EE488: System Software Design
Fall 2022

GDB, ctags, cscope and make

Youjip Won

KAIST

gdb

GDB, the tool

- What is GDB?
  - Default debugger for GNU software system
  - GDB can trace execution of program and support other functionalities.
- How to compile file that can be debugged by gdb?
  - `$ gcc -g gdb_test.c -o gdb_test`
- Run gdb
  - `$ gdb [FILENAME]`
- How to pass arguments to program in gdb?
  - `$ gdb [FILENAME]`
  - `(gdb) set args [argument]`

GDB - basic commands

- `(gdb) continue or c`
  - Continue execution of program
- `(gdb) step or s`
  - Run a line of code in program. If the code calls function, it enter to inside of function
- `(gdb) next or n`
  - Run a line of code in program. If the code calls function, it not enter to inside of function but execu te next line.
- `(gdb) finish`
  - Continue execution of program until current function returns
- `(gdb) return value`
  - Stop the execution of current function, and use value as return value

GDB - basic commands (Cont.)

- `(gdb) list 90`
  - Print source code at 90 line
- `(gdb) list badfunc`
  - Print the source code of specific function that you specified
- `(gdb) set listsize n`
  - Set the number of lines that is printed when the list command is excuted
  - default is 10

GDB - print

- When you want to examine some variables
  - `(gdb) whatis [variable]` : Print the type of variable
  - `(gdb) print [variable]` : Print the value of variable

- Print
  - `(gdb) print a->member`
  - `(gdb) print add(1,2)`
  - `(gdb) print /x value`
  - `(gdb) print p->state`
  - `(gdb) print p->pid`
  - `(gdb) print (*p)`

  - You can specify the format of output as use x, u, o, c keyword

GDB - break

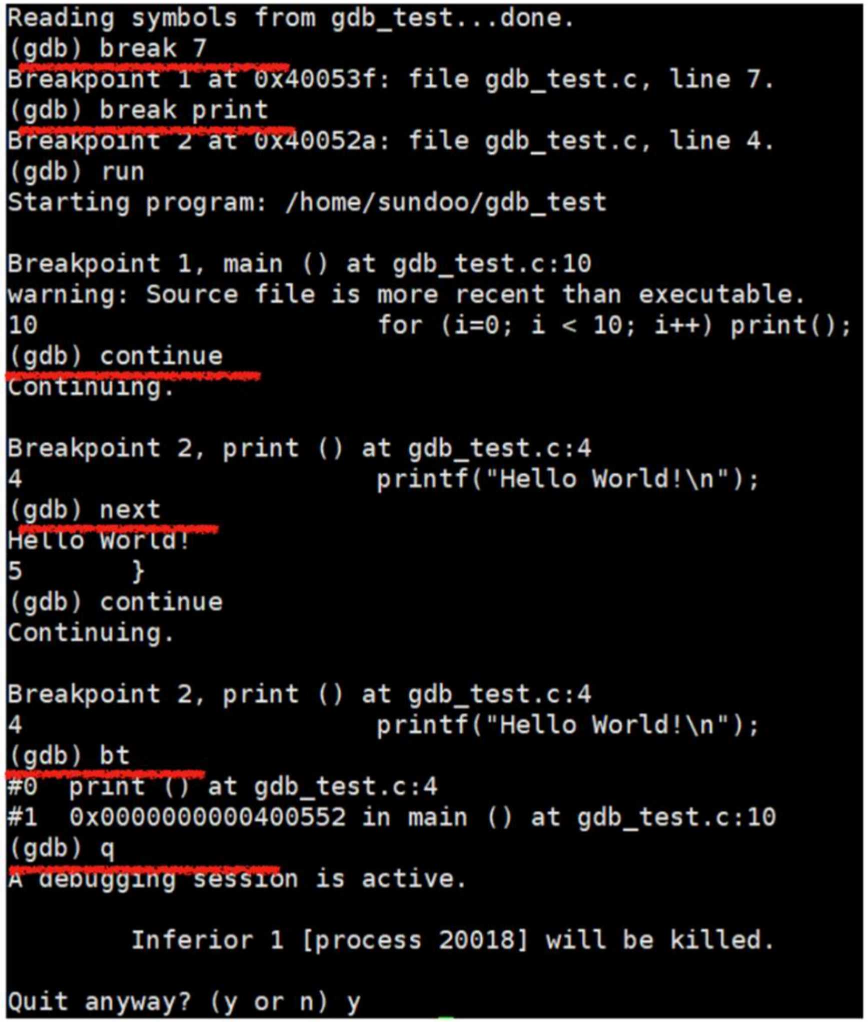
- When you want to stop execution of program at position you want
  - `(gdb) break 31`
  - `(gdb) break func`
  - `(gdb) break hello.c:main`
  - `(gdb) break utilc:300`
  - `(gdb) info break`
  - `(gdb) delete 1` : (If you not specify number, all breakpoints are deleted)

- Print the backtrace information in current function
  - `(gdb) backtrace`
  - `(gdb) backtrace n`

gdb_test.c

```c
#include <stdio.h>

void print() {
    printf(“hello world!\n”);
}

int main() {

   int i;
   for (i=0; i<10; i++) print();
   return 0;
}
```

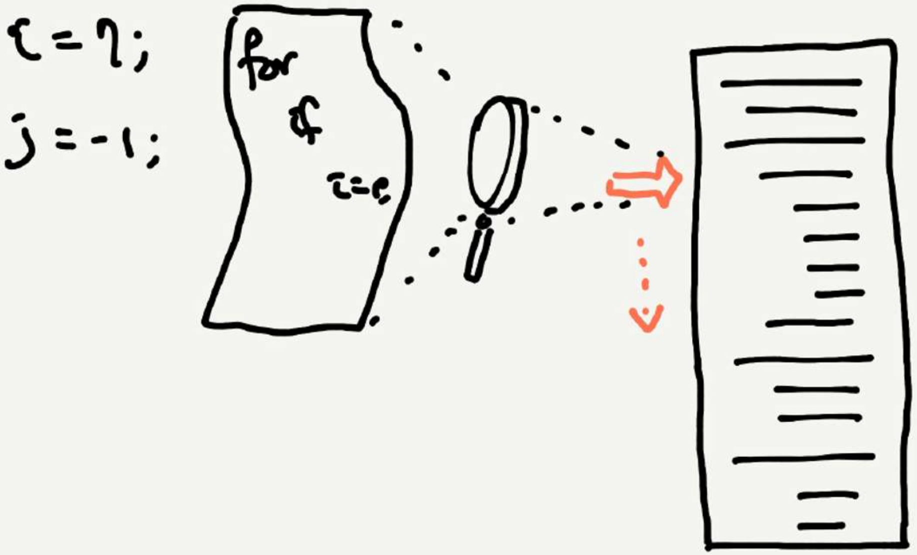
GDB(Cont.)

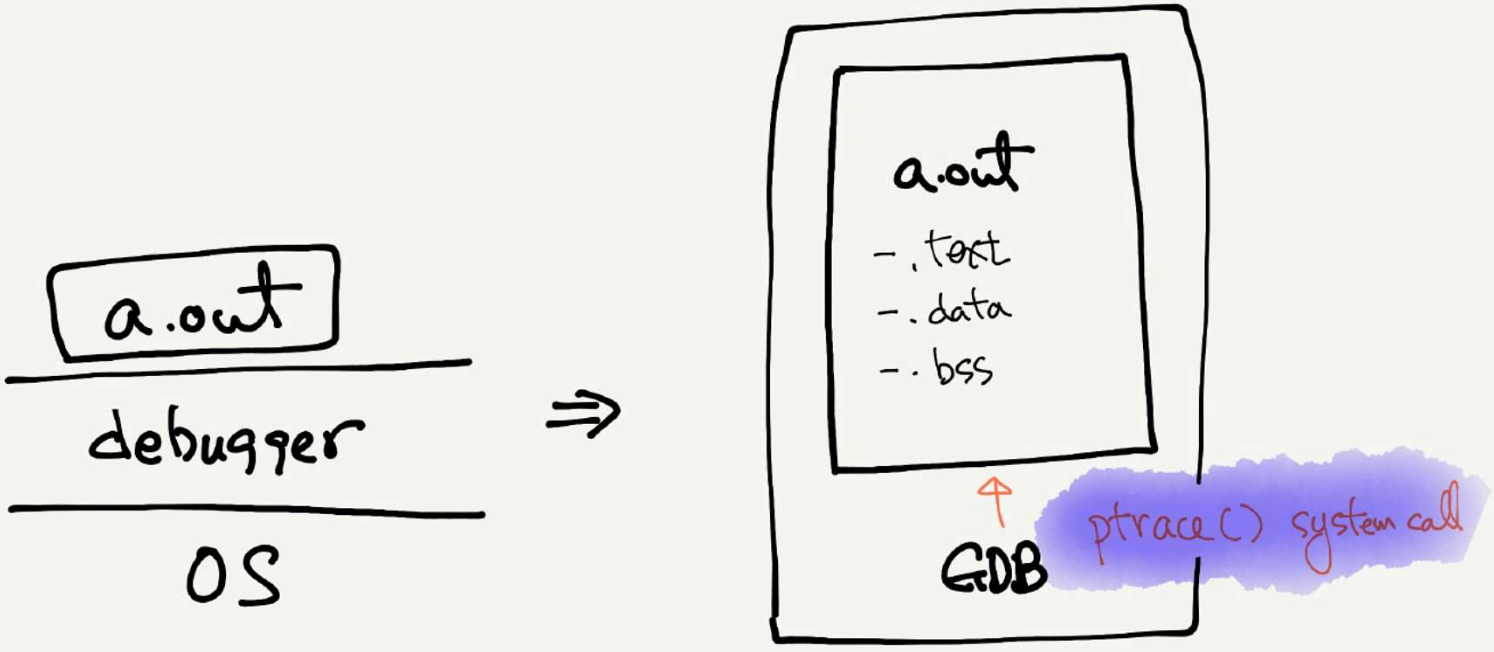


```
Reading symbols from gdb_test...done.
(gdb) break 7
Breakpoint 1 at 0x40053f: file gdb_test.c, line 7.
(gdb) break print
Breakpoint 2 at 0x40052a: file gdb_test.c, line 4.
(gdb) run
Starting program: /home/sundoo/gdb_test

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
```

Debugging

While the program is running, examine the program behavior.

