

Computer Programming

Lecture 6

이윤진
서울대학교
2007.1.7.

Slide Credits

- 엄현상 교수님
 - 서울대학교 컴퓨터공학부
 - Computer Programming, 2007 봄학기

gdb and awk

순서

- C Compiler and Linker 보충
 - Static vs Shared Libraries (계속)
- gdb
- awk
- Q&A

Shared vs Static Libraries (계속)

- Advantage of Using Libraries
 - Reduced Complexity of Managing Projects
- Advantages of Using Shared Libraries
 - Reduced Size of Each Executable File
 - Easy Library-Functions Replacement
- Disadvantage of Using Shared Libraries
 - Dynamic Linking Overhead

gdb 기본

- Features
 - Run a Program
 - Stop Execution within the Program
 - Examine and Change Variables during Execution
 - Trace How the Program Executes
 - Provide Command-Line Editing and History Features

Using gdb

- Compilation with the `-g` Option
 - Generate an Expanded Symbol Table for Use with gdb

```
martini:~$ gcc -g main.c func.o
```

- Execution of gdb
 - Permit Debugging a Program Compiled with the `-g` Option
 - Permit Discovering Where and Why the Program Failed (with a Core Dump)

```
martini:~$ gdb a.out  
martini:~$ gdb a.out core
```

Core dump: copy of memory image;
run `'ulimit -c unlimited'` to make the size unlimited

gdb Basic Commands

- help: Print a List of Commands or Topics
- run: Execute the Program
- quit: Exit gdb

```
(gdb) help
```

```
List of classes of commands
```

```
... (omitted)
```

```
running – Running the program
```

```
...
```

```
(gdb) help running
```

```
...
```

```
run – Start debugged program
```

```
...
```

```
(gdb) run 1
```

Argument list

```
...
```

```
(gdb) quit
```

[Ctrl]-C to stop the program

gdb Example

```
/*          test.c          */
struct {
    int *i_p;
} s;
void foo() {
    *(s.i_p) = 2;
    printf("s.i = %d\n", *(s.i_p));
}
main(int argc, char *argv[]){
    int *i_p;
    *i_p = atoi(argv[1]);
    printf("i = %d\n", *i_p);
    foo();
}
```

```
martini:~$ gcc -g test.c
```

```
martini:~$ a.out 1
```

```
i = 1
```

```
세그멘테이션 오류 (core dumped)
```

gdb Example (계속)

- list: Show the Lines Surrounding the Code Just Executed
- whatis: Print the Type of a Variable or Function
- ptype: Show the Contents of a Data Type

```
martini:~$ gcc a.out core
```

```
... (omitted)
```

```
Program terminated with signal 11, Segmentation fault.
```

```
...
```

```
#0 0x0804846b in foo () at test.c:6
```

```
6      *(s.i_p) = 2;
```

```
(gdb) list
```

```
...
```

```
*(s.i_p) = 2;
```

```
...
```

```
(gdb) whatis s
```

```
type = struct {...}
```

```
(gdb) ptype s
```

```
type = struct {
```

```
    int *i_p;
```

```
}
```

Call stack number - 0: most recent

Line number

more powerful than whatis

gdb Example (계속)

- print: Print the Value of a Variable or Expression
- up/down: Move Up/Down the Stack Frame to Make another Function the Current One

```
(gdb) print s
$1 = {i_p = 0x0}
(gdb) print s.i_p
$2 = (int *) 0x0
(gdb) print *(s.i_p)
Cannot access memory at address 0x0
(gdb) up
#1 0x080484c9 in main (argc=2, argv=0xbffffce4) at test.c:15
15  foo();
(gdb) print i_p
$3 = (int *) 0xbfffc98
(gdb) print *i_p
$4 = 1
(gdb) print $4+1
$5 = 2
```

'print func(...)' possible

Value history identifiers

The error occurred at `*(s.i_p) = 2;`

gdb Example (계속)

