CS 238P
Operating Systems
Discussion 3

Today’s agenda

- Basic GDB
- Solving homework 2

GDB debugger

- Control the execution flow of the program (stop/resume)
- View/modify the system status (register, memory contents, …)
- Run the target(inferior) inside gdb or attach to the running process
- Remote debugging
- For Mac OS users: use lldb

How to use

- Use -g flag when compiling

gcc -g -o sh ./sh.c

- To start:

gdb ./EXECUTABLE_NAME ARG1 ARG2 …

Example: gdb ./sh

GNU Debugger(GDB)

- Check debug information
  - l (or list)

```
list
list <filename>:<function>
list <filename>:<line_number>
```

```
(gdb) l
1       #include <stdio.h>
2
3       int main()
4       {
5       char str[2][3] = {0,};
6       printf ("%p\n", str);
7       printf ("%p\n", &str[0]);
8       printf ("%p\n", &str[1]);
9       printf ("%p\n", &str[1][0]);
10      printf ("%p\n", &str[2]);
(gdb) list
11      printf ("%p\n", &str[2][0]);
12      printf ("%p\n", &str[2][1]);
13      return 0;
14      }
15
16
```

GNU Debugger(GDB)

- breakpoint: stop the program at certain point
  - where?
    - a line of the source code
    - or at specific memory address
- info b: list breakpoints
- delete <num>

```
(gdb) break 5
Breakpoint 1 at 0x400525: file test.c, line 5.
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400525 in main at test.c:5
(gdb) delete 1
(gdb) info b
No breakpoints or watchpoints.
(gdb) break 5
Breakpoint 2 at 0x400525: file test.c, line 5.
(gdb) run
Starting program: /home/saehansy/Workspace/ics143a/FQ19/test.exe

Breakpoint 2, main () at test.c:5
5       char str[2][3] = {0,};
```

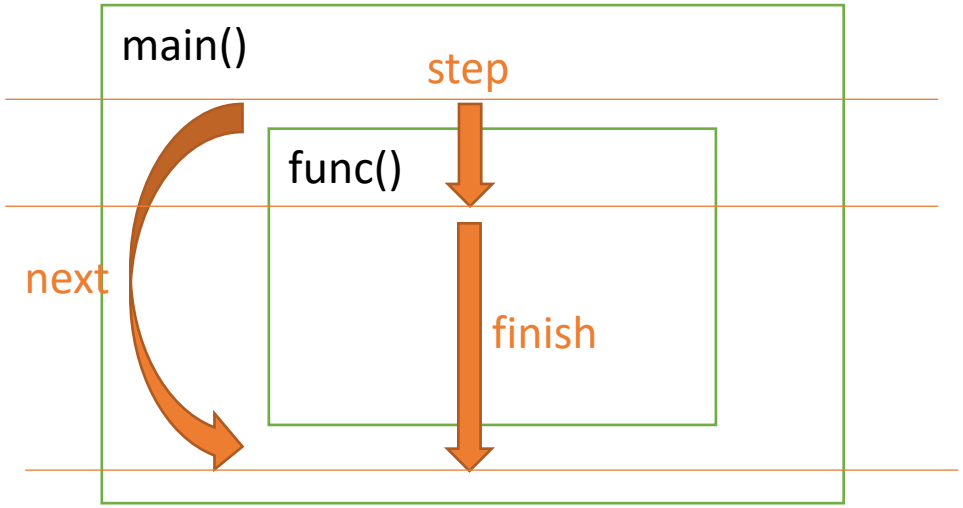
GNU Debugger(GDB)

- run & continue
  - **run**: run the program. If there's no breakpoint, the program will run until the end as if there is no gdb
  - **continue**: when program stopped at some breakpoint, *continue* will make the program run until the next breakpoint; otherwise, no further breakpoint, it run until the end

GNU Debugger(GDB)

- next, step in & out
  - step over: execute one line (gdb command: next)
  - step in: execute one line & go inside the function (gdb command: step)
  - step out: skip the rest of the current function (gdb command: finish)

```
(gdb) step
step      stepi     stepping
(gdb) stepi
0x000000000040052c      5       char str[2][3] = {0,};
(gdb)
6       printf ("%p\n", str);
(gdb)
0x0000000000400536      6       printf ("%p\n", str);
(gdb)
0x0000000000400539      6       printf ("%p\n", str);
```