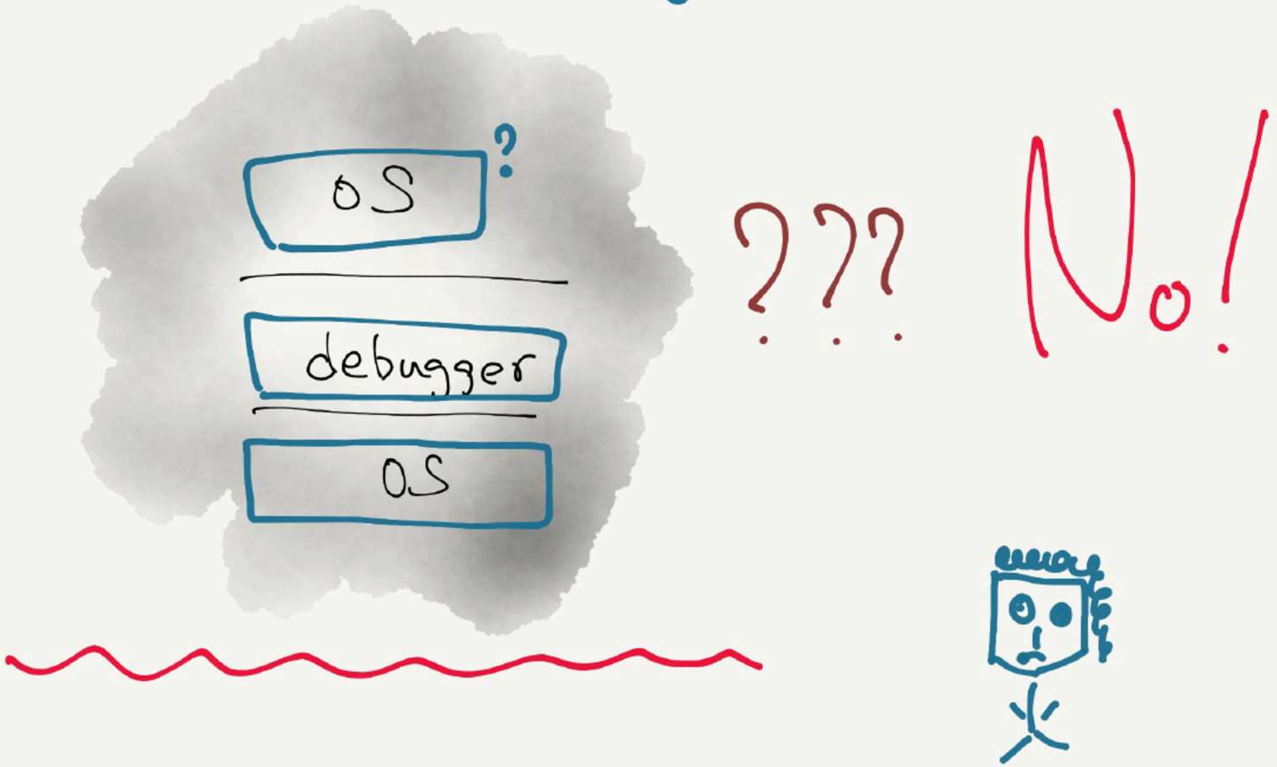


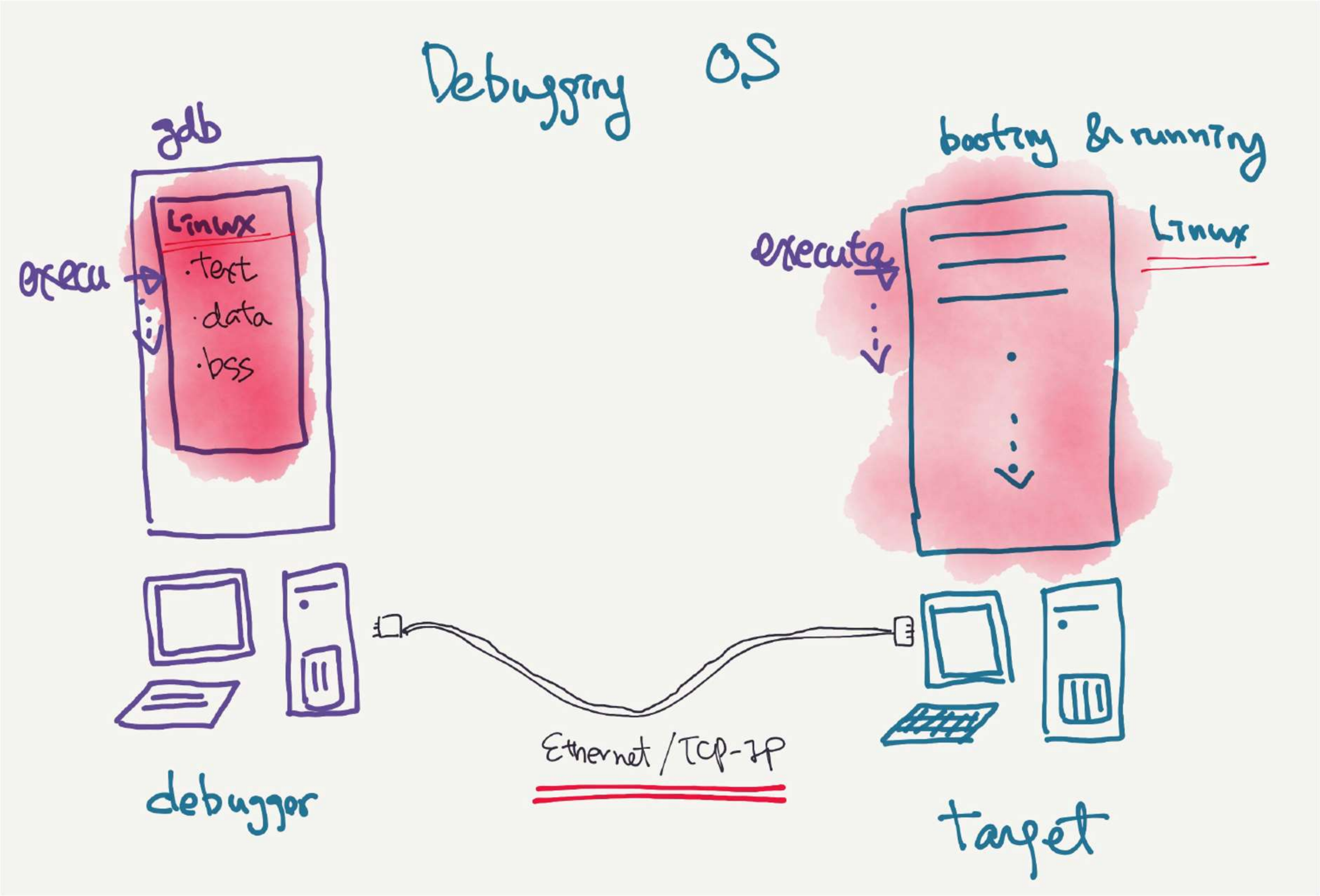


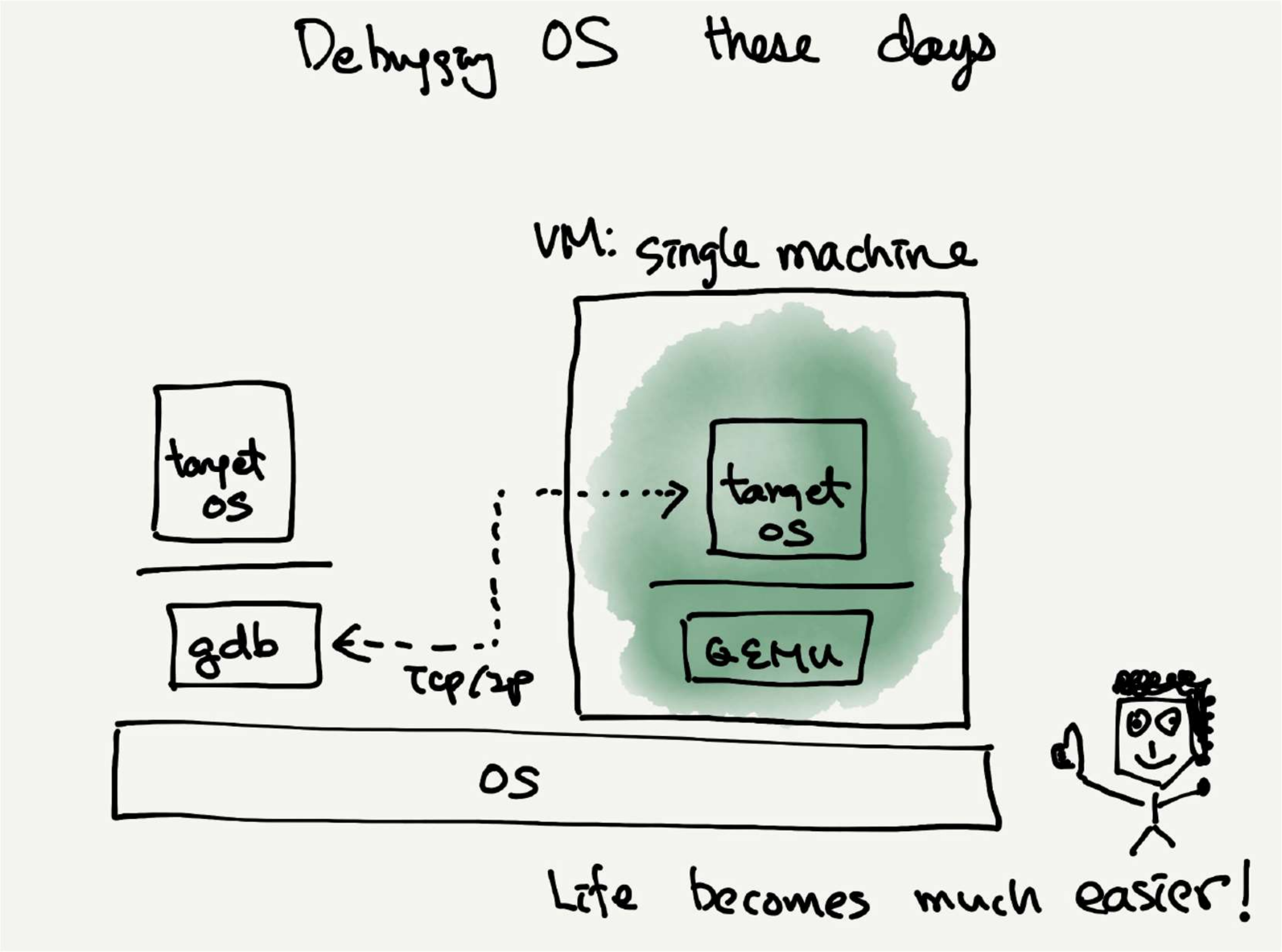
ptrace system call: a process controls and examines the other process

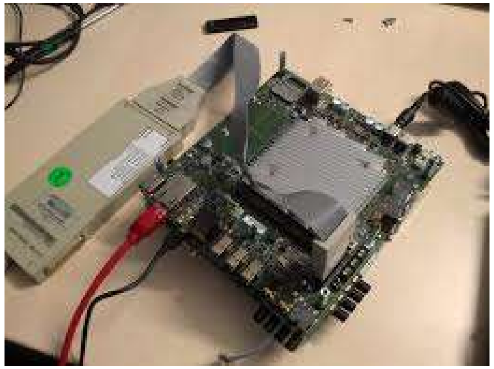
How to debug OS?

We need to debug OS!!!




Debugging OS
gdb
Linux
.text
.data
.bss
execu
booting & running
execute
Linux
Ethernet / TCP-IP
debugger
target


Debugging OS these days
VM: single machine
target OS
target OS
QEMU
gdb
TCP/IP
OS
Life becomes much easier!



normal booting & debug mode booting

- make qemu

```
qemu-system-i386 -serial mon:stdio -drive
file=fs.img,index=1,media=disk,format=raw -drive
file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
```

- make qemu-gdb

```
qemu-system-i386 -serial mon:stdio -drive
 file=fs.img,index=1,media=disk,format=raw -drive
 file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512  -S
-gdb tcp::25501
```

- Difference between two upper command is “-S -gdb tcp::25501”
  - `-S` : suspend the debug target just before the booting starts
  - `-gdb tcp::[port]`
    - port number that is used to accept the command from gdb.

First, run the debug target

```
$ qemu-system-i386 -serial mon:stdio -drive file=fs.img, index=1,media=disk,
format=raw -drive file=xv6.img, index=0,media=disk,format=raw -smp 2 -m 512 -S
-gdb tcp::25501
```

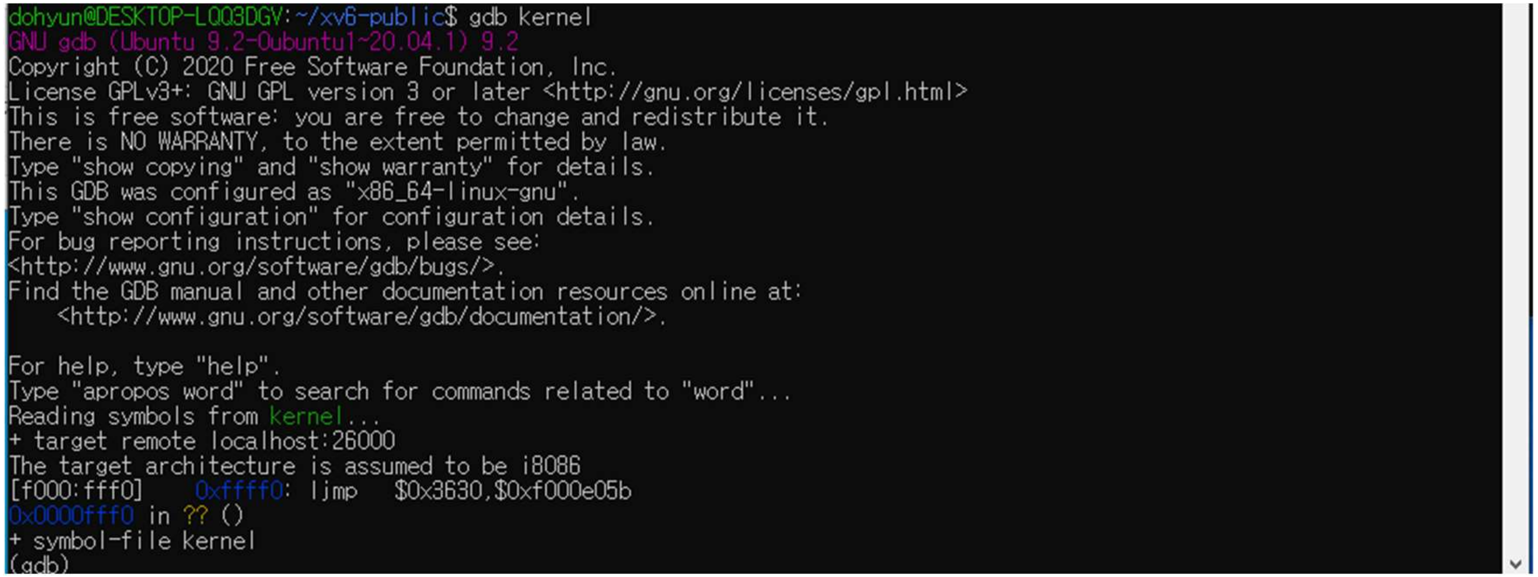
- execute upper command from the shell.
- QEMU stops and waits for the gdb command.

Second, run the debugger

- Open a new terminal.
- Go to the directory where kernel binary resides.
- Execute gdb with loading the target binary.

```
$ gdb [binary file to load]
```

```
$ gdb kernel                    // linux & WSL
$ i386-elf-gdb kernel           // macOS
```



```
dohyun@DESKTOP-LQQ3DGV:~/xv6-public$ gdb kernel
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...
+ target remote localhost:26000
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
(gdb)
```

Output of WSL

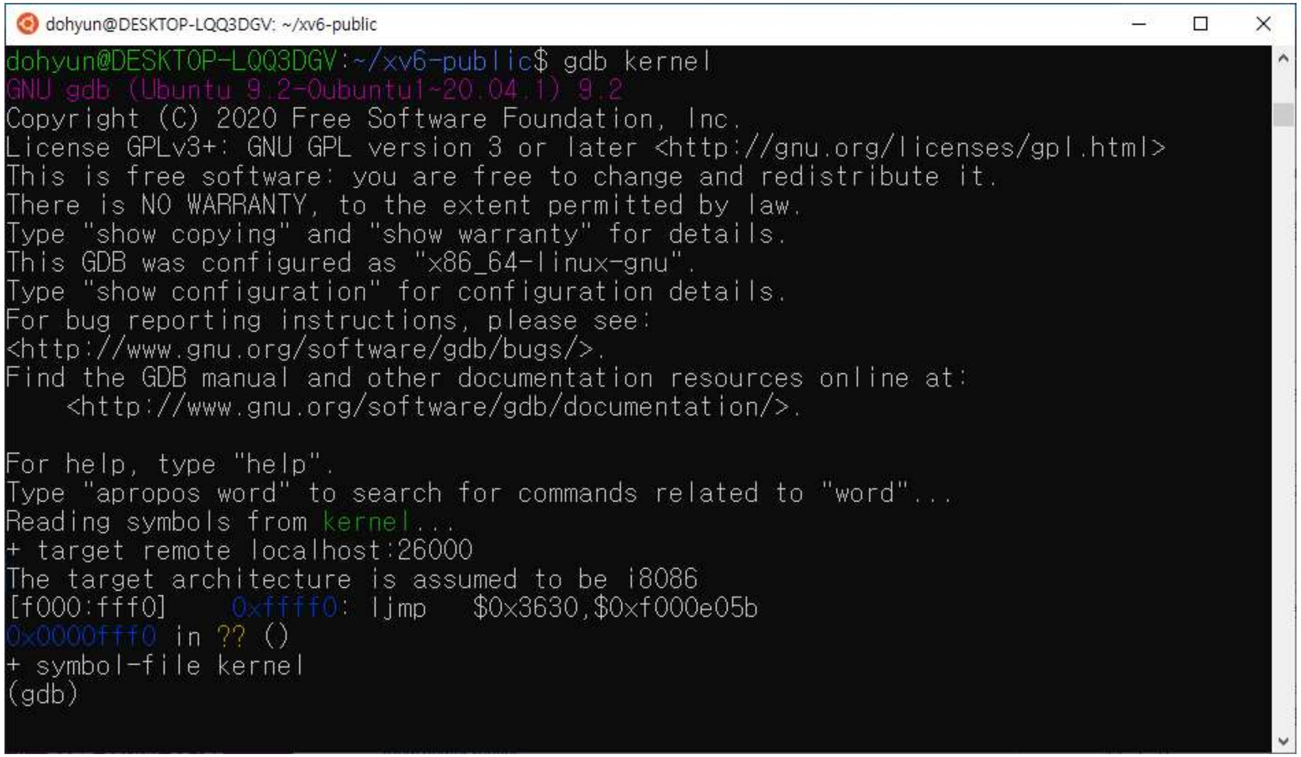
.gdbinit

- GDB reads the `.gdbinit` first and executes the commands in this file.
- `.gdbinit` helps to automatically execute the commands you need to execute each time.

```
target remote localhost:26000
```

