Programming Tools

Henning Schulzrinne
Dept. of Computer Science
Columbia University

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
                 U printf
0000000000000000 t
0000000000000000 d
0000000000000000 b
0000000000000000 r
0000000000000000 ?
0000000000000000 a *ABS*
0000000000000000 T c
0000000000000000 a const.c
0000000000000048 T main
```


Linker

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

```
libc.so.1 =>     /usr/lib/libc.so.1
libdl.so.1 =>    /usr/lib/libdl.so.1
/usr/platform/SUNW,Ultra-5_10/lib/libc_psr.so.1
```


Creating a static library

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

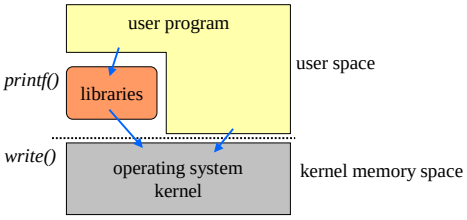
Creating a dynamic library

- Details differ for each platform
- `gcc -shared -fPIC -o libhelper.so *.o`
- use same as for static (-l*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



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 libc`
- `-f`: follow children
- `-p`: attach to existing process (e.g., `truss -p 27878` to see what process is doing when doing certain action)

truss example

```
$ truss a.out
execve("a.out", 0xFFBEF6FC, 0xFFBEF704)  argc = 1
mmap(0x00000000, 8192, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE|MAP_ANON, -1, 0) = 0xFF3A0000
resolvepath("/usr/lib/ld.so.1", "/usr/lib/ld.so.1", 1023) = 16
open("/var/ld/ld.config", O_RDONLY)             Err#2 ENOENT
open("/opt/CUCStc1/lib/libc.so.1", O_RDONLY)    Err#2 ENOENT
open("/opt/CUCStc18.3/lib//libc.so.1", O_RDONLY) Err#2 ENOENT
open("/usr/openwin/lib/libc.so.1", O_RDONLY)    Err#2 ENOENT
open("/usr/local/lib/libc.so.1", O_RDONLY)      Err#2 ENOENT
...
ioctl(1, TCGETA, 0xFFBEF45C)                    = 0
Hello World
write(1, " H e l l o   W o r l d\n", 12)        = 12
llseek(0, 0, SEEK_CUR)                          = 19444
_exit(0)
```


strace

- similar to truss, for Linux
- -T for timing
- `$ strace -t -T cat foo`

```
14:26:59 open("foo", O_RDONLY|O_LARGEFILE) = 3 <0.000712>
14:26:59 fstat(3, {st_mode=S_IFREG|0644, st_size=6, ...}) = 0
    <0.000005>
14:26:59 brk(0x8057000)                 = 0x8057000 <0.000011>
14:26:59 read(3, "hello\n", 32768)      = 6 <0.000010>
14:26:59 write(1, "hello\n", 6hello
)         = 6 <0.000015>
14:26:59 read(3, "", 32768)             = 0 <0.000005>
14:26:59 close(3)                       = 0 <0.000010>
14:26:59 _exit(0)                       = ?
```


Memory utilization: top

- Show top consumers of CPU and memory

```
load averages:  0.42,  0.22,  0.16                                  14:17:35
274 processes: 269 sleeping, 1 zombie, 3 stopped, 1 on cpu
CPU states: 81.3% idle,  5.2% user, 13.4% kernel,  0.1% iowait,  0.0% swap
Memory: 512M real, 98M free, 345M swap in use, 318M swap free

   PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
   144 root       1  53    0 3384K 1728K sleep   33.3H  3.67% ypserv
 11011 hgs        1  48    0 2776K 2248K sleep    0:00  0.57% tcsh
 11040 hgs        1  55    0 1800K 1352K cpu/0    0:00  0.39% top
   281 root       1  58    0 4240K 2720K sleep  313:03  0.38% amd
 10933 kbutler    1  58    0   11M 8376K sleep    0:00  0.17% lisp
  1817 yjh9       1  58    0 8968K 7528K sleep    0:39  0.10% emacs
 13955 yjh9       1  58    0 8496K 7200K sleep    2:47  0.09% emacs
```

Debugging

- Interact with program while running
  - step-by-step execution
    - instruction
    - source line
    - procedure
  - inspect current state
    - call stack
    - global variables
    - local variables

Debugging

- Requires compiler support:
  - generate mapping from PC to source line
  - symbol table for variable names
- Steps:

```
$ gcc -g -o loop loop.c
$ gdb loop
(gdb) break main
(gdb) run foo
Starting program: src/test/loop

Breakpoint 1, main (argc=2, argv=0xffbef6ac) at loop.c:5
5         for (i = 0; i < 10; i++) {
```


gdb

```
(gdb) n
6 printf("i=%d\n", i);
(gdb) where
#0  loop (i=1) at loop.c:4
#1  0x105ec in main (argc=2, argv=0xffbef6a4) at loop.c:11
(gdb) p i
$1 = 0
(gdb) break 9
Breakpoint 2 at 0x105e4: file loop.c, line 9.
(gdb) cont
Continuing.
i=0
i=1
...
Breakpoint 2, main (argc=1, argv=0xffbef6ac) at loop.c:9
9   return 0;
```


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:

```
$ gdb a.out core
#0  0x10604 in main (argc=1, argv=0xffbef6fc) at loop.c:14
*s = '\0';
(gdb) print i
$1 = 10
```

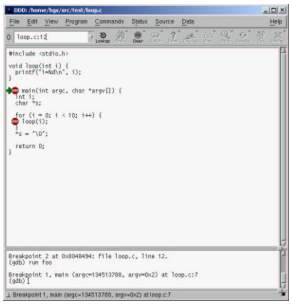

gdb - execution

run *arg*	run program
call *f(a,b)*	call function in program
step *N*	step *N* times into functions
next *N*	step *N* times over functions
up *N*	select stack frame that called current one
down *N*	select stack frame called by current one

gdb – break points

break *main.c:12*	set break point
break *foo*	set break at function
clear *main.c:12*	delete breakpoint
info break	show breakpoints
delete *1*	delete break point 1
display *x*	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
```

tab!

make

```
all: hello clean
clean:
  rm -f *.o
helper.o: helper.c
OBJ = helper.o \
  hello.o
hello: $(OBJ)
  $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $(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
#endif
#include ".state"
```


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 -DHACK -I../foobar -xyz
depend:
  makedepend -- $(CFLAGS) -- $(SRCS)
```


make environment

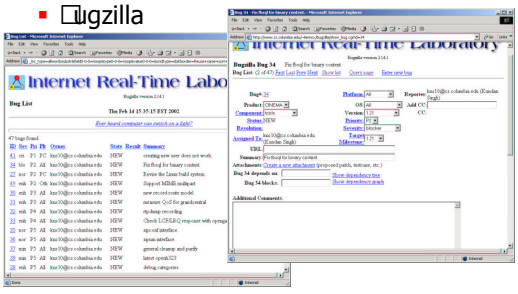
- Environment variables (`PATH`, `HOME`, `USER`, etc.) are available as `$(PATH)`, etc.
- Also passed to commands invoked
- Can create new variables (gmake):
  `export FOOBAR = foobar`

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, Forte, ...
- 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 OS 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

```
cvsroot
    CVSROOT
        history, loginfo, 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

```
cvs -d :pserver:alice@tune.cs.columbia.edu:/u/kon/hgs/src/cvsroot login
```

2. `cd testcvs` to your working directory
3. `cvs import` *`rdir vendortag releasetag`*: create rdir under $CVSROOT/ repository from current directory, with tag vendortag for branch, tag releasetag for release (generally, "start"); creates branch 1.1.1 with cvsroot/testcvs/hello.c,v

cvs: adding a module

```
$ cvs -t import -m "Sample program" testcvs sample
    start
N testcvs/hello.c
No conflicts created by this import
```

1. add module name to cvsroot/CVSROOT/modules

```
testcvs testcvs
something directory/something_else
```

2. use `cvs checkout` if you can

```
$ cvs checkout CVSROOT/modules
$ cd CVSROOT
$ vi modules
$ 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:J21GHe78i3d5Y:hgs
```

- add entry to `loginfo` to generate email

```
testcvs /usr/ucb/Mail -s "%s" alice bob
```


cvs: using a repository

- $ a developer, login if on remote server:

```
cvs -d
   :pserver:alice:secret@tune.cs.columbia.edu:/u/kon/hgs/src/cvsroo
   t login
```