- backtrace/where: Print the Current Location and a Stack Trace
- set variable: Assign a Value to a Variable
- print: Print the Assigned Value

```
martini:~$ gdb a.out
```

```
... (omitted)
```

```
(gdb) run 1
```

```
...
```

```
Program received signal SIGSEGV, Segmentation fault.
```

```
0x0804846b in foo () at test.c:6
```

```
6      *(s.i_p) = 2;
```

same as 'where'

```
(gdb) backtrace
```

```
#0 0x0804846b in foo () at test.c:6
```

```
#1 0x080484c9 in main (argc=2, argv=0xbffffcb4) at test.c:15
```

```
(gdb) up
```

```
#1 0x080484c9 in main (argc=2, argv=0xbffffcb4) at test.c:15
```

```
15     foo();
```

```
(gdb) set variable (*i_p)++
```

Assignments do not work with core

```
(gdb) print *i_p += 1
```

```
$1 = 3
```

gdb Example (계속)

- break: Set Breakpoints in the Program
- watch: Set “Break-If” Breakpoints
- info breakpoints: Show All Breakpoints and Watchpoints

```
martini:~$ gdb a.out
```

```
... (omitted)
```

```
(gdb) break foo
```

```
Breakpoint 1 at 0x8048466: file test.c, line 6.
```

```
(gdb) break test.c:6
```

```
Note: breakpoint 1 also set at pc 0x8048466.
```

```
Breakpoint 2 at 0x8048466: file test.c, line 6.
```

```
(gdb) run 1
```

```
...
```

```
Breakpoint 1, foo () at test.c:6
```

```
6      *(s.i_p) = 2;
```

```
(gdb) watch *i_p > 0
```

```
Hardware watchpoint 3: *i_p > 0
```

```
(gdb) info breakpoints
```

```
...
```

```
... (omitted)
```

```
5 void foo() {
```

```
6 *(s.i_p) = 2;
```

```
...
```

gdb Example (계속)

- kill: Abort the Running Process
- delete: Delete a Breakpoint or a Watchpoint
- next: Execute the Next Line, Executing a Function

```
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb) delete 2
(gdb) delete 1
(gdb) break 11
Breakpoint 4 at 0x8048495: file test.c, line 11.
(gdb) run 1
Breakpoint 1, main (argc=2, argv=0xbffffcb4) at test.c:12
12    *i_p = atoi(argv[1]);
(gdb) print *i_p
$1 = -1073742712
(gdb) next
13    printf("i = %d\n", *i_p);
(gdb) print *i_p
$2 = 1
```

```
9 ... (omitted)
10 main(...)
    Int *i_p;
...
```

gdb Example (계속)

- step: Execute the Next Line, Stepping into a Function
- continue: Continue Execution

```
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb) run 1
... (omitted)
(gdb) step
13    printf("i = %d\n", *i_p);
(gdb) step
i = 1
15    foo();
(gdb) step
foo () at test.c:6
6     *(s.i_p) = 2;
(gdb) continue
...
```

gdb Example (계속)

- info line: Show Where the Object Code Begins and Ends
- disassemble: Produce a Machine Listing
- info registers: List Integer Registers and Their Contents

```
(gdb) kill
```

```
Kill the program being debugged? (y or n) y
```

```
(gdb) run 1
```

```
... (omitted)
```

```
(gdb) info line 12
```

```
Line 12 of "test.c" starts at address 0x8048495 <main+9>  
and ends at 0x80484af <main+35>.
```

```
(gdb) disassemble main
```

```
...
```

```
(gdb) x 0x80484ca
```

```
0x80484ca <main+62>: 0x895590c3
```

```
(gdb) info registers
```

```
...
```

```
0x80484ca <main+62>: ret
```

awk (Aho Weinberger Kernighan)

- PL for Computing and Data-Manipulation Tasks (Especially, Columns of Data)
 - <http://wiki.kldp.org/wiki.php/Awk>
- Example

```
EXECUTION TIME = 2.0  
EXECUTION TIME = 4.2
```

temp

```
martini:~$ awk '{sum = sum + $4} END {print "avg = " sum/NR}' temp
```

- Result

```
martini:~$ avg = 3.1
```

awk 사용법

- 사용법

- `awk 'program' input-file1 input-file2 ...`
- `awk -f program-file input-file1 input-file2 ...`

```
awk 'BEGIN {sum=0 }{sum = sum + $4} END {print "avg = " sum/NR}' temp
```

```
awk -f avg.awk temp
```

```
#!/bin/awk #  
# BEGIN :  
BEGIN { sum = 0 }  
# ROUTINE :  
{ sum = sum + $4 }  
# END :  
END { print "avg = " sum/NR}
```