- Git repository of xv6-public already has `.gdbinit` file.
- If gdb is not connected to QEMU automatically, please check whether there is the `.gdbinit` file or not.

connect the gdb to debug target



dohyun@DESKTOP-LQQ3DGV: ~/xv6-public

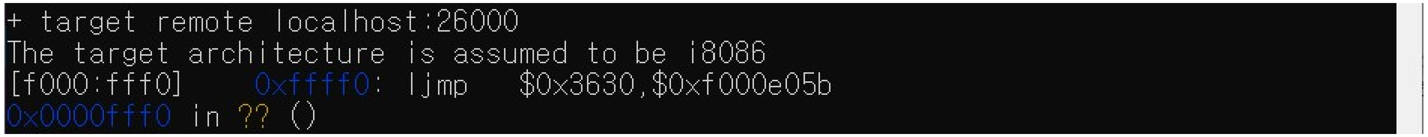
```
dohyun@DESKTOP-LQQ3DGV:~/xv6-public$ gdb kernel
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel...
+ target remote localhost:26000
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
(gdb)
```

- Load the kernel to gdb.
- It loads the program information and the symbols.
- It was not connected to the actual debug target.

connect to the debug target

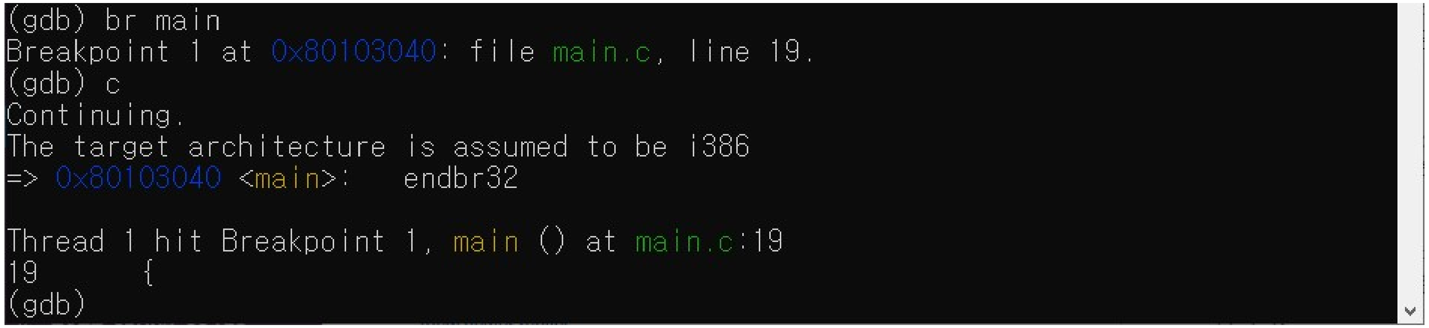
```
(gdb) target remote localhost:[port]
```

- Specify the port number with the one you



```
+ target remote localhost:26000
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
```

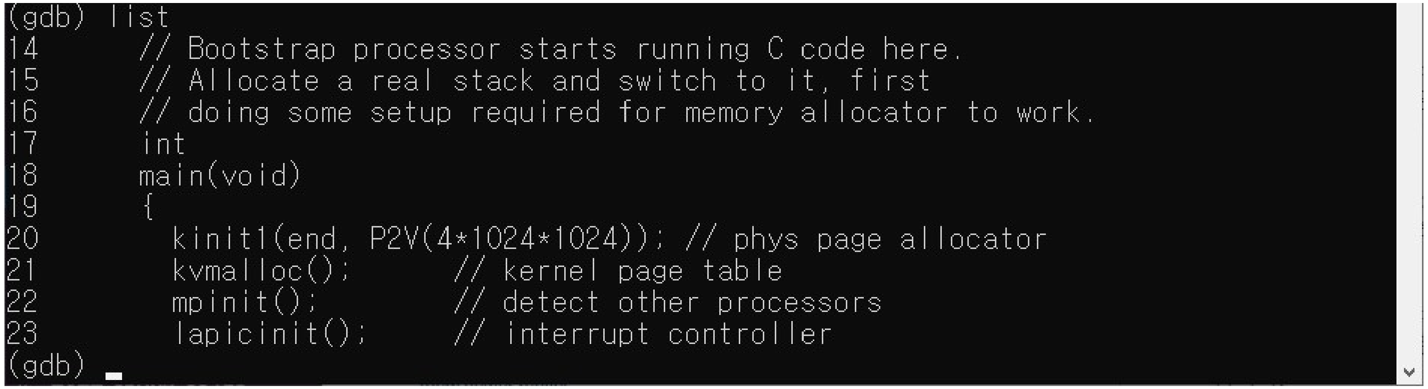
- setup the break point.
  - `(gdb) br main                    // set the breakpoint at main`
  - `(gdb) c                          // continue`



```
(gdb) br main
Breakpoint 1 at 0x80103040: file main.c, line 19.
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x80103040 <main>:   endbr32

Thread 1 hit Breakpoint 1, main () at main.c:19
19        {
(gdb)
```

connect to the debug target

```
(gdb) list
```



```
(gdb) list
14      // Bootstrap processor starts running C code here.
15      // Allocate a real stack and switch to it, first
16      // doing some setup required for memory allocator to work.
17      int
18      main(void)
19      {
20        kinit1(end, P2V(4*1024*1024)); // phys page allocator
21        kvmalloc();      // kernel page table
22        mpinit();        // detect other processors
23        lapicinit();     // interrupt controller
(gdb) _
```

```
(gdb) br userinit

(gdb) c

(gdb) n or s

      s: step into

      n: next
```

ctags

Materials from EE485 taught by
Youjip Won and Kyungsoo Park

ctags

- ctags
  - A tool for making “tag file” which contains the location of function, variable, string, and etc. of a source file.
  - You can use ”tag file” to find the objects by name.
- ctags installation (linux)

  -
    ```
    $ sudo apt-get install ctags
    ```

Setup ctags (1)

- Step1. building tag file
  - `$ ctags [option] [filename(s)]`
  - example

    `$ ctags hello.c` :making tag file which include index of “hello.c”

    `$ ctags -R` :making the tag file for all subdirectories under current directory.
  - option list

option	PATH
-R	Scan all subdirectory recursively.
--exclude=[pattern]	Exclude files and directories which have ‘pattern’ in name fr om creating tag file.
-x	Print tags as table to stdout without creating tag file.

Setup ctags (2)

- Step2. Setting Tag file for Vim
  - open vim and type command below in vim command mode

    ```
    set tags=[path of tag file]
    ```

  - adding above command in ~/.vimrc will save this configuration

Prepare the practice for ctags (1)

- Download the source code of bash with below command

```
$ git clone https://sourceware.org/git/glibc.git
```

- Move to the `glibc` directory in and Make `tags` file with below command

```
$ cd glibc
$ ctags -R
```

- If you are using an old version of `ctags` so you cannot use the `–R` option, make `tags` file with below command (The eelab5 server has an old version)

```
$ cd glibc
$ find . -name "*.[chS]" | ctags -
```

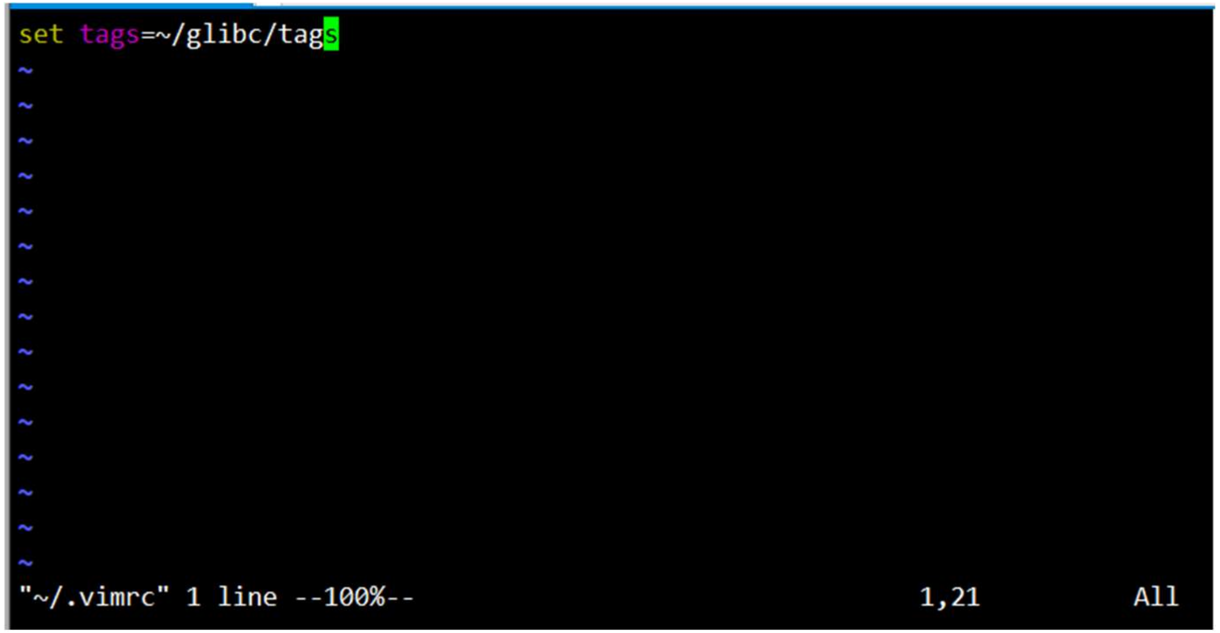
Prepare the practice for the ctags (2)

- Open the ~/.vimrc file

```
$ vi ~/.vimrc
```

- Type below text in `~/.vimrc` file

```
set tags+="Path of tha tags file you made"
```



```
set tags=~/glibc/tags
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"~/.vimrc" 1 line --100%--                              1,21          All
```

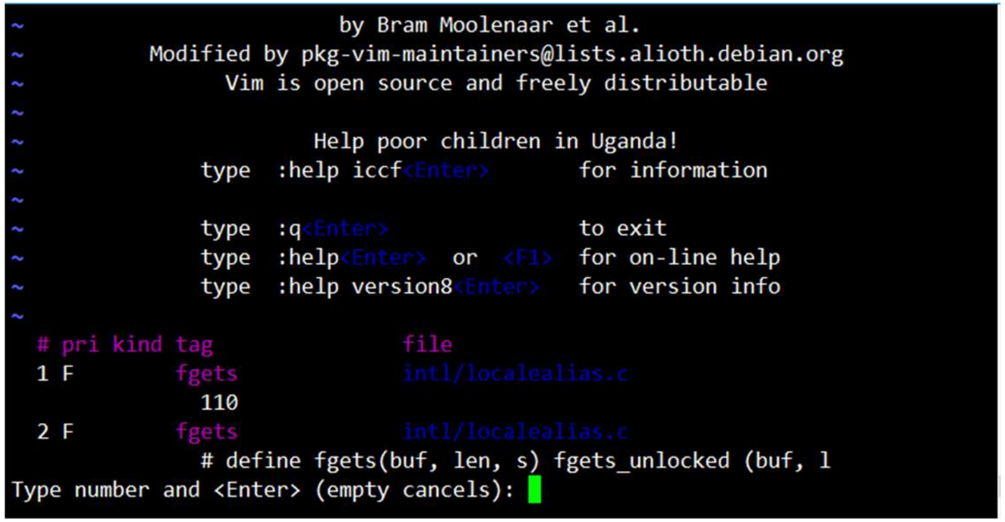
Prepare the practice for the ctags (3)

- Save and quit the vi

```
:wq
```

- Open vi in glibc directory and run below command for testing

```
:tj fgets
```



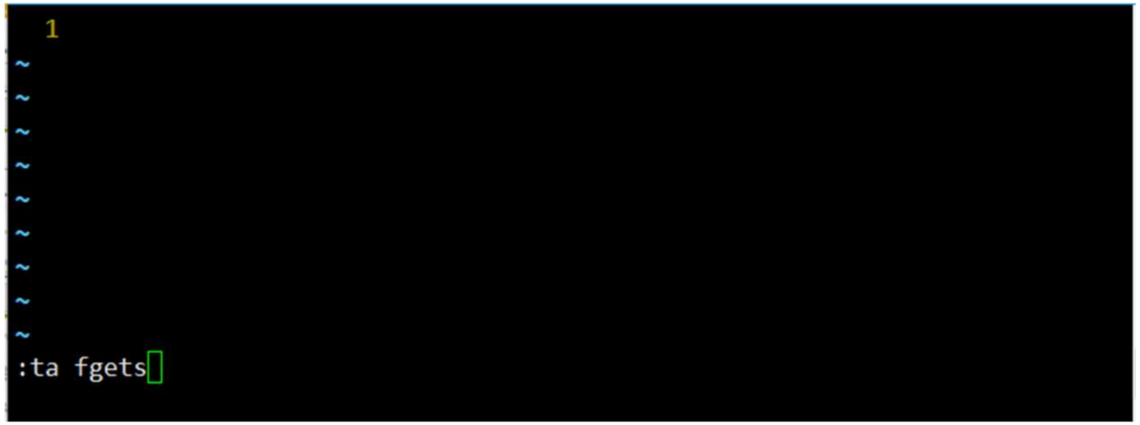
```
~                         by Bram Moolenaar et al.
~          Modified by pkg-vim-maintainers@lists.alioth.debian.org
~                Vim is open source and freely distributable
~
~                       Help poor children in Uganda!
~              type  :help iccf<Enter>       for information
~
~              type  :q<Enter>               to exit
~              type  :help<Enter>  or  <F1>  for on-line help
~              type  :help version8<Enter>   for version info
~
  # pri kind tag               file
  1 F        fgets             intl/localealias.c
               110
  2 F        fgets             intl/localealias.c
               # define fgets(buf, len, s) fgets_unlocked (buf, l
Type number and <Enter> (empty cancels):
```

ctags command in vim

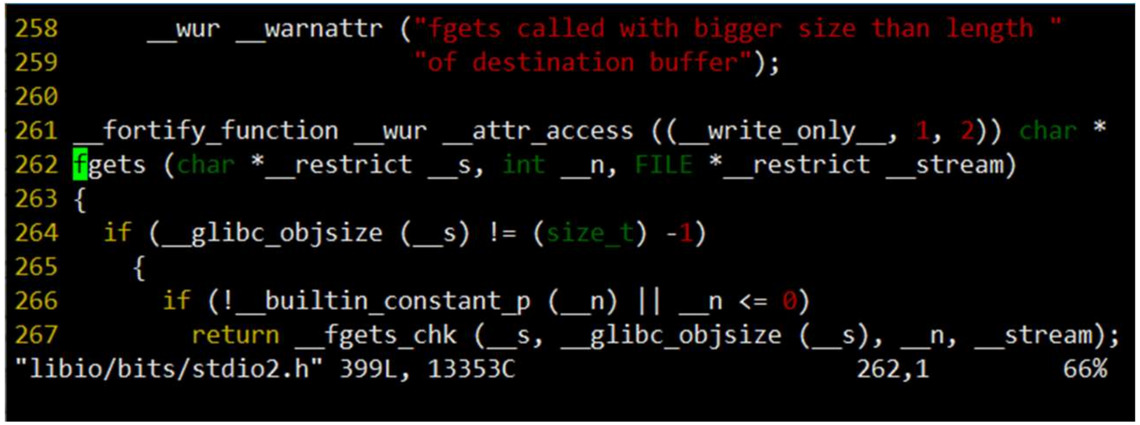
Command	Description of the command
`:ta [tagname]`	Jump to definition of tagname
`:po`	Jump back to previous position
`Ctrl + ]`	Shorcut of command ‘ta’. The tagname is a string pointed by current cursor.
`Ctrl + t`	Shorcut of command ‘po’.
`:ts [tagname]`	Print list of definition of tagname
`:tj [tagname]`	Jump to definition of tagname if there is single definition Otherwise, print list of definition of tagname
`:tn`	Jump to next definition in list printed by command ts
`:tp`	Jump to previous definition in list printed by command ts
`:tags`	Print history of jump using ctags
`:sts`	Same with command ‘ts’, but split windows

The command ‘ta’

“:ta fgets”:
Jump to definition of fgets