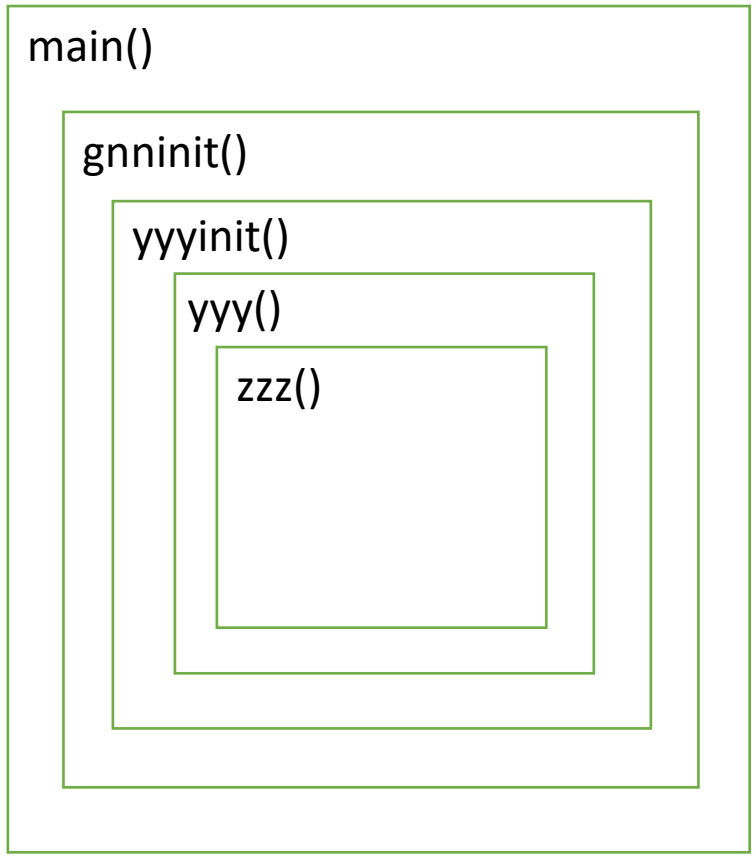


- execute one instruction: stepi, nexti

GNU Debugger(GDB)

- bt (or backtrace): shows the *call stack*

```
(gdb) bt
#0 zzz () at zzz.c:96
#1 0xf7d39cba in yyy (arg=arg@entry=0x0) at
yyy.c:542
#2 0xf7d3a4f6 in yyyinit () at yyy.c:590
#3 0x0804ac0c in gnninit () at gnn.c:374
#4 main (argc=1, argv=0xffffd5e4) at gnn.c:389
```



GNU Debugger(GDB)

- info & help
  - **info reg**
  - **info frame**

```
(gdb) info reg
rax            0x7fffffffdbe0   140737488346080
rbx            0x0      0
rcx            0x4005f0 4195824
rdx            0x7fffffffdce8   140737488346344
rsi            0x7fffffffdbe0   140737488346080
rdi            0x1      1
rbp            0x7fffffffdbf0   0x7fffffffdbf0
rsp            0x7fffffffdbe0   0x7fffffffdbe0
r8             0x7ffff7dd5e80   140737351868032
r9             0x0      0
r10            0x7fffffffd880   140737488345216
r11            0x7ffff7a302e0   140737348043488
r12            0x400430 4195376
r13            0x7fffffffdcd0   140737488346320
r14            0x0      0
r15            0x0      0
rip            0x400539 0x400539 <main+28>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

```
(gdb) info
address                 copying                 inferiors
all-registers           dcache                  line
args                    display                 locals
auto-load               extensions              macro
auxv                    files                   macros
bookmarks               float                   mem
breakpoints             frame                   os
checkpoints             frame-filter            pretty-printer
classes                 functions               probes
common                  handle                  proc
```

```
(gdb) help stepping
Specify single-stepping behavior at a tracepoint.
Argument is number of instructions to trace in single-step mode
following the tracepoint.  This command is normally followed by
one or more "collect" commands, to specify what to collect
while single-stepping.
```

```
(gdb) info frame
Stack level 0, frame at 0x7fffffffdc00:
 rip = 0x400539 in main (test.c:6); saved rip 0x7ffff7a303d5
 source language c.
 Arglist at 0x7fffffffdbf0, args:
 Locals at 0x7fffffffdbf0, Previous frame's sp is 0x7fffffffdc00
 Saved registers:
  rbp at 0x7fffffffdbf0, rip at 0x7fffffffdbf8
```

GNU Debugger(GDB)

- breakpoints using address
  - b *0x4005b4
  - For addresses, use * in front of it
- Useful print command
  - **p (or print)** <var_name> or *<address> or $registers
  - **x/[NUM][FMT] $sp**: show stack memory; FMT can be x(hex) f(float), …

prints 10 words in hexadecimal above the stack pointer($sp)

```
(gdb) x/10x $sp
0xffeac63c: 0xf7d39cba 0xf7d3c0d8 0xf7d3c21b 0x00000001
0xffeac64c: 0xf78d133f 0xffeac6f4 0xf7a14450 0xffeac678
0xffeac65c: 0x00000000 0xf7d3790e
```

GNU Debugger(GDB)

- Debugging assembly
  - **objdump –D <exec>:** human-readable dump of instructions of a program
  - **objdump –D exec_file > result.txt; vi result.txt**
- Additional windows(helpful)
  - In some systems, **tui enable** – **layout asm** – **tui disable**
  - or **tui reg general** – **layout asm**
  - To turn it off, C-x a(or C-x C-a, no need to lift the control key up)

GNU Debugger(GDB)

- For more information, search for “GDB cheatsheet”
  - https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

Solving homework 2 (exec)

- Please use execvp(const char **file*, char *const *argv*[])
- v - stands for argV (accept arguments as an array)
- p - stands for Path (include search in a $PATH variable)

Solving homework 2 (exec)

- Don’t forget to use fork & wait
- Do fork and then do exec in the child(!) process
- Do wait in the parent process to wait until children would finish

Solving homework 2 (pipes)

- You need to do 2 fork here (one for left part and one for right part)
- In the children close input/ouput, duplicate read/write of a pipe, close BOTH sides of pipe
- In then parent close pipe, do wait for children