awk 구조

- 구조

- 시작(BEGIN) : 입력데이터를 실행하기에 적합한 형태로 바꾸어주는 단계.
- 실행(Routine) : [시작 단계]에서 잘 처리된(정규화된) 데이터를 실제 루틴으로 처리하는 것.
- 끝(END) : [시작 단계]와 마찬가지로, 데이터가 처리된 후에 처리해야 할 내용들을 담음.

```
awk 'BEGIN {sum=0 }{sum = sum + $4} END {print "avg = " sum/NR}' temp
```

awk 내부변수

- FS: Fields Separator
- RS : Records Separator
- NF : Number of Fields
- NR : Number of Records
- FNR: 입력파일이 여러 개인 경우에 현재 파일에서의 NF를 표시
- OFS: Output Fields Separator
- ORS: Output Records Separator

awk 예제

- Symbol table로부터 Break points 만들기
 - <http://kldp.org/node/68354>

```
nm a.out
```

```
$ cat c1.nm | grep -i " t " | grep -v gcc2 | awk '{print "b", $NF}'  
b _init  
b _start  
b call_gmon_start  
b __do_global_dtors_aux  
b frame_dummy  
b before_main  
b before_main_2nd  
b after_main  
b main  
b __do_global_ctors_aux  
b _fini
```

```
w _Jv_RegisterClasses  
w __deregister_frame_info_bases  
w __gmon_start_  
U __libc_start_main@@GLIBC_2.0  
w __register_frame_info_bases  
U printf@@GLIBC_2.0  
08048230 T _init  
08048280 T _start  
080482a4 t call_gmon_start  
080482a4 t gcc2_compiled.  
080482d0 t __do_global_dtors_aux  
08048330 t frame_dummy  
08048394 T before_main  
080483a8 T before_main_2nd  
080483bc T after_main  
080483d0 T main  
08048400 t __do_global_ctors_aux  
08048430 T _fini  
08048430 t gcc2_compiled.  
08048460 R _fp_hw  
...
```

awk 예제

- Regular expression
- http://www.delorie.com/gnu/docs/gawk/gawk_19.html

```
% awk '/foo/ { print $0 }' BBS-list
% fooy 555-1234 2400/1200/300 B
% foot 555-6699 1200/300 B
% macfoo 555-6480 1200/300 A
% sabafoo 555-2127 1200/300 C
```

Regular Expression 간단한 예

- <http://ko.wikipedia.org/wiki/> 출처
- 예: "(fa | mo | br?o)ther"는 "father", "mother", "brother", "bother"를 나타낸다.

	여러 식 중에서 하나를 선택한다. 예를 들어, "abc adc"는 abc라는 문자열과 adc라는 문자열을 모두 포함한다.
()	여러 식을 하나로 묶을 수 있다. "abc adc"와 "a(b d)c"는 같은 의미를 가진다.
*	0개 이상. "a*b"는 "b", "ab", "aab", "aaab"를 포함한다.
+	1개 이상. "a+b"는 "ab", "aab", "aaab"를 포함하지만 "b"는 포함하지 않는다.
?	0개 또는 1개. "a?b"는 "b", "ab"를 포함한다.
{m, n}	m개 이상 n개 미만. "a{1,3}b"는 "ab", "aab", "aaab"를 포함하지만, "b"나 "aaaab"는 포함하지 않는다.
[]	: "["과 "]" 사이의 문자 중 하나를 선택한다. 예를 들면 [abc]d는 ad, bd, cd를 뜻한다. 또한, "-" 기호와 함께 쓰면 문자의 범위를 지정할 수 있다. "[a-z]"는 a부터 z까지 중 하나, "[1-9]"는 1부터 9까지 중의 하나를 뜻한다.
[^]	"[^"과 "]" 사이의 문자를 제외한 나머지 하나를 선택한다. 예를 들면 [^abc]d는 ad, bd, cd는 포함하지 않고 ed, fd 등을 포함한다. [^a-z]는 알파벳 소문자로 시작하지 않는 모든 문자를 나타낸다.
^, \$	각각 문자열의 처음과 끝을 나타낸다. "^\$"는 빈 줄과 일치한다.