```
~
~
~
~
~
~
~
~
~
:ta fgets
```

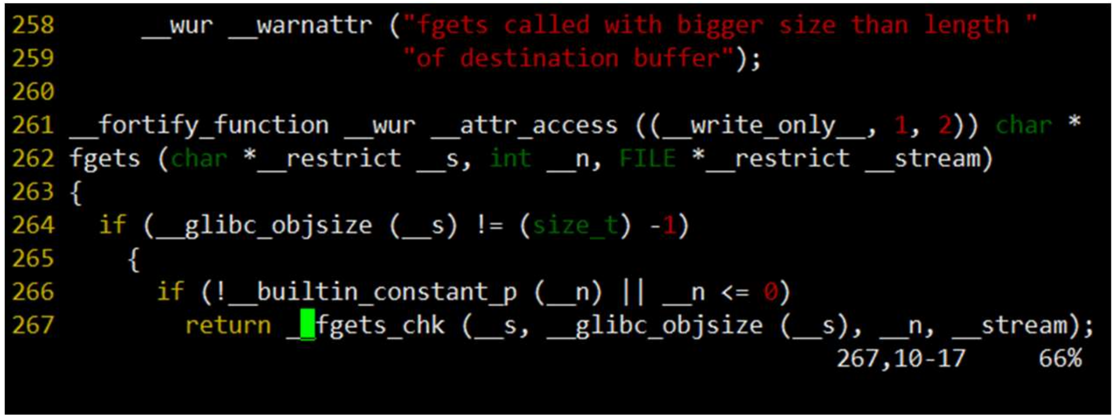


```
     __wur __warnattr ("fgets called with bigger size than length "
                       "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
"libio/bits/stdio2.h" 399L, 13353C                      262,1         66%
```

Shorcut of the command ‘ta’

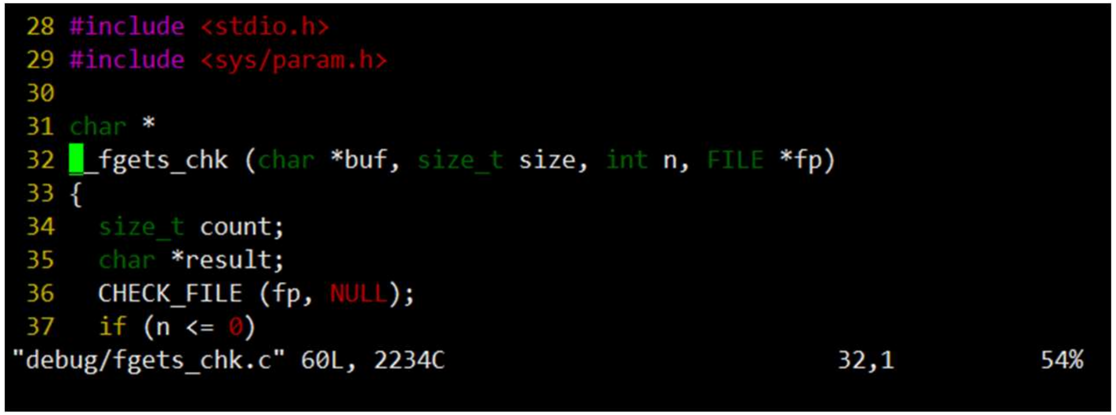
“Ctrl-]”:

Jump to definition of function pointed by current cursor



```
      __wur __warnattr ("fgets called with bigger size than length "
                        "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
                                                      267,10-17     66%
```

Type Ctrl-]

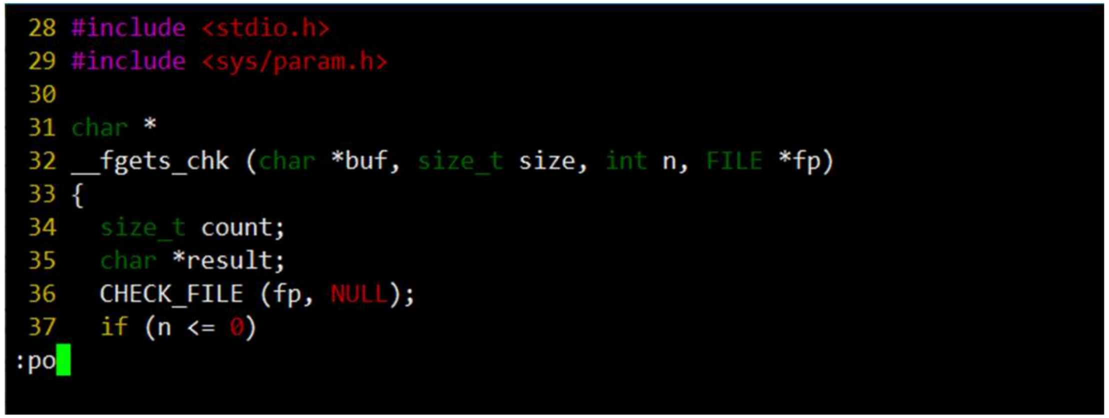


```
#include <stdio.h>
#include <sys/param.h>

char *
__fgets_chk (char *buf, size_t size, int n, FILE *fp)
{
  size_t count;
  char *result;
  CHECK_FILE (fp, NULL);
  if (n <= 0)
"debug/fgets_chk.c" 60L, 2234C                        32,1           54%
```

The command “po” and its shortcut

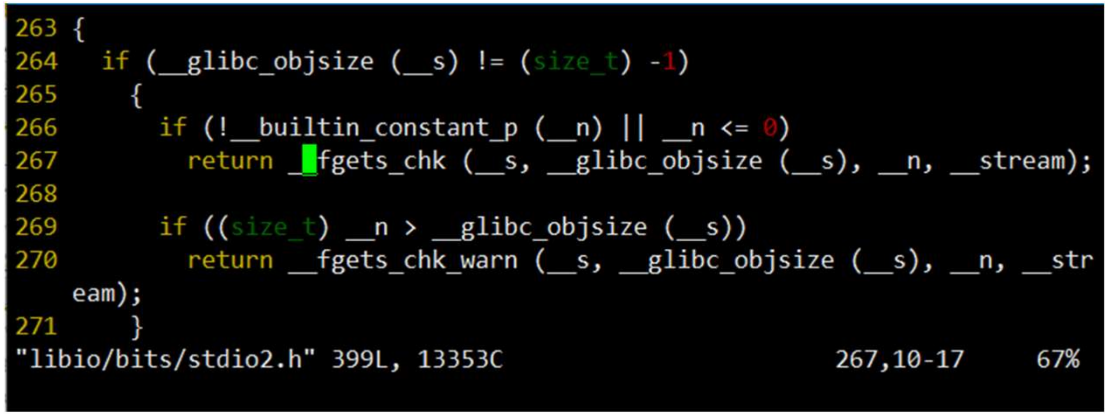
“:po” & “Ctrl-t”:

Jump to previous position (Continue to previous slide...)



```
#include <stdio.h>
#include <sys/param.h>

char *
__fgets_chk (char *buf, size_t size, int n, FILE *fp)
{
  size_t count;
  char *result;
  CHECK_FILE (fp, NULL);
  if (n <= 0)
:po
```

Type po or Ctrl-t

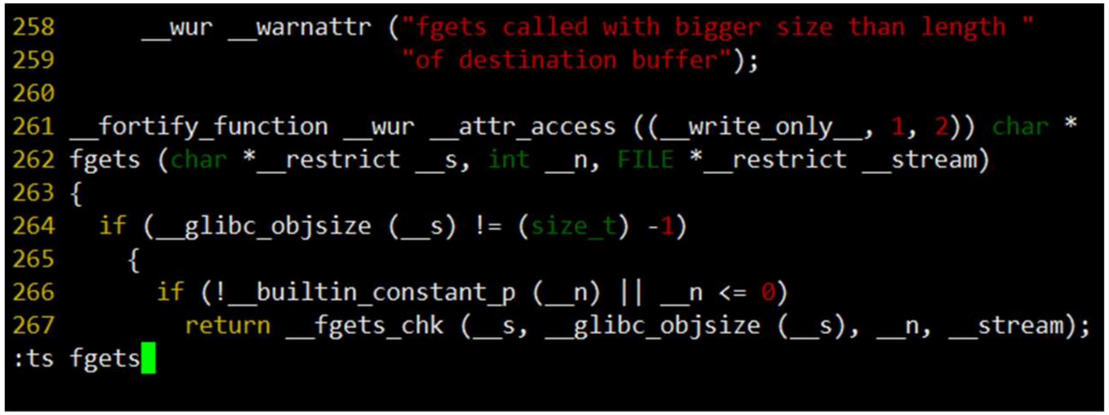


```
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);

      if ((size_t) __n > __glibc_objsize (__s))
        return __fgets_chk_warn (__s, __glibc_objsize (__s), __n, __str
eam);
    }
"libio/bits/stdio2.h" 399L, 13353C                      267,10-17      67%
```

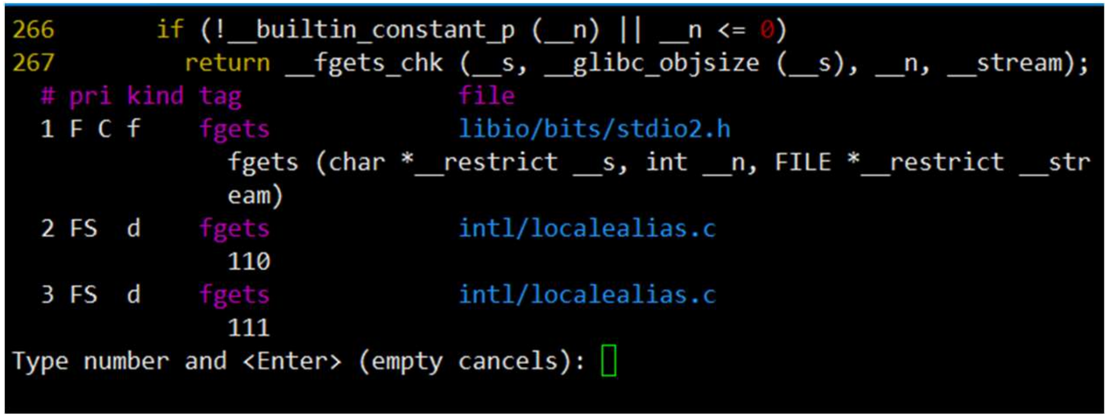
The command “ts”

`“:ts fgets”:`

Print list of definition of fgets



```
       __wur __warnattr ("fgets called with bigger size than length "
                         "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
:ts fgets
```



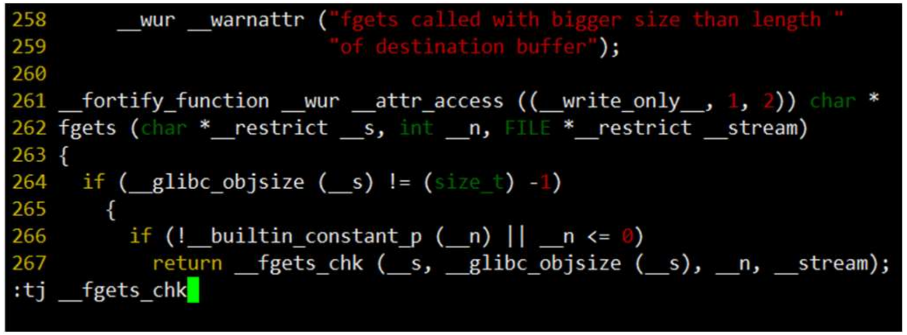
```
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
  # pri kind tag               file
  1 F C f    fgets             libio/bits/stdio2.h
               fgets (char *__restrict __s, int __n, FILE *__restrict __str
               eam)
  2 FS  d    fgets             intl/localealias.c
               110
  3 FS  d    fgets             intl/localealias.c
               111
Type number and <Enter> (empty cancels):
```

The command “tj”

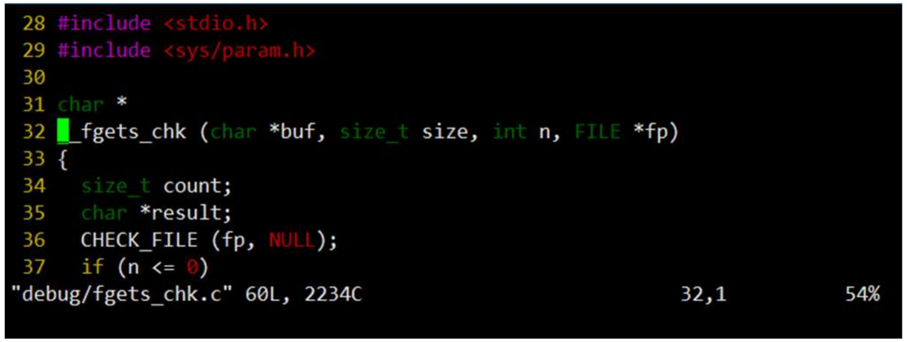
“:tj __fgets_chk”:

Jump to definition of _fgets_chk

*It has single definition



