Operating Systems Lab: Tools

- GDB, ctags, cscope -

KAIST

Youjip Won

Contents

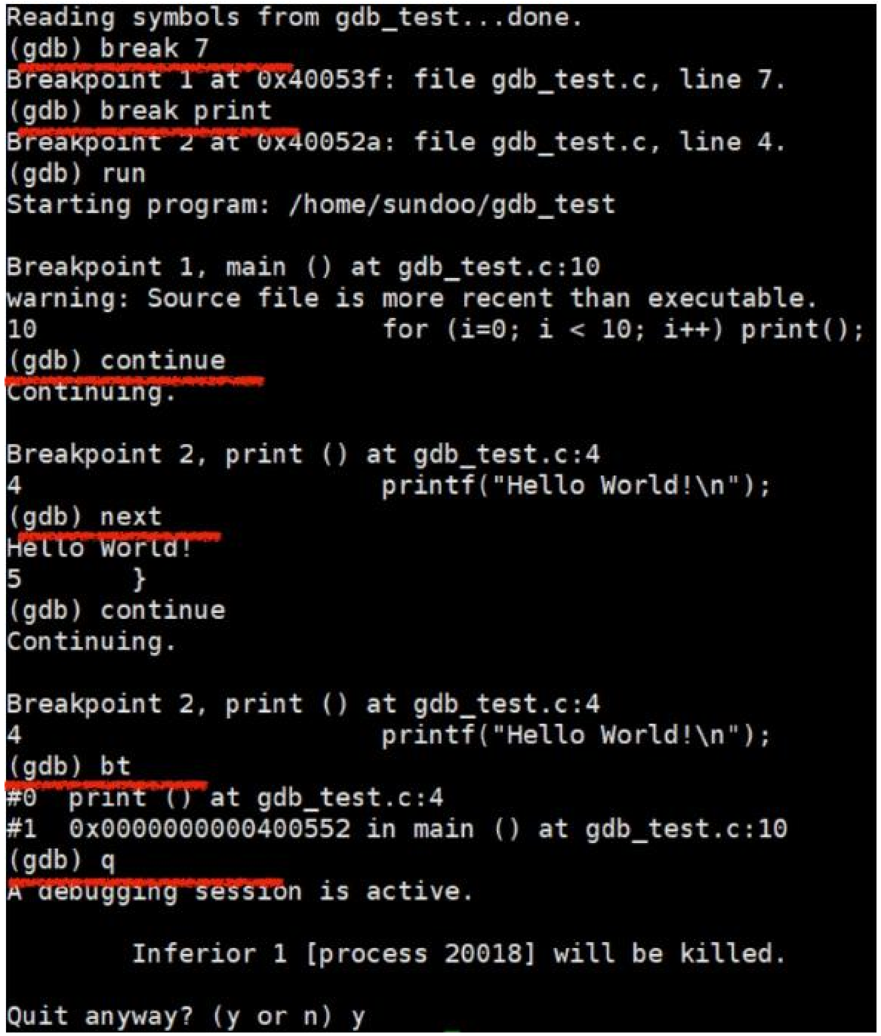
GDB

- What is GDB?
  - Command line debugging tool made by GNU
  - It provides many functions which can trace the internal behaviors of the computer p rogram while it is executed.
- Compile for GDB
  - `$ gcc -g -o gdb_test gdb_test.c`
- Run GDB
  - `$ gdb [file]`
- Pass argument to GDB
  - `$ gdb [file]`
  - `(gdb) set args argument`

GDB (Cont.)

gdb_test.c

```
1   #include <stdio.h>
2
3   void print() {
4         printf("Hello World!\n");
5   }
6
7   int main(void) {
8
9         int i;
10        for(i = 0; i < 10; i++) print();
11        return 0;
12  }
```



```
Reading symbols from gdb_test...done.
(gdb) break 7
Breakpoint 1 at 0x40053f: file gdb_test.c, line 7.
(gdb) break print
Breakpoint 2 at 0x40052a: file gdb_test.c, line 4.
(gdb) run
Starting program: /home/sundoo/gdb_test

Breakpoint 1, main () at gdb_test.c:10
warning: Source file is more recent than executable.
10                      for (i=0; i < 10; i++) print();
(gdb) continue
Continuing.

Breakpoint 2, print () at gdb_test.c:4
4                       printf("Hello World!\n");
(gdb) next
Hello World!
5         }
(gdb) continue
Continuing.

Breakpoint 2, print () at gdb_test.c:4
4                       printf("Hello World!\n");
(gdb) bt
#0  print () at gdb_test.c:4
#1  0x0000000000400552 in main () at gdb_test.c:10
(gdb) q
A debugging session is active.

        Inferior 1 [process 20018] will be killed.

Quit anyway? (y or n) y
```

GDB commands

- **continue(c)**: Run program continuously.
- **step(s)**: Run the code line by line. If cursor is on a function, GDB enters the function.
- **next(n)**: Run the code line by line. If cursor is on a function, GDB runs the function and move to next line.
- **finish**: Run the function to the end and stop.
- **return [value]**: Cancel the running function and return with [value].
- **list**: print the source of the running part.
  - It prints main function before program run.
- **list [number]**: Print [number] line.
- **list [function]**: Print source code of [function]
- **set listsize [n]**: Set line size of list. Default is 10

GDB commands (Cont.)

- Commands for print: check the state of program
  - **whatis [var]**: Print type of variable.
  - **print(p) [var]**: Print value of variable.
    - print a->member
    - print add(1,2)
    - print /x value // you can set the print type using x, u, o, c

GDB commands (Cont.)

