Sample program" testcvs sample
  start
  hello.c
  # conflicts created by this import
  1. $add module name to cvsroot/CVSROOT/modules
  2. $checkout if you can
     $cvs checkout
     $cvs checkout
     $cvs commit modules
     $cd ...
     $cvs release -d CVSROOT # only if no longer needed
     you have (0) altered files in this repository.
     Are you sure you want to release (and delete) directory
     ‘cvsroot’? 
cvs: adding a user
1. ypcat passwd | fgrep alice
2. add user entry to CVSROOT/passwd
   alice:323Gho7813dSV: hgs
3. add entry to loginfo to generate email
   testcvs /usr/ucb/Mail -s "Ms" alice bob

---

cvs: using a repository
- as a developer, login if on remote server:
  cvs add
  cvs del
- only needed once – stored in $HOME/.cvs/pass

---

cvs: using a repository
- Check out the source code files from repository:
  cvs checkout testcvs
  cvs checkout: Updating testcvs
  U testcvs/hello.c
  ls -l
  ...
  CVS/ Makefile hello.c
  CVS:
  Entries Repository Root

---

cvs: committing changes
- create or edit a file
- add file if new
  $ cvs add makefile
  cvs add: scheduling file 'makefile' for addition
  cvs add: use 'cvs commit' to add this file permanently

---

cvs: committing changes
- commit changes (all files based on modification date):
  $ cvs commit
  checking in hello.c
  /home/hgp/src/cvsroot/temcvsv/hello.c,v <-- hello.c
  new revision 1.2; previous revision 1.1
  done

---

cvs: catching up
- No notification beyond email.
- Always update before editing
  $ cvs update
  cvs update: Updating .
  M hello.c
- merges changes, may produce conflicts
- output:
  U file updated: file not in working directory
  or no local changes
  M file modified, merged
  C file conflict detected, marked by >>> ... 
  ? file stray file in working directory
cvs: deleting files

- delete first, then remove from CFS
  - rm -f notes.txt
  - cvs revdel notes.txt
- shortcut: `cvs rm -f notes.txt`
- ends up in .cvs, i.e., can be restored

9-Mar-02 Advanced Programming Spring 2002 43

cvs: viewing differences

- Difference between checked out and working copy:
  - `cvs diff hello.c`
  - `cvs diff -rl.6 hello.c`
  - `cvs diff -rl.6 hello.c`
  - `cvs diff -rl.6 hello.c`

9-Mar-02 Advanced Programming Spring 2002 44

Cvs: revisions

- each revision increases rightmost number by one: 1.1, 1.2, ...
- more than one period -> branches
- versions of file = CFS revisions
- (released) versions of software = CFS releases
- new file gets highest first digit
- `cvs commit -r 2.0`: makes all revisions to 2.0
- `cvs update -A` goes to latest

9-Mar-02 Advanced Programming Spring 2002 45

Cvs: revision tagging

- Use cvs tag to tag revisions (software release)
  - `cvs tag `release`
  - `cvs tag -m hello.c`
  - `cvs status -r hello.c`
  - `cvs status -m hello.c`
  - `cvs status -r hello.c`

9-Mar-02 Advanced Programming Spring 2002 46

Cvs: branches

- released (stable) vs. development (unstable, main branch) version
- branch on revision tree for released version
  - `cvs tag -b rel-1-fix`
  - `cvs rtag -b rel-1 rel-1-fixes testcvs`

9-Mar-02 Advanced Programming Spring 2002 47

Cvs: history

- `cvs annotate hello.c`

9-Mar-02 Advanced Programming Spring 2002 48
cvs: notifications

- **cvs status** reports status
  
  `cvs: hgto.c
  status: up-to-date`

- modified steps:
  
  `modif: hgto.c
  status: out-of-date`

- watch certain files for modifications:
  
  `cvs watch on hello.c
  -> cvs edit hello.c needed`

Other source-code management systems

- IDE for Java:
  
  - IDE with a compiler, debugger, etc. and C++ built in

- Microsoft **Visual SourceSafe**
  
  - library system, i.e., only one user can check out a specific file at any time

Which file is this?

- find out in binary which version was used
  
  `Log`

- static char \*id="@(#) $Id$"
  
  becomes on checkout

  `ident hello or what hello
  hello:
  id: hello.c,v 2.1 2002/02/21 20:46:16 hgs wp $`

RPM - RedHat Linux package manager

- Activities for an application:
  
  - Installation – on different architectures
  - Updates
  - Inventory: what's installed
  - Un-install

- Each Unix architecture seems to have one: Solaris pkg, RPM (www.rpm.org), ...