```
       __wur __warnattr ("fgets called with bigger size than length "
                          "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
:tj __fgets_chk
```

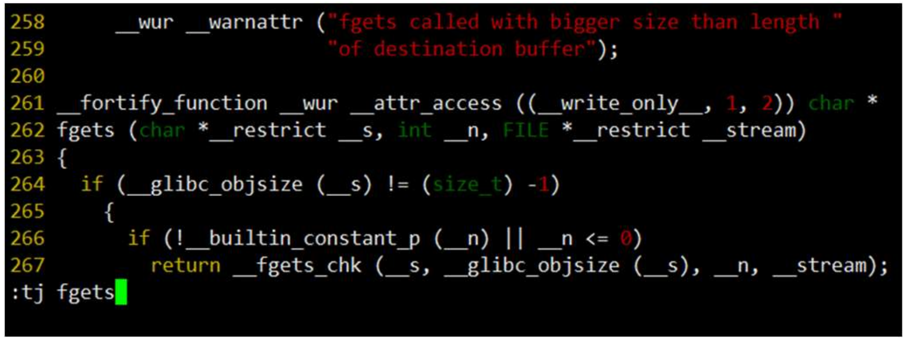


```
#include <stdio.h>
#include <sys/param.h>

char *
_fgets_chk (char *buf, size_t size, int n, FILE *fp)
{
  size_t count;
  char *result;
  CHECK_FILE (fp, NULL);
  if (n <= 0)
"debug/fgets_chk.c" 60L, 2234C                         32,1          54%
```

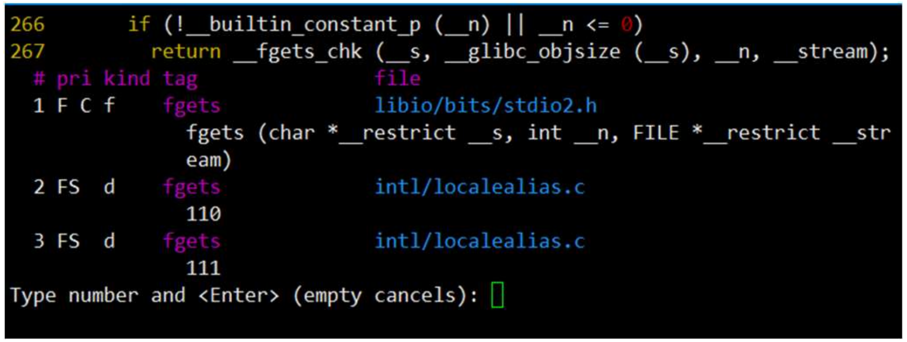
“:tj fgets”:

Print the list of definition of fgets

*It has multiple definitions



```
       __wur __warnattr ("fgets called with bigger size than length "
                          "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
:tj fgets
```



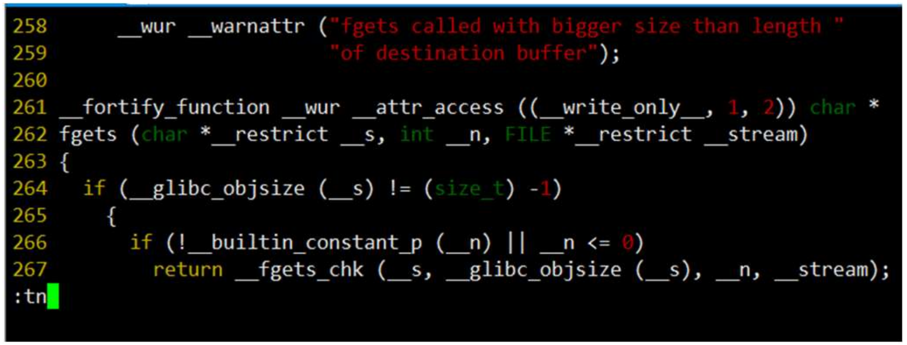
```
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
  # pri kind tag              file
  1 F C f    fgets            libio/bits/stdio2.h
               fgets (char *__restrict __s, int __n, FILE *__restrict __str
               eam)
  2 FS  d    fgets            intl/localealias.c
               110
  3 FS  d    fgets            intl/localealias.c
               111
Type number and <Enter> (empty cancels):
```

Type the number of tag you want

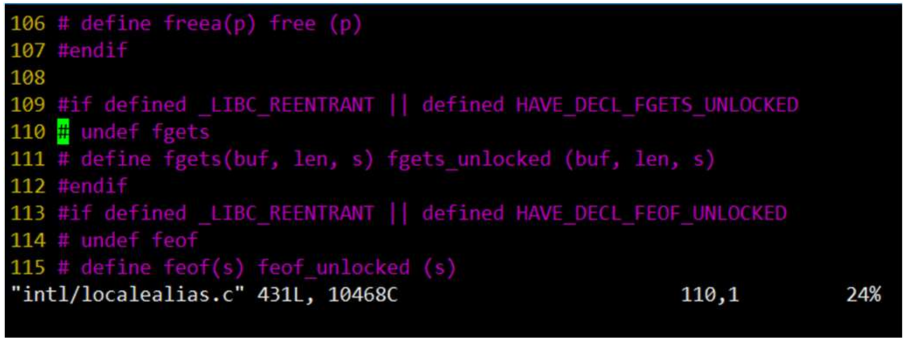
The command “tn” and “tp” (Cont. to “tj fgets”)

“:tn”:
Jump to next definition of fgets

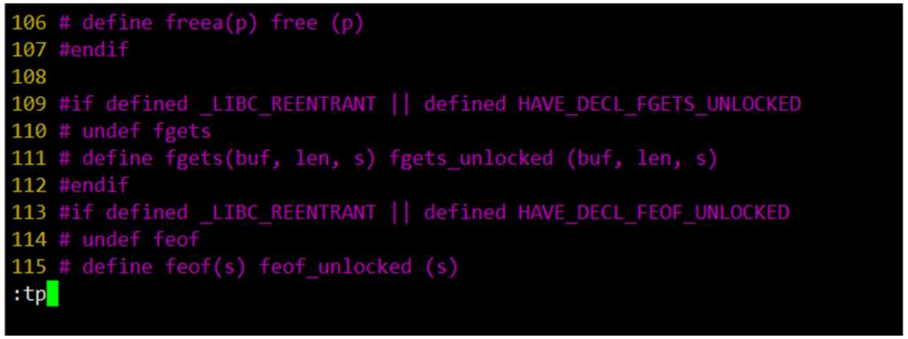
“:tp”:
Jump to previous definition of fgets



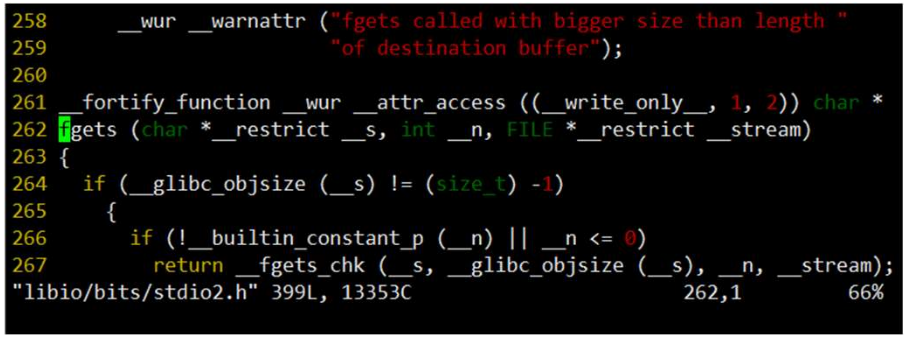
```
     __wur __warnattr ("fgets called with bigger size than length "
                       "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
:tn
```



```
# define freea(p) free (p)
#endif

#if defined _LIBC_REENTRANT || defined HAVE_DECL_FGETS_UNLOCKED
# undef fgets
# define fgets(buf, len, s) fgets_unlocked (buf, len, s)
#endif
#if defined _LIBC_REENTRANT || defined HAVE_DECL_FEOF_UNLOCKED
# undef feof
# define feof(s) feof_unlocked (s)
"intl/localealias.c" 431L, 10468C                    110,1          24%
```



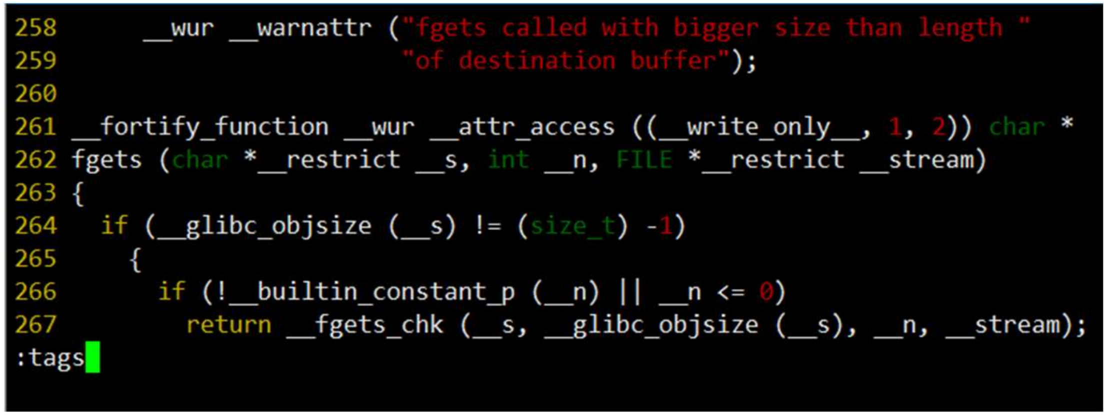
```
# define freea(p) free (p)
#endif

#if defined _LIBC_REENTRANT || defined HAVE_DECL_FGETS_UNLOCKED
# undef fgets
# define fgets(buf, len, s) fgets_unlocked (buf, len, s)
#endif
#if defined _LIBC_REENTRANT || defined HAVE_DECL_FEOF_UNLOCKED
# undef feof
# define feof(s) feof_unlocked (s)
:tp
```



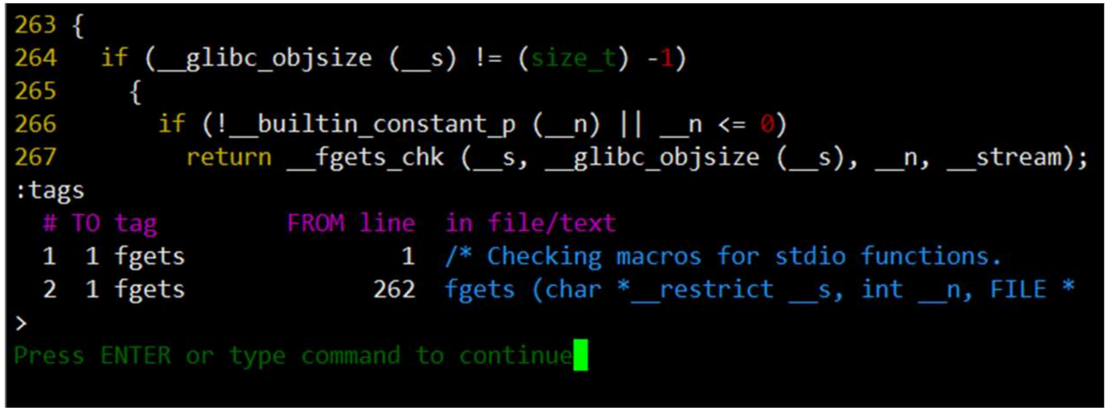
```
     __wur __warnattr ("fgets called with bigger size than length "
                       "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
"libio/bits/stdio2.h" 399L, 13353C                   262,1          66%
```

The command “tags” (Cont. to prev. slide)

“:tags”:
Print history of jump using ctags



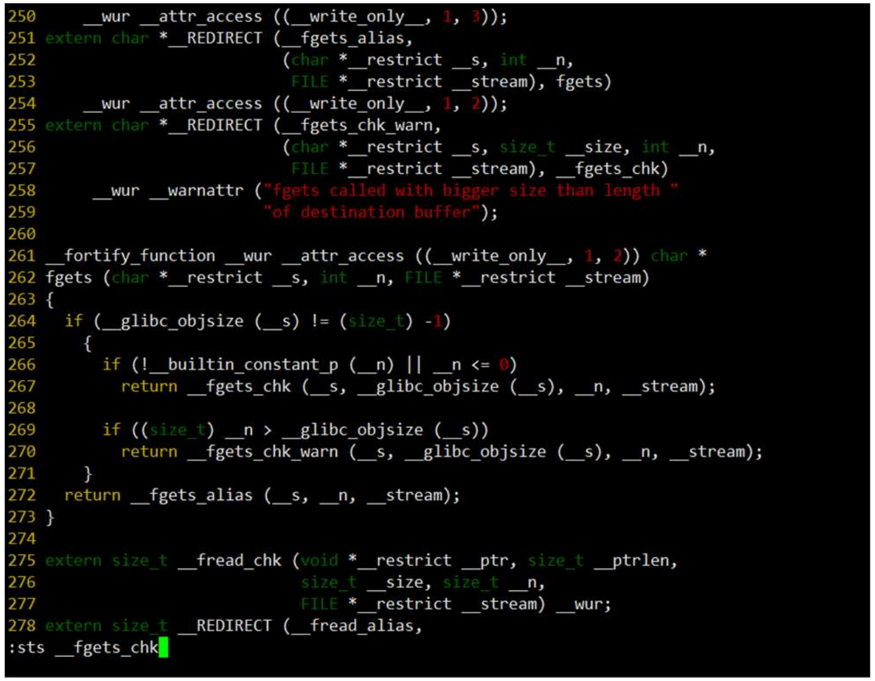
```
258       __wur __warnattr ("fgets called with bigger size than length "
259                          "of destination buffer");
260
261 __fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
262 fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
263 {
264   if (__glibc_objsize (__s) != (size_t) -1)
265     {
266       if (!__builtin_constant_p (__n) || __n <= 0)
267         return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
:tags
```



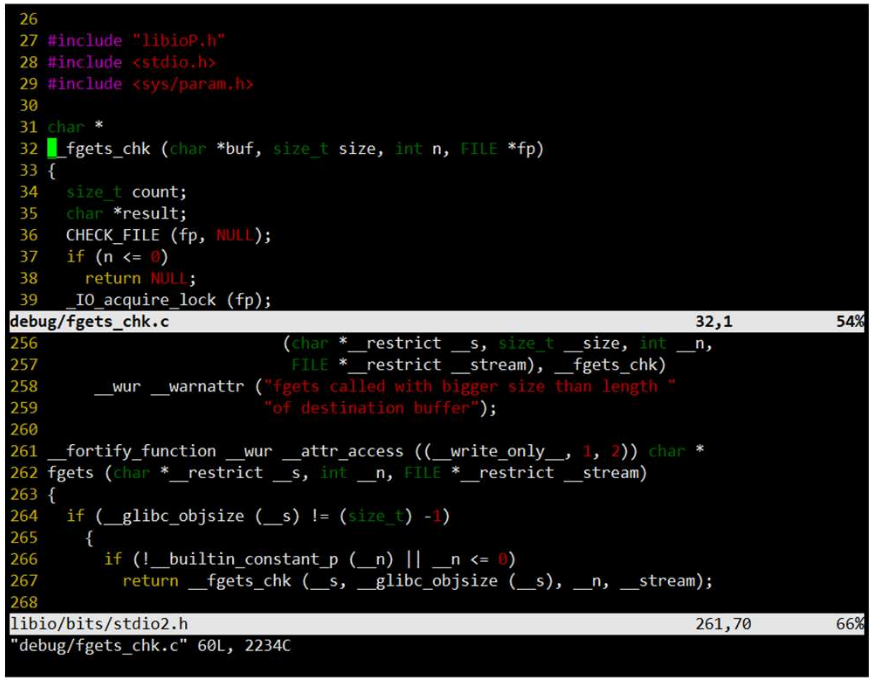
```
263 {
264   if (__glibc_objsize (__s) != (size_t) -1)
265     {
266       if (!__builtin_constant_p (__n) || __n <= 0)
267         return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);
:tags
  # TO tag         FROM line  in file/text
  1  1 fgets               1  /* Checking macros for stdio functions.
  2  1 fgets             262  fgets (char *__restrict __s, int __n, FILE *
>
Press ENTER or type command to continue
```

The command “sts” (Cont. to prev. slide)

“:sts __fgets_chk”:
Same with command ‘ts’, but split windows