- Commands for break: Pause the process at the location you want
  - **break(b) [number]**: Pause the process at [number] line.
  - **break(b) [function]**: Pause the process at the [function].
  - **break(b) [file:function]**: Pause the process at the [function] in [file].
  - **break(b) [file:number]**: Pause the process at [number] line in [file].
  - **info break**: Print state of break point
  - **delete [number]**: Delete break point of [number]. If any number is not set, delete al l break point.
- Command for call stack(history of calling)
  - **backtrace(bt)**: Print all called functions until this functions is called.
  - **backtrace(bt) [number]**: Print [number] lines of bt command result.

GDB commands (Cont.)

- Stack frame
  - Call stack consists of stack frames(or frames).
  - Stack frame contains arguments, local variables and return address of function.
  - commands
    - **frame [number]**: Select a frame of [number], print name of the selected function.
    - **select-frame [number]**: Select a frame of [number], don't print name of the selected functi on.
    - **info frame**: Print the stored data of the selected frame.

Sample Script - gdb (1)

- Run pintos with gdb (`--qemu` can be omitted if you use `bochs`)

  ```
  % cd pintos/src/threads
  % make
  % cd build
  % pintos --gdb --qemu -- run alarm-multiple
  ```

  Now pintos is waiting connection from gdb

- Run gdb and connect to pintos

  Open another terminal and follow next steps in the terminal

  ```
  % cd pintos/src/threads/build
  % gdb kernel.o
  ```

  Now gdb shell is opened

  ```
  (gdb) target remote localhost:1234
  ```

  Now gdb is connected to pintos, and pintos is waiting execution order of the gdb

Sample Script - gdb (2)

- Debug `main()` function in pintos

  In the gdb terminal,

  ```
  (gdb) break main
  (gdb) info breakpoints
  (gdb) continue
  ```

  Pintos will be executed, stop at start of main function, and gdb shell will be available

  ```
  (gdb) delete 1
  (gdb) list
  (gdb) list thread_init
  (gdb) next
  (gdb) next
  ```

  Now gdb is on code calling `read_command_line().`

  Let's go to inside of `read_command_line()`

  ```
  (gdb) step
  ```

  Now gdb came inside of `read_command_line()`

  ```
  (gdb) next
  ```

  Now a variable `argc` has been set. Let's figure value in `argc` out

  ```
  (gdb) print argc
  ```

  It will print "`$1=2`". So value in `argc` is `2`

Sample Script - gdb (3)

- Debug `main()` function in pintos (Continue…)

  Now, gdb is on code "`argv[i] = p`".

  Let's figure variable `p` out.

  ```
  (gdb) print p
  ```

  It will print `$2 = 0xc0007d3e "run"`.

  It means that `p` is pointer including `0xc0007d3e` where have string "`run`"

  Now, there is no stuff to debug in `read_command_line()`.

  Let's go back to outside.

  ```
  (gdb) finish
  ```

  Now gdb is on return point of `read_command_line()` at `main()`.

  We don't need to execute each code with next command to return function

ctags

- The tool for making tag file which points location of function, variable, string, etc. of a s ource file.
- "tags" file: dictionary that contain all variables and function names and the associated lo cations.
- Create a tag file.

ctags [option] [file(s)] / etags [option] [file(s)]

  - -R: Scan all subdirectory recursively.
  - --exclude=[pattern]: Exclude files and directories which have 'pattern' in name from creatin g tag file.
  - -f [file]: Create tag file with 'file'. If 'file' is '-', tag file has a name with 'tags(case of ctags)', ' TAGS(case of etags)' defaultly.
  - --list-languages: Print language list supporting tag file create.
  - -x: Print tags as table to stdout without creating tag file.

ctags command in vim

Keyboard command	Action
Ctrl-]	Jump to the tag underneath the cursor
:ts <tag> <RET>	Search for a particular tag
:tn	Go to the next definition for the last tag
:tp	Go to the previous definition for the last tag
:ts	List all of the definitions of the last tag
Ctrl-t	Jump back up in the tag stack

Sample Script

- Install ctags (in Linux)

```
%sudo apt-get install ctags
```

- Create tag file

```
%cd pintos/src
%ctags -R
%ls tags
```

- vim and ctag

```
% cd pintos/src
%vim tags
```

  - Set tag file from the command mode

```
set tags=./tags
```

- Basic command
  - In command mode, type "`tj main`" → list the path, filename and line number of `main()` fu nctions
  - Move to main() pintos (line 35, threads/init.c)
  - Locate the cursor at function bss_init at main() and type `Ctrl-]`.
  - Locate the cursor at memset and type `Ctrl-]`
  - `tp` from command mode: goes to previous location.

cscope

- The tool for accessing object like ctags, but with more powerful features.
- Find the functions that "call" a given function or that "is called" by a given function.
  - Find all functions that call malloc()
  - Find all functions that are called by malloc().

```
cs find <Command Character> <String>
```

Command	Description
s	Find C symbol `String`
g	Find definition `String`
d	Find functions called by function `String`
c	Find functions calling function `String`
t	Find text `String`
e	Find egrep pattern `String`
f	Find file `String`
i	Find files #including file `String`

Sample Script

- Install `cscope`

```
% sudo apt-get install cscope
```

- Create tag file

```
% cd pintos/src
% find ./ -name “*.[chS]” > cscope.files
% cscope –i cscope.files
% ls cscope.out
```

- Use cscope in vim
  - Open the cscope.out file from vim and register the cscope.out for source code navigation.

```
% cd pintos/src
% vim cscope.out
```

  - Register scope file from the command mode of vim

```
cs add ./cscope.out
```

- Basic command
  - In command mode, type “`cs find c memset`” → list the path, filename and line number of fu nctions calling `memset()`
  - In command mode, type “`cs find g main`” → list the path, filename and line number of `mai n()` functions
  - In command mode, type “`cs find e shut*`” → list the path, filename and line number of cod e including string of “`shut*`”