- nly needed once – stored in `$HOME/.cvspass`

cvs: using a repository

- Check out the source code files from repository:

```
cvs checkout testcvs
cvs checkout: Updating testcvs
U testcvs/hello.c

ls -R
.:
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/hgs/src/cvsroot/testcvs/hello.c,v  <--  hello.c
new revision: 1.6; previous revision: 1.5
done
```


cvs: catching up

- No notification beyond email.
- lways 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 CVS

```
$ rm notes.txt
$ cvs remove notes.txt
cvs remove: scheduling `notes.txt' for removal
cvs remove: use 'cvs commit' to remove this file permanently
Removing notes.txt;
/home/hgs/src/cvsroot/testcvs/notes.txt,v  <--  notes.txt
new revision: delete; previous revision: 1.2
done
```

- shortcut: `cvs remove -f notes.txt`
- ends up in Attic, i.e., can be restored

cvs: viewing differences

- Difference between checked out and working copy:

```
$ cvs diff hello.c
Index: hello.c
===================================================================
RCS file: /home/hgs/src/cvsroot/testcvs/hello.c,v
retrieving revision 1.6
diff -r1.6 hello.c
31a32
>   printf("John Doe\n");
```


cvs: revisions

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

cvs: revision tagging

- Use cvs tag to tag revisions (software release)

```
$ cvs tag rel-0 hello.c
T hello.c
$ cvs status -v hello.c
===================================================================
File: hello.c           Status: Up-to-date

   Working revision:    2.1     Thu Feb 21 20:46:56 2002
   Repository revision: 2.1     /home/hgs/src/cvsroot/testcvs/hello.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
        ap2002                          (branch: 2.0.2)
        rel-0                           (revision: 1.2)
        start                           (revision: 1.1.1.1)
        sample                          (branch: 1.1.1)
```


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
```


cvs: history

- `cvs annotate hello.c`

```
Annotations for hello.c
***************
1.1          (hgs      08-Sep-99): int main(int argc, char *argv[])
1.1          (hgs      08-Sep-99): {
1.5          (hgs      21-Feb-02):   /* this is the classical hello world output
   */
1.1          (hgs      08-Sep-99):   printf("hello world!\n");
1.6          (hgs      21-Feb-02):   printf("Henning Schulzrinne\n");
2.0          (hgs      21-Feb-02):   printf("John Doe\n");
1.2          (hgs      08-Sep-99):
2.1          (hgs      21-Feb-02):   exit(0);
1.1          (hgs      08-Sep-99): }
```

cvs: notifications

- `cvs status` reports status

```
File: hello.c          Status: Up-to-date

   Working revision:    2.1     Thu Feb 21 20:46:56 2002
   Repository revision: 2.1
/home/hgs/src/cvsroot/testcvs/hello.c,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)
```

- watch certain files for modifications:

```
$ cvs watch on hello.c
-> cvs edit hello.c needed
$ cvs watch off hello.c
```


cvs notifications

- **`cvs watch add`**
- **`cvs watchers`** : list people watching

```
$ cvs watchers
hello.c hgs     edit    unedit  commit
```

- cvs editors: current list of editors

```
$ cvs editors
hello.c hgs  Thu Feb 21 21:00:56 2002 GMT  bart.cs.columbia.edu    /tmp/testcvs
```


Other source-code management systems

- IBM Visual Age for Java:
  - IDE with a compiler, debugger, etc. and CVS built in
- Microsoft Visual SourceSafe
  - library system, i.e., only one user can check out a specific file at any given time

Which file is this?

- find out in binary which version was used
- `$Log$`
- `static char *id="@(#) $Id$"`

becomes on checkout

```
static char *id="@(#) $Id: hello.c,v 2.1 2002/02/21 20:46:56 hgs Exp $";
```

- `ident hello` or `what hello`

```
hello:
        $Id: hello.c,v 2.1 2002/02/21 20:46:56 hgs Exp $
      SunOS 5.8 Generic February 2000
```


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-5.001m-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
  - MD□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 -i telnet
Name        : telnet                       Relocations: (not relocateable)
Version     : 0.17                               Vendor: Red Hat, Inc.
Release     : 18.1                      Build Date: Wed Aug 15 15:08:03 2001
Install date: Fri Feb  8 16:50:03 2002  Build Host: stripples.devel.redhat.com
Group       : Applications/Internet     Source RPM: telnet-0.17-18.1.src.rpm
Size        : 88104                     License: BSD
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Summary     : The client program for the telnet remote login protocol.
Description :
Telnet is a popular protocol for logging into remote systems over the
Internet.  The telnet package provides a command line telnet client.

Install the telnet package if you want to telnet to remote machines.

This version has support for IPv6.
```