```
      __wur __attr_access ((__write_only__, 1, 3));
extern char *__REDIRECT (__fgets_alias,
                         (char *__restrict __s, int __n,
                          FILE *__restrict __stream), fgets)
      __wur __attr_access ((__write_only__, 1, 2));
extern char *__REDIRECT (__fgets_chk_warn,
                         (char *__restrict __s, size_t __size, int __n,
                          FILE *__restrict __stream), __fgets_chk)
       __wur __warnattr ("fgets called with bigger size than length "
                         "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);

      if ((size_t) __n > __glibc_objsize (__s))
        return __fgets_chk_warn (__s, __glibc_objsize (__s), __n, __stream);
    }
  return __fgets_alias (__s, __n, __stream);
}

extern size_t __fread_chk (void *__restrict __ptr, size_t __ptrlen,
                           size_t __size, size_t __n,
                           FILE *__restrict __stream) __wur;
extern size_t __REDIRECT (__fread_alias,
:sts __fgets_chk
```



```

#include "libioP.h"
#include <stdio.h>
#include <sys/param.h>

char *
__fgets_chk (char *buf, size_t size, int n, FILE *fp)
{
  size_t count;
  char *result;
  CHECK_FILE (fp, NULL);
  if (n <= 0)
    return NULL;
  _IO_acquire_lock (fp);
debug/fgets_chk.c                                            32,1           54%
                         (char *__restrict __s, size_t __size, int __n,
                          FILE *__restrict __stream), __fgets_chk)
       __wur __warnattr ("fgets called with bigger size than length "
                         "of destination buffer");

__fortify_function __wur __attr_access ((__write_only__, 1, 2)) char *
fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
{
  if (__glibc_objsize (__s) != (size_t) -1)
    {
      if (!__builtin_constant_p (__n) || __n <= 0)
        return __fgets_chk (__s, __glibc_objsize (__s), __n, __stream);

libio/bits/stdio2.h                                          261,70         66%
"debug/fgets_chk.c" 60L, 2234C
```

cscope

Materials from EE485 taught by
Youjip Won and Kyungsoo Park

cscope

- The tool supports functionalities that are not in ctags.
- The functionalities include:
  - Find the definition of symbol.
  - Find the positions where a given function is called or a variable is accessed.
  - Find the functions that specific function calls.
  - Find text string.
- Installation (Linux)

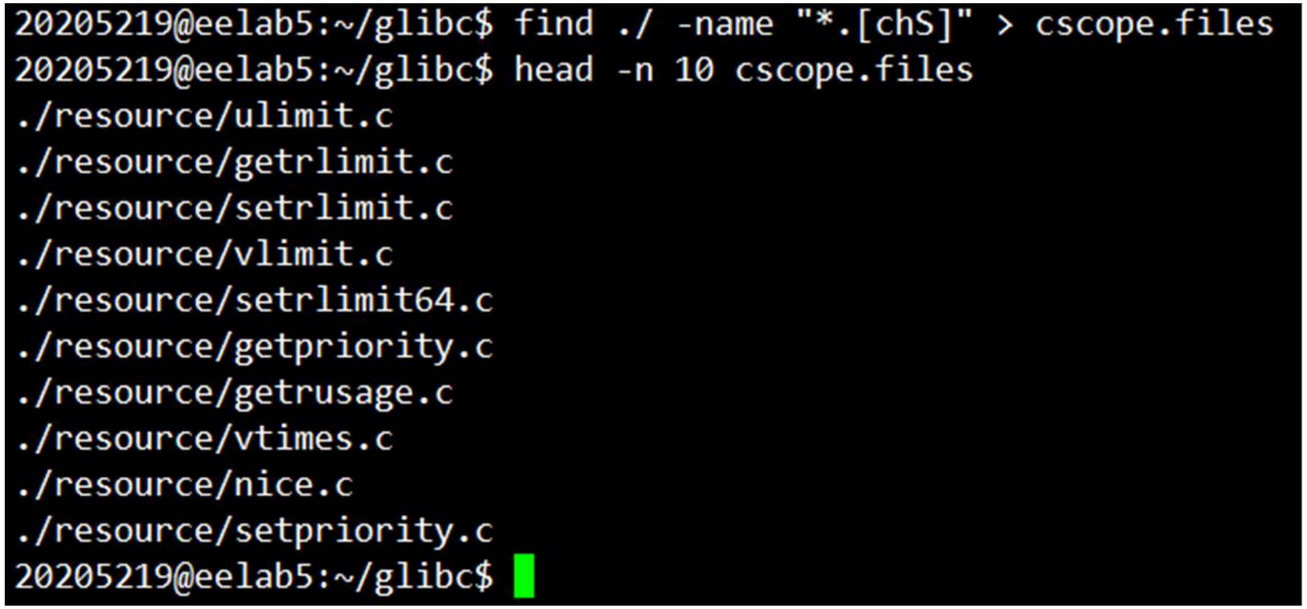
  -
    ```
    $ sudo apt-get install cscope
    ```

Setup cscope (1)

- Step 1. Make a file “cscope.files” containing list of files what you want to analyze in the `glibc` directory.

```
$ find . -name “*.[chS]” > cscope.files
```

- Result:



```
20205219@eelab5:~/glibc$ find ./ -name "*.[chS]" > cscope.files
20205219@eelab5:~/glibc$ head -n 10 cscope.files
./resource/ulimit.c
./resource/getrlimit.c
./resource/setrlimit.c
./resource/vlimit.c
./resource/setrlimit64.c
./resource/getpriority.c
./resource/getrusage.c
./resource/vtimes.c
./resource/nice.c
./resource/setpriority.c
20205219@eelab5:~/glibc$ 
```

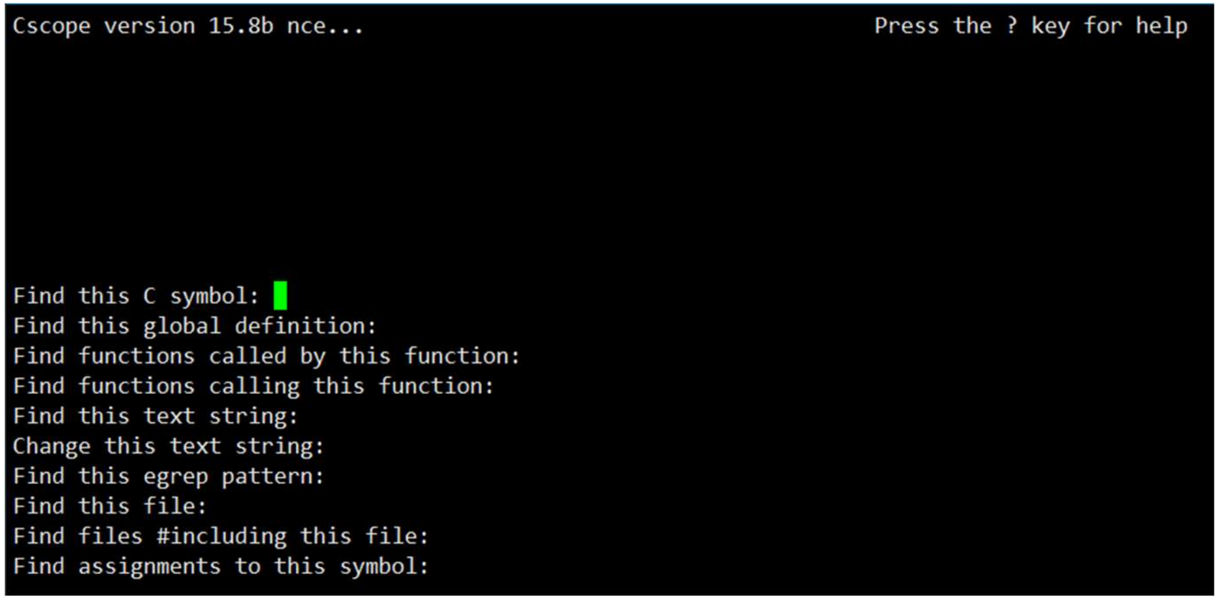
Setup cscope (2)

- Step 2. Make a file “cscope.out” which is a database file for cscope with below command

```
$ cscope -i cscope.files
```

The cscope.out has the information of each file in cscope.files

- Result:



```
Cscope version 15.8b nce...                                  Press the ? key for help




Find this C symbol: █
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

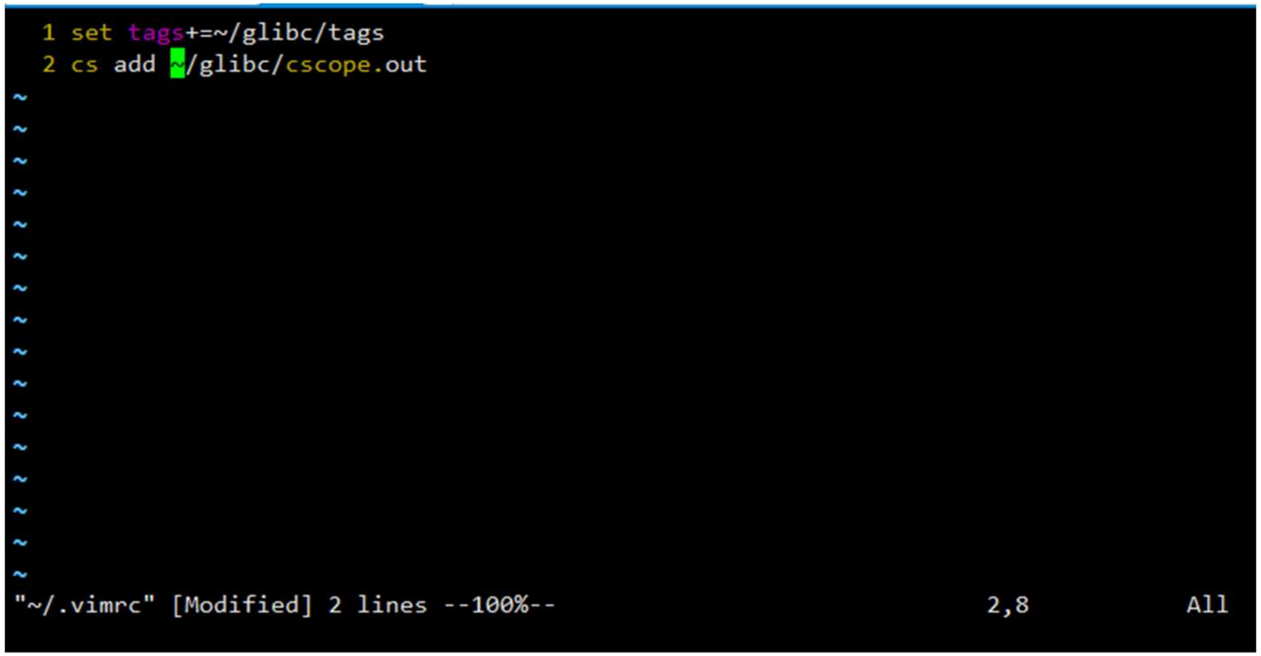
*** You can leave the cscope with Ctrl-D**

Setup cscope (3)

- Step 3. Set the cscope in vim with the command below.

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

- Add above command in `~/.vimrc` will save this configuration.