Make

순서

- make
 - 기본
 - 용어
 - Examples
- Q&A

make 기본

- 정의
 - PL for Automating Large Compilation
 - Permit reducing the compilation time
 - Permit guaranteeing that programs are compiled and linked correctly
- Idea
 - To Recompile the Source File Only If Necessary

```
martini:~$ gcc main.c func.c
```

if modified

- To Avoid Recompiling It If a Current Object File Already Exists

```
martini:~$ gcc main.c func.o
```

if current

make 용어

- Target
 - Task That Needs to Be Performed
 - e.g., name of a file to create
- Dependency
 - Relationship between Two Targets
 - e.g., change dependency
- Up to Date
 - File More Recent than Any of the Files on Which It Depends
 - e.g., up-to-date object file
- makefile
 - File Describing How to Create Targets

makefile Example

- Set of Instructions about How to Build Targets

```
# makefile
# targets beginning at the left margin, followed by :
test:
# commands to create test, beginning with [tab]
    gcc -o test main.c func.c
test.debug:
    gcc -g -o test.debug main.c func.c
```

Comments

- Invocation

```
martini:~$ make test
martini:~$ make test.debug
martini:~$ make
```

make the first target, test

make Examples

- Recursive Checking of Dependencies
 - Until All Files Needed to Generate a Target Become Up to Date

```
test: main.o func.o
    gcc -o test main.o func.o
main.o: main.c
    gcc -c main.c
func.o: func.c
    gcc -c func.c
```

- Result

```
martini:~$ make test
gcc -c main.c
gcc -c func.c
gcc -o test main.o func.o
```

make Examples (계속)

```
func.do: func.c
    gcc -o func.do -c -g func.c
main.do: main.c
    gcc -o main.do -c -g main.c
test.debug: main.do func.do
    gcc -o test.debug main.do func.do
clean:
    rm *.o *.do test test.debug
```

- Result

```
martini:~$ make test.debug
gcc -o main.do -c -g main.c
gcc -o func.do -c -g func.c
gcc -o test.debug main.do func.do
martini:~$ make clean
rm *.o *.do test test.debug
```

Abbreviations, Macros, & More

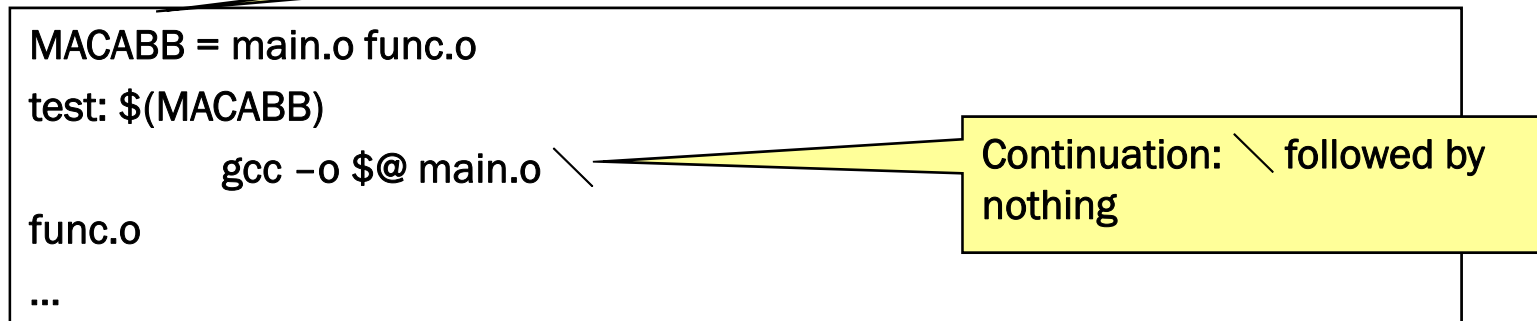
- Abbreviations

- $\$@$: Target Name
- $\$*$: Target Name w/o the Suffix



- Macros

Macro Abbreviation



Default Compilation Rules - Efficient Ways for Building Targets

- Two Methods to Define Defaults
 - Suffix Rules

```
# clear the list of suffixes
.SUFFIXES:
# specify suffix rules for .c and .o
.SUFFIXES: .c .o
.c.o:: gcc -c -o $@ $*.c
test: main.o func.o
    gcc -o test main.o func.o
main.o: main.c
func.o: func.c
```

\$<

Wildcard

- Pattern Rules

```
%o : %c
    gcc -c $<
```