RPM

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

Building your own rpm

- Either in /usr/src/redhat or create your own:
  - □UILD
  - RPMS/i□□6: *.i□□6.rpm
  - S□URCES: *.tgz
  - SPECS: build specification
  - SRPMS: source RPMS, (.src.rpm)

Building your own rpm: spec

```
#
# spec file for hello world app
#
Summary: hello world
Name: hello
Version: 1.0
Release: 1
Copyright: GPL
Group: Applications/Test
Source: http://www.cs.columbia.edu/IRT/software/
URL: http://www.cs.columbia.edu/IRT/software/
Distribution: Columbia University
Vendor: IRT
Packager: Henning Schulzrinne <hgs@cs.columbia.edu>
BuildRoot: /home/hgs/src/rpm

%description
The world's most famous C program.
```

Building your own rpm: spec

```
%prep
rm -rf $RPM_BUILD_DIR/hello-1.0
zcat $RPM_SOURCE_DIR/hello-1.0.tgz | tar -xvf -

%build
make

%install
make ROOT="$RPM_BUILD_ROOT" install

%files
%doc README
/usr/local/bin/hello
/usr/local/man/man1/hello.1

%clean
```


Building your own rpm

- create ~/.rpmmacros

  `%_topdir /home/hgs/src/test/rpm`
- `cd /home/hgs/src/test/rpm/SPECS`
- `rpm -ba --buildroot /home/hgs/tmp hello-1.0.spec`
- creates binary and source RPM

Memory leaks and overruns

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

ElectricFence

- `gcc -g test.c -L/home/hgs/sun5/lib -lefence -o test`

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
int main(int argc, char *argv[])
{
  char *s = malloc(5);
  strcpy(s, "A very long string");
  return 0;
}
```

- use gdb:

```
Program received signal SIGSEGV, Segmentation fault.
0xff2b2f94 in strcpy () from /usr/lib/libc.so.1
(gdb) up
#1  0x10adc in main (argc=1, argv=0xffbef684) at test.c:10
10        strcpy(s, "A very long string");
```


dmalloc - memory leaks

$ dmalloc -l logfile -i 100 high

`setenv DMALLOC_OPTIONS debug=0x4f47d03,inter=100,log=logfile`

- create file

```
#ifdef DMALLOC
#include "dmalloc.h"
#endif
```

- link: `gcc -g -DDMALLOC dmalloc.c -L/home/hgs/sun5/lib/ -ldmalloc -o dm`
- run program

dmalloc output

```
1014925598: 1: Dmalloc version '4.8.2' from 'http://dmalloc.com/'
1014925598: 1: flags = 0x4f47503, logfile 'logfile'
1014925598: 1: interval = 100, addr = 0, seen # = 0
1014925598: 1: starting time = 1014925598
1014925598: 1: free bucket count/bits:  255/5
1014925598: 1: basic-block 8192 bytes, alignment 8 bytes, heap grows up
1014925598: 1: heap: 0x64000 to 0x6a000, size 24576 bytes (3 blocks)
1014925598: 1: alloc calls: malloc 1, calloc 0, realloc 0, free 0
1014925598: 1: alloc calls: recalloc 0, memalign 0, valloc 0
1014925598: 1:  total memory allocated: 10 bytes (1 pnts)
1014925598: 1:  max in use at one time: 10 bytes (1 pnts)
1014925598: 1: max alloced with 1 call: 10 bytes
1014925598: 1: max alloc rounding loss: 22 bytes (68%)
1014925598: 1: max memory space wasted: 8138 bytes (99%)
1014925598: 1: final user memory space: basic 0, divided 1, 8170 bytes
1014925598: 1:  final admin overhead: basic 1, divided 1, 16384 bytes (66%)
1014925598: 1:  final external space: 0 bytes (0 blocks)
1014925598: 1: top 10 allocations:
1014925598: 1:  total-size  count in-use-size  count  Source
1014925598: 1:          10      1          10      1  dmalloc.c:8
1014925598: 1:          10      1          10      1  Total of 1
1014925598: 1: dumping not-freed pointers changed since 0:
1014925598: 1:  not freed: '0x68008[s1]' (10 bytes) from 'dmalloc.c:8'
1014925598: 1:  total-size  count  Source
1014925598: 1:          10      1  dmalloc.c:8
1014925598: 1:          10      1  Total of 1
1014925598: 1:  known memory: 1 pointer, 10 bytes
1014925598: 1: ending time = 1014925598, elapsed since start = 0:00:00
```

profiling

- execution profile of call graph
- Example:

```
int inner(int x) {
  static int sum;
  sum += x;
  return sum;
}

double outer(int y) {
  int i;
  double x = 1;
  double sum = 0;
  for (i = 0; i < 10000; i++) {
    x *= 2; sum += inner(i + y);
  }
  return sum;
}

int main(int argc, char *argv[])
{
  int i;
  for (i = 0; i < 1000; i++) {outer(i);}
  exit(0);
}
```


profiling

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

gprof flat profile

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 59.50      2.88     2.88                             internal_mcount
 21.69      3.93     1.05     1000     1.05     1.92  outer
 17.15      4.76     0.83 10000000     0.00     0.00  inner
  0.83      4.80     0.04     1000     0.04     0.04  _libc_write
  0.83      4.84     0.04                             _mcount
  0.00      4.84     0.00     2000     0.00     0.00  _realbufend
  0.00      4.84     0.00     2000     0.00     0.00  ferror_unlocked
  0.00      4.84     0.00     1890     0.00     0.00  .mul
  0.00      4.84     0.00     1000     0.00     0.04  _doprnt
  0.00      4.84     0.00     1000     0.00     0.04  _xflsbuf
  0.00      4.84     0.00     1000     0.00     0.00  memchr
  0.00      4.84     0.00     1000     0.00     0.04  printf
```


gprof call graph

- Time spent in function and its children

```
index % time    self  children    called     name
                                              <spontaneous>
[1]     60.0    2.88    0.00                 internal_mcount [1]
                0.00    0.00       1/3           atexit [15]
-----------------------------------------------
                1.05    0.87    1000/1000        main [3]
[2]     40.0    1.05    0.87    1000         outer [2]
                0.83    0.00 10000000/10000000     inner [5]
                0.00    0.04    1000/1000        printf [6]
-----------------------------------------------
                0.00    1.92       1/1           _start [4]
[3]     40.0    0.00    1.92       1         main [3]
                1.05    0.87    1000/1000        outer [2]
                0.00    0.00       1/1           exit [19]
-----------------------------------------------
                                              <spontaneous>
[4]     40.0    0.00    1.92                 _start [4]
                0.00    1.92       1/1           main [3]
                0.00    0.00       2/3           atexit [15]
-----------------------------------------------
                0.83    0.00 10000000/10000000     outer [2]
[5]     17.3    0.83    0.00 10000000          inner [5]
```

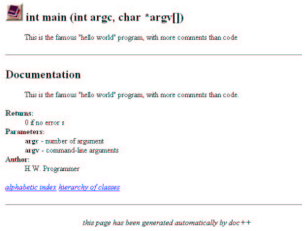
caller

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++

- Special comments: /** */, ///



int main (int argc, char *argv[])

This is the famous "hello world" program, with more comments than code.

Documentation

This is the famous "hello world" program, with more comments than code.

Returns:
0 if no error s
Parameters:
argc - number of argument
argv - command-line arguments
Author:
H.W. Programmer

alphabetic index hierarchy of classes

this page has been generated automatically by doc++

doc++

```
/**
This is the famous "hello world" program, with more comments than code.

@author H.W. Programmer
@return 0 if no error
@param argc  number of argument
@param argv  command-line arguments
@returns
*/
#include <stdio.h>
int main(int argc, char *argv[]) {
  printf("Hello World!");
  return 0;
}
```


doc++

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

Other tools useful to know

- configuration:
  - autoconf: configuration files
  - automake: make files
- code generation:
  - indent (e.g., `indent -kr -i2 hello.c`): automated indentation for C programs
  - lex, flex: lexical analyzers
  - yacc, bison: compiler generator