RPM

- Package label, e.g., perl-0.01m-4:
  
  - software name
  - software version
  - package release

- Package-wide information
  
  - date and time built
  - description of contents
  - total size of all files
  - grouping information
  - digital signature
**RPM**
- Per-file information:
  - name of file and where to install it
  - file permissions
  - owner and group specification
  - MD5 checksum
  - file content

*Using rpm*
- rpm -i  install package, check for dependencies
- rpm -e  erase package
- rpm -U  upgrade package
- rpm -q  query packages (e.g., -a = all)

*rpm -q*

```
rpm -q
NAME  : telnet
version: (not relocatable)
release: 0.1
website: http://www.redhat.com
license: (not relocatable)
```

http://www.redhat.com/docs/books/max-rpm/
- but: current version (4.0) is a bit different

**RPM**

**Building your own rpm**
- Either in /usr/src/redhat or create your own:
  - CPB
  - RPM/5.1.8
  - SOURCE: *.tgz
  - SPECS: build specification
  - SRPMS: source RPMs, (.src.rpm)

**Building your own rpm: spec**

```
# spec: file for hello world rpm
# Summary: hello world
# Version: 1.0
# Buildroot: /usr/src/redhat
# Buildroot: /usr/src/redhat
# License: GPL
# License: GPLv2
```

This makes it one of the most famous C programs.
Building your own rpm: spec

```
rpmbuild -s /home/hgs/rop/2002/9-Mar-02-Advanced-Programming
rpmbuild --create spec
```

Memory leaks and overruns

- **see**
  
  [link](http://www.cs.colorado.edu/users/inval/public_html/MallocDebug.html)
- **Graphical tool**: purify
- **Simple library**: ElectricFence
  - catches
  - overruns a malloc() boundary
  - touch (read, write) memory released by free()
  - places inaccessible (CM) memory page after each allocation
  - only for debugging (memory hog)

```bash
dmalloc -l logfile -s /home/hgs/sun5/lib/dmalloc
```

ElectricFence

- gcc -g test.c -o /home/hgs/sun5/lib -leffence
- `test`
  - `malloc`:
    - `malloc(0)` returns 0
  - `free(0)` returns 0

```bash
dmalloc output
```

- grep `malloc` shell script output
  - `malloc(0)` returns 0
profiling

- execution profile of call graph
- Example:
  ```c
  void f1() { 
    a1 = a2 + a3; 
    b1 = b2 * b3; 
  }

  void f2() { 
    if (x > 3) { 
      a2 = a4; 
      a3 = a5; 
    } 
  }

  void f3() { 
    if (y > 3) { 
      a4 = a6; 
    } else { 
      a5 = a7; 
    }
  }

  void f4() { 
    if (z > 3) { 
      a6 = a8; 
    } else { 
      a7 = a9; 
    }
  }
  ```

profiling

- gcc -pg nested.c -o nested
- change function invocation to do logging (call _count)
- also, PC sampling (e.g., 100 times/second)
- generate a call graph
- gprof nested gmon.out

gprof flat profile

```
gprof flat profile

<table>
<thead>
<tr>
<th>Function</th>
<th>Self Time</th>
<th>Shared Time</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1()</td>
<td>0.02</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>f2()</td>
<td>0.01</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>f3()</td>
<td>0.005</td>
<td>0.01</td>
<td>0.015</td>
</tr>
<tr>
<td>f4()</td>
<td>0.005</td>
<td>0.01</td>
<td>0.015</td>
</tr>
</tbody>
</table>
```

gprof call graph

```
gprof call graph

```

doc++

- documentation system for C/C++ and Java
- generate LaTeX for printing and HTML for viewing
- hierarchically structured documentation
- automatic class graph generation (Java applets for HTML)
- cross references
- formatting (e.g., equations)
doc++

```c
/*
 * This is the famous "hello world" program, with rude comments that can;
 */
//
//  Copyright H.W. programmer
//  $Id: hello.c $ $Date: 2002/03/09 10:00:00 $ $Revision: 1.2 $
//
//  Usage: print("Hello world!");

#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello world!");
    return 0;
}
```

**Other tools useful to know**

- configuration:
  - autoconf: configuration files
  - automake: make files

- code generation:
  - indent (e.g., indent -kr -1 2 hello.c): automated indentation for C programs
  - lex, flex: lexical analyzers
  - yacc, bison: compiler generator

- docify to create minimal version
- doc++ -d outdir hello.c