```
set tags+=~/glibc/tags
cs add ~/glibc/cscope.out
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"~/.vimrc" [Modified] 2 lines --100%--                        2,8           All
```

command

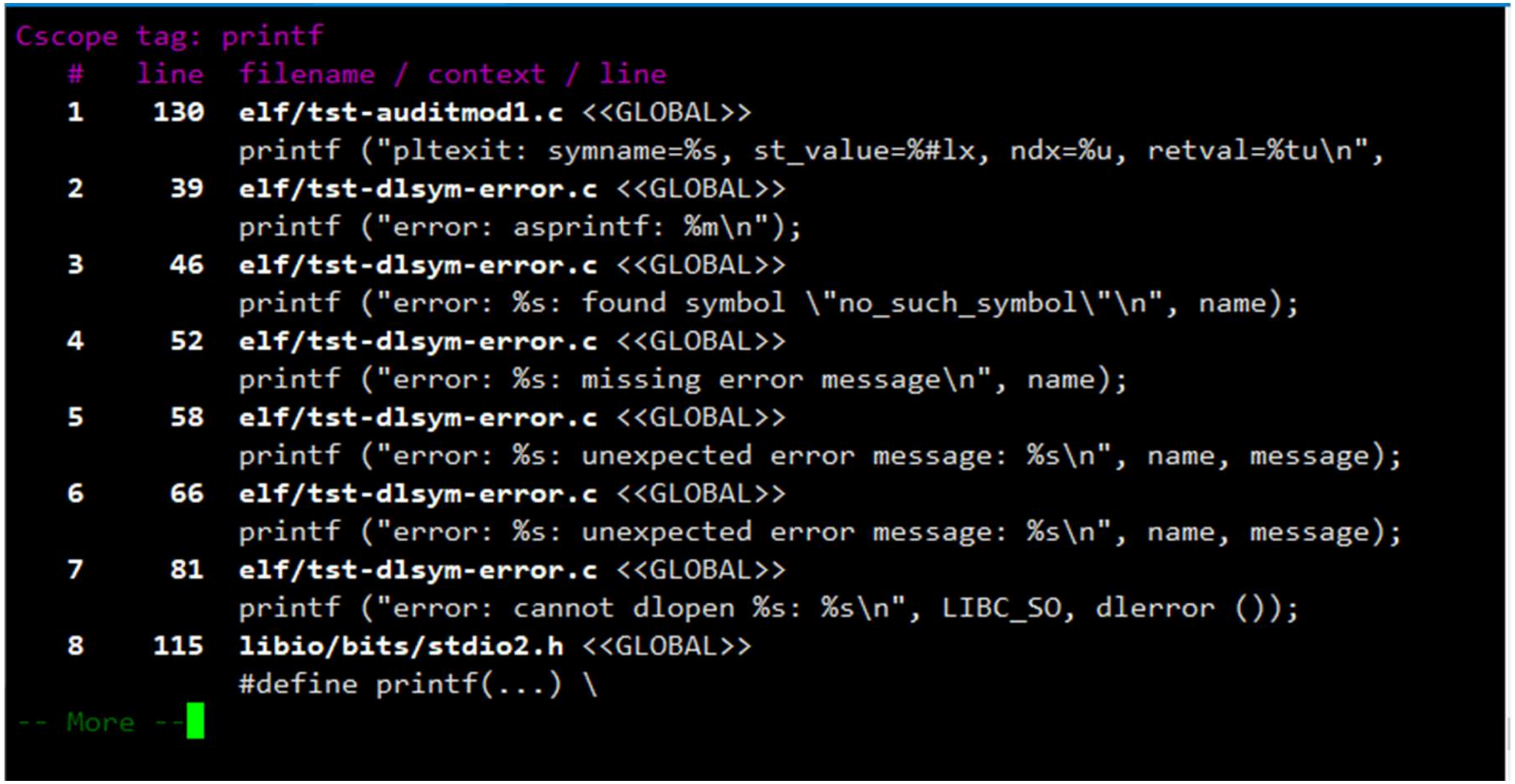
- Find the functions that "call" a given function or that "is called" by a given function.
  - Find all functions that call malloc()
  - Find all functions that are called by malloc().

```
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`

Usage of cscope in vim

“cs find s printf”:
Search the symbol “printf”

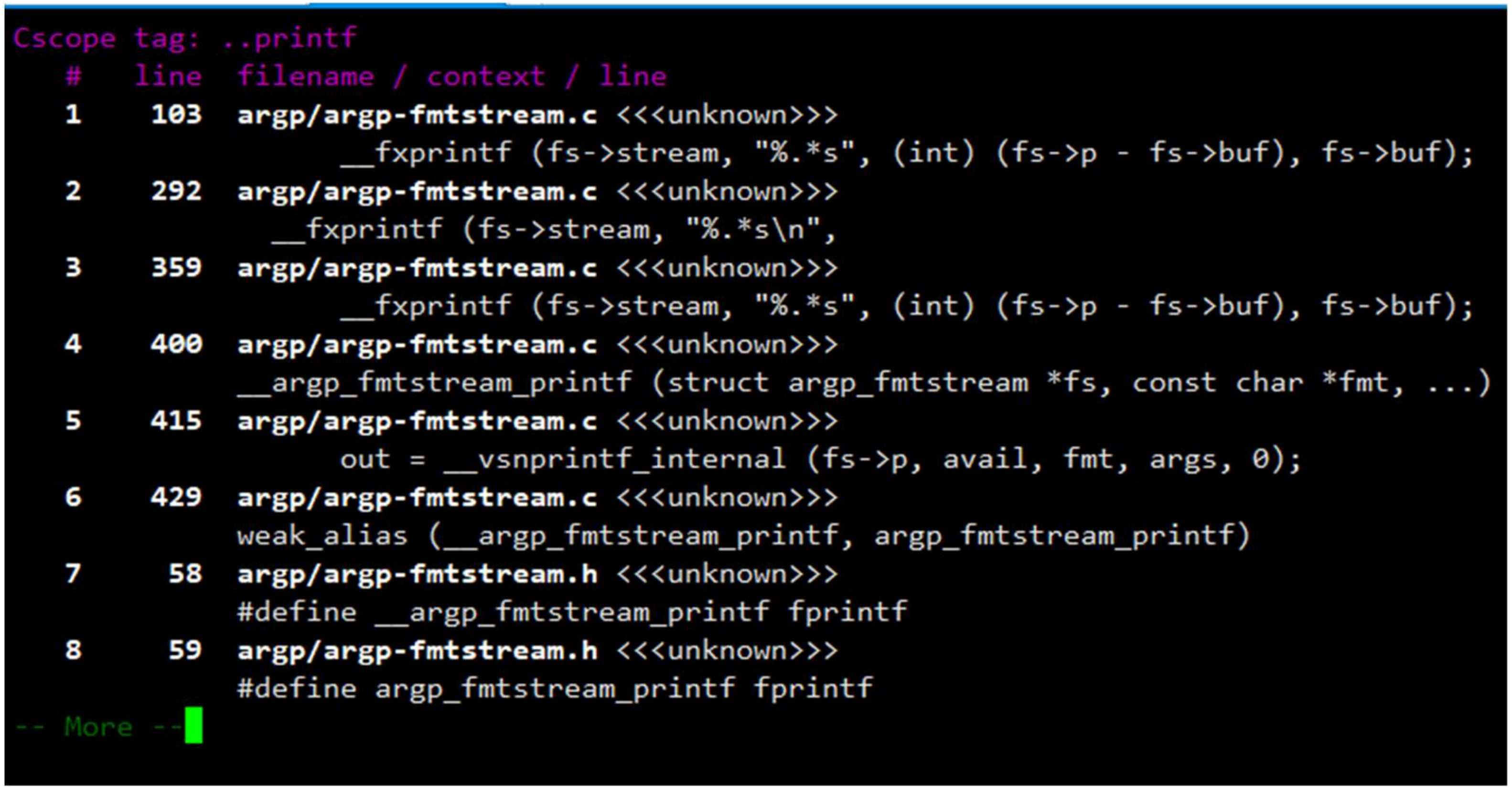


```
Cscope tag: printf
   #   line  filename / context / line
   1    130  elf/tst-auditmod1.c <<GLOBAL>>
             printf ("pltexit: symname=%s, st_value=%#lx, ndx=%u, retval=%tu\n",
   2     39  elf/tst-dlsym-error.c <<GLOBAL>>
             printf ("error: asprintf: %m\n");
   3     46  elf/tst-dlsym-error.c <<GLOBAL>>
             printf ("error: %s: found symbol \"no_such_symbol\"\n", name);
   4     52  elf/tst-dlsym-error.c <<GLOBAL>>
             printf ("error: %s: missing error message\n", name);
   5     58  elf/tst-dlsym-error.c <<GLOBAL>>
             printf ("error: %s: unexpected error message: %s\n", name, message);
   6     66  elf/tst-dlsym-error.c <<GLOBAL>>
             printf ("error: %s: unexpected error message: %s\n", name, message);
   7     81  elf/tst-dlsym-error.c <<GLOBAL>>
             printf ("error: cannot dlopen %s: %s\n", LIBC_SO, dlerror ());
   8    115  libio/bits/stdio2.h <<GLOBAL>>
             #define printf(...) \
-- More --
```

Usage of cscope in vim

`“cs find e ..printf”:`

Search the symbol `..printf` with regular expression

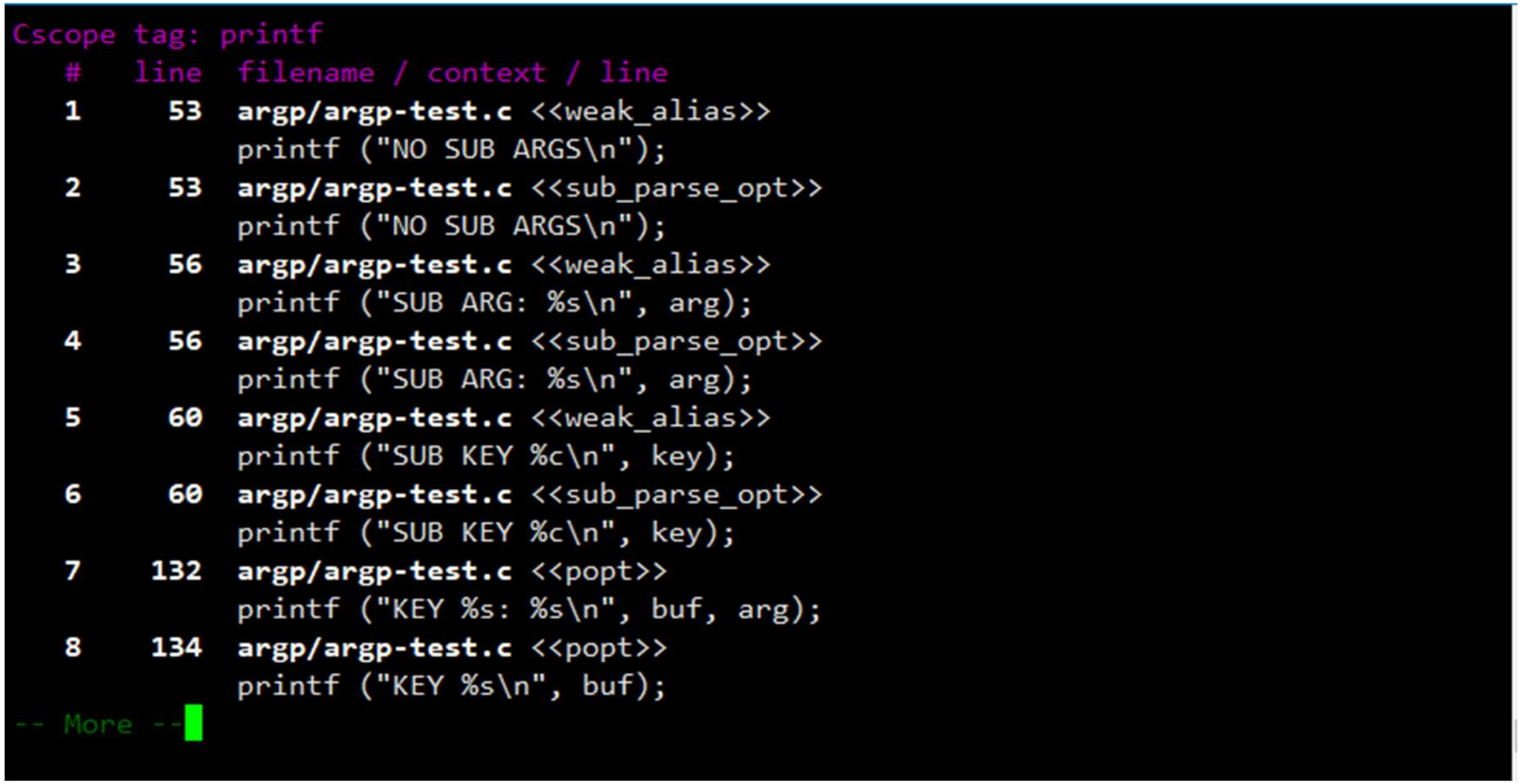


```
Cscope tag: ..printf
   #   line  filename / context / line
   1    103  argp/argp-fmtstream.c <<<unknown>>>
                   __fxprintf (fs->stream, "%.*s", (int) (fs->p - fs->buf), fs->buf);
   2    292  argp/argp-fmtstream.c <<<unknown>>>
               __fxprintf (fs->stream, "%.*s\n",
   3    359  argp/argp-fmtstream.c <<<unknown>>>
                   __fxprintf (fs->stream, "%.*s", (int) (fs->p - fs->buf), fs->buf);
   4    400  argp/argp-fmtstream.c <<<unknown>>>
             __argp_fmtstream_printf (struct argp_fmtstream *fs, const char *fmt, ...)
   5    415  argp/argp-fmtstream.c <<<unknown>>>
                   out = __vsnprintf_internal (fs->p, avail, fmt, args, 0);
   6    429  argp/argp-fmtstream.c <<<unknown>>>
             weak_alias (__argp_fmtstream_printf, argp_fmtstream_printf)
   7     58  argp/argp-fmtstream.h <<<unknown>>>
             #define __argp_fmtstream_printf fprintf
   8     59  argp/argp-fmtstream.h <<<unknown>>>
             #define argp_fmtstream_printf fprintf
-- More --
```

Usage of cscope in vim

“cs find c printf”:

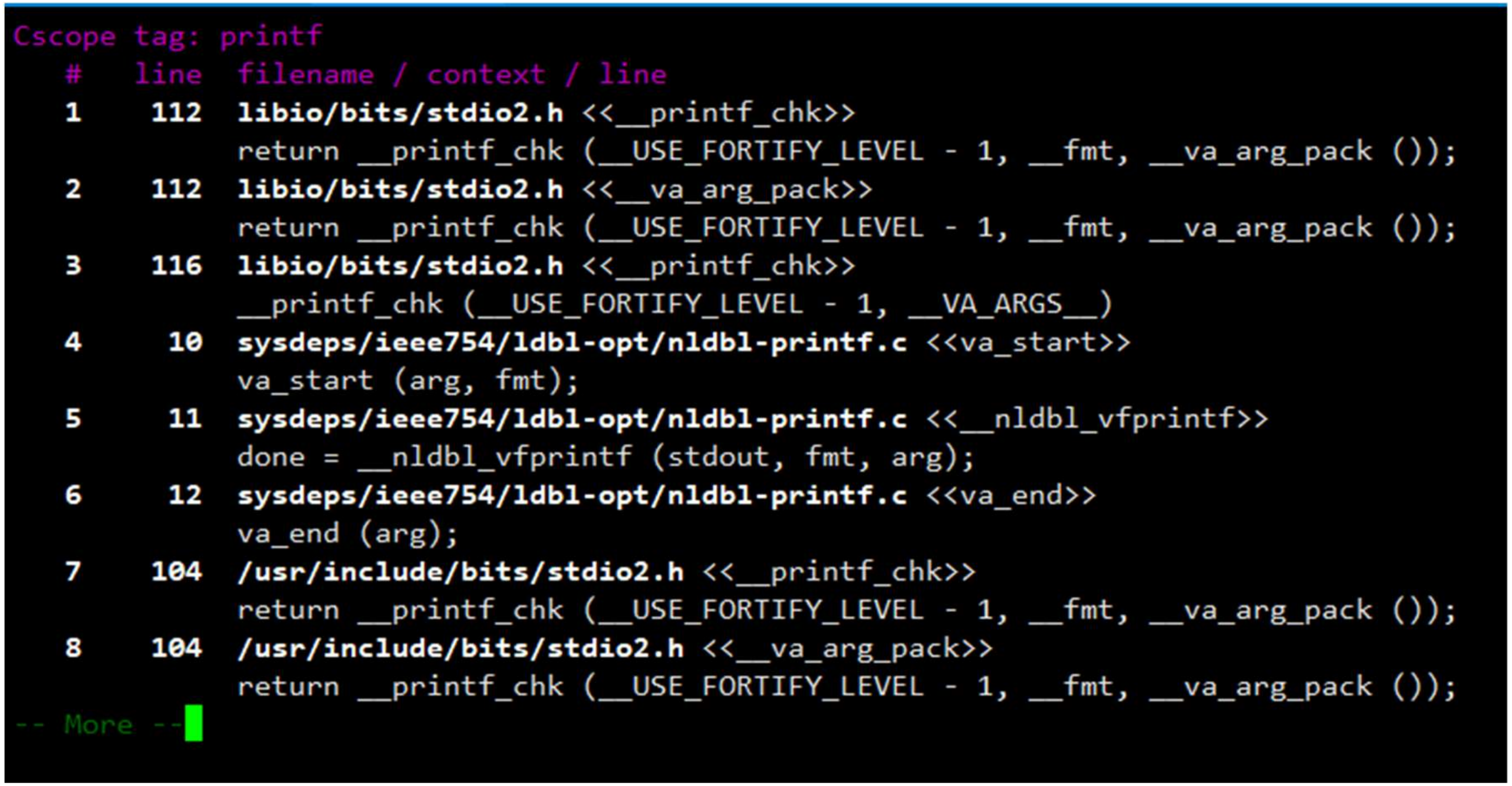
Prints the functions calling the symbol printf



```
Cscope tag: printf
   #   line  filename / context / line
   1     53  argp/argp-test.c <<weak_alias>>
             printf ("NO SUB ARGS\n");
   2     53  argp/argp-test.c <<sub_parse_opt>>
             printf ("NO SUB ARGS\n");
   3     56  argp/argp-test.c <<weak_alias>>
             printf ("SUB ARG: %s\n", arg);
   4     56  argp/argp-test.c <<sub_parse_opt>>
             printf ("SUB ARG: %s\n", arg);
   5     60  argp/argp-test.c <<weak_alias>>
             printf ("SUB KEY %c\n", key);
   6     60  argp/argp-test.c <<sub_parse_opt>>
             printf ("SUB KEY %c\n", key);
   7    132  argp/argp-test.c <<popt>>
             printf ("KEY %s: %s\n", buf, arg);
   8    134  argp/argp-test.c <<popt>>
             printf ("KEY %s\n", buf);
-- More --
```

Usage of cscope in vim

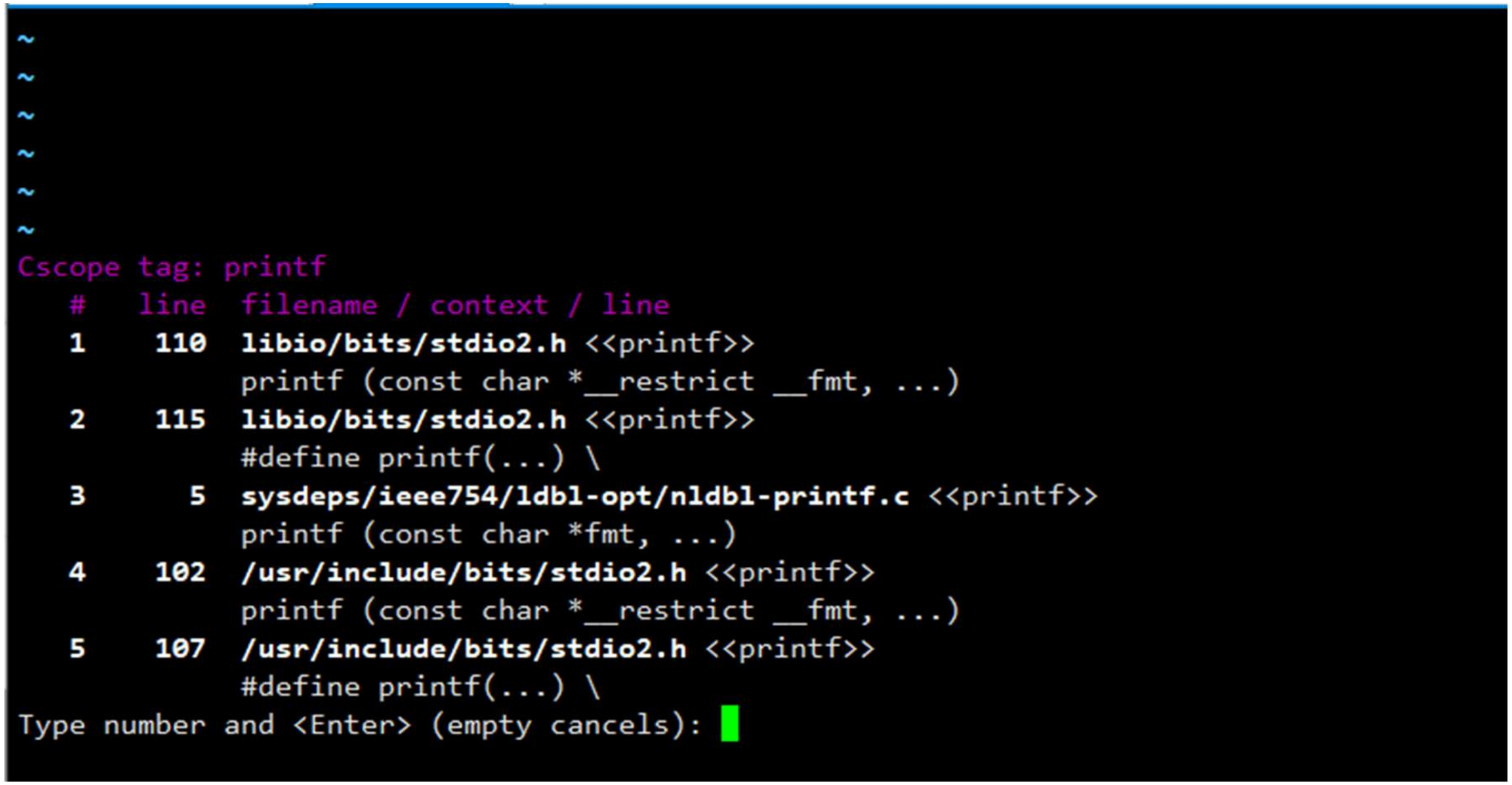
“cs find d printf”:
Print the functions called by the symbol printf.



```
Cscope tag: printf
   #   line  filename / context / line
   1    112  libio/bits/stdio2.h <<__printf_chk>>
             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
   2    112  libio/bits/stdio2.h <<__va_arg_pack>>
             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
   3    116  libio/bits/stdio2.h <<__printf_chk>>
             __printf_chk (__USE_FORTIFY_LEVEL - 1, __VA_ARGS__)
   4     10  sysdeps/ieee754/ldbl-opt/nldbl-printf.c <<va_start>>
             va_start (arg, fmt);
   5     11  sysdeps/ieee754/ldbl-opt/nldbl-printf.c <<__nldbl_vfprintf>>
             done = __nldbl_vfprintf (stdout, fmt, arg);
   6     12  sysdeps/ieee754/ldbl-opt/nldbl-printf.c <<va_end>>
             va_end (arg);
   7    104  /usr/include/bits/stdio2.h <<__printf_chk>>
             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
   8    104  /usr/include/bits/stdio2.h <<__va_arg_pack>>
             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
-- More --
```

Usage of cscope in vim

“cs find g printf”:
Search position defining the symbol printf



```
~
~
~
~
~
~
Cscope tag: printf
   #   line  filename / context / line
   1    110  libio/bits/stdio2.h <<printf>>
             printf (const char *__restrict __fmt, ...)
   2    115  libio/bits/stdio2.h <<printf>>
             #define printf(...) \
   3      5  sysdeps/ieee754/ldbl-opt/nldbl-printf.c <<printf>>
             printf (const char *fmt, ...)
   4    102  /usr/include/bits/stdio2.h <<printf>>
             printf (const char *__restrict __fmt, ...)
   5    107  /usr/include/bits/stdio2.h <<printf>>
             #define printf(...) \
Type number and <Enter> (empty cancels):
```

make

Materials from EE485 taught by
Youjip Won and Kyungsoo Park

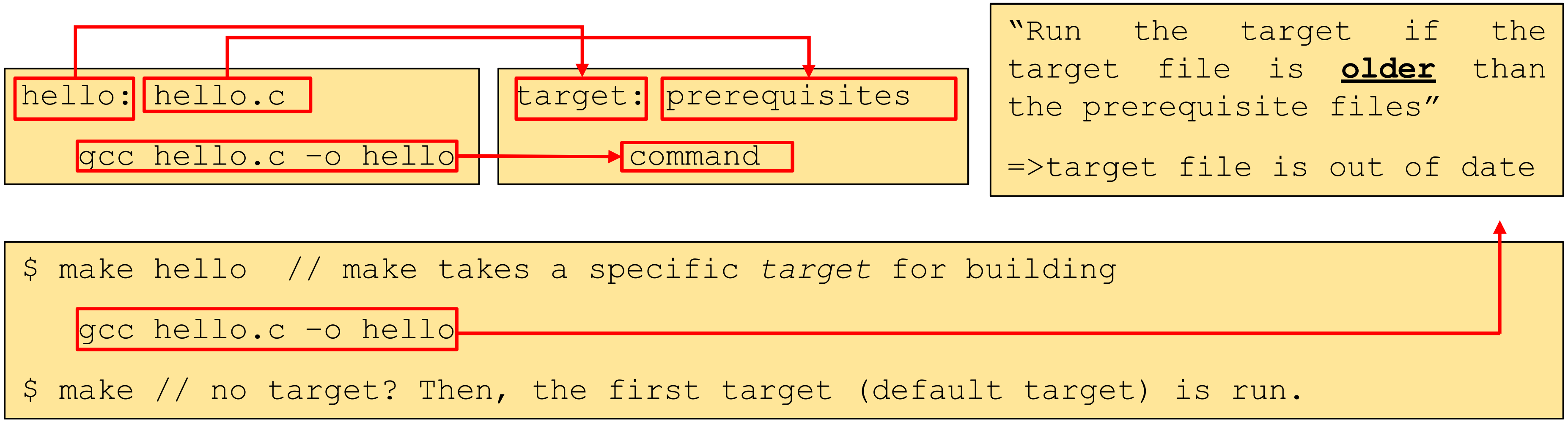
What Is Makefile?

- A file that helps decide which parts of a large program need to be recompiled.
  - Makefile = a configuration file for partial build.
- Example: build a wordprocessor with 10,000 source code files.
  - Don’t want to recompile them all if you edit just one line.
  - Recompile only those files that have changed.
  - make (a tool) enables this task.
    - A GNU tool that executes the ***“appropriate”*** rules in makefile.

```
$ make
… (recompilation)
```

`make` & makefile

- `make` executes rules in 'makefile' (or ‘Makefile’ or ‘GNUMakefile’).
  - ‘makefile’ content:



```
hello: hello.c
    gcc hello.c -o hello
```

```
target: prerequisites
        command
```

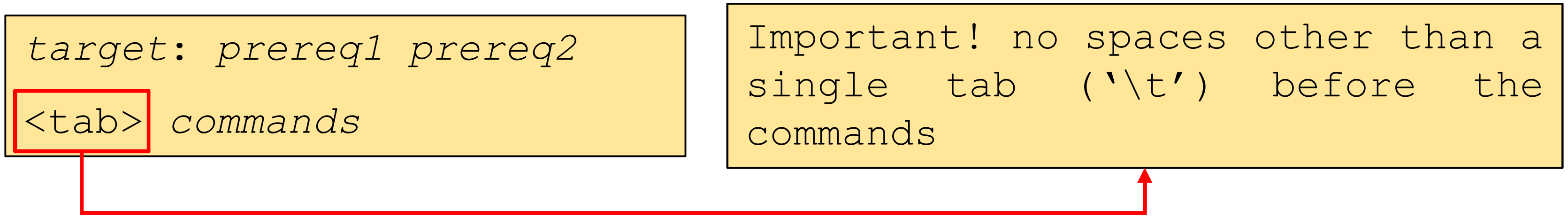
“Run the target if the target file is **older** than the prerequisite files”

=>target file is out of date

```
$ make hello  // make takes a specific target for building
   gcc hello.c -o hello
$ make // no target? Then, the first target (default target) is run.
```

Targets and Prerequisites

- ‘makefile’ contains a set of rules to build an application.
  - Can have multiple rules in one file.
  - Each rule can have (multiple) dependencies.
- **Default** rule = the first rule in ‘makefile’.
  - Default rule is executed if make runs without any arguments.
- A rule consists of a target, prerequisite(s), and command(s).



```
target: prereq1 prereq2
<tab> commands
```

Important! no spaces other than a single tab (‘\t’) before the commands

Example Rules in Makefile

```
foo: foo.o libfoo.o foo.h
    gcc foo.o libfoo.o -o foo
```

  - *target*=foo, *prerequisites*=foo.o, libfoo.o, and foo.h
  - *command*=gcc foo.o libfoo.o -o foo

- How to evaluate a rule?

  1. It finds the files for the prerequisites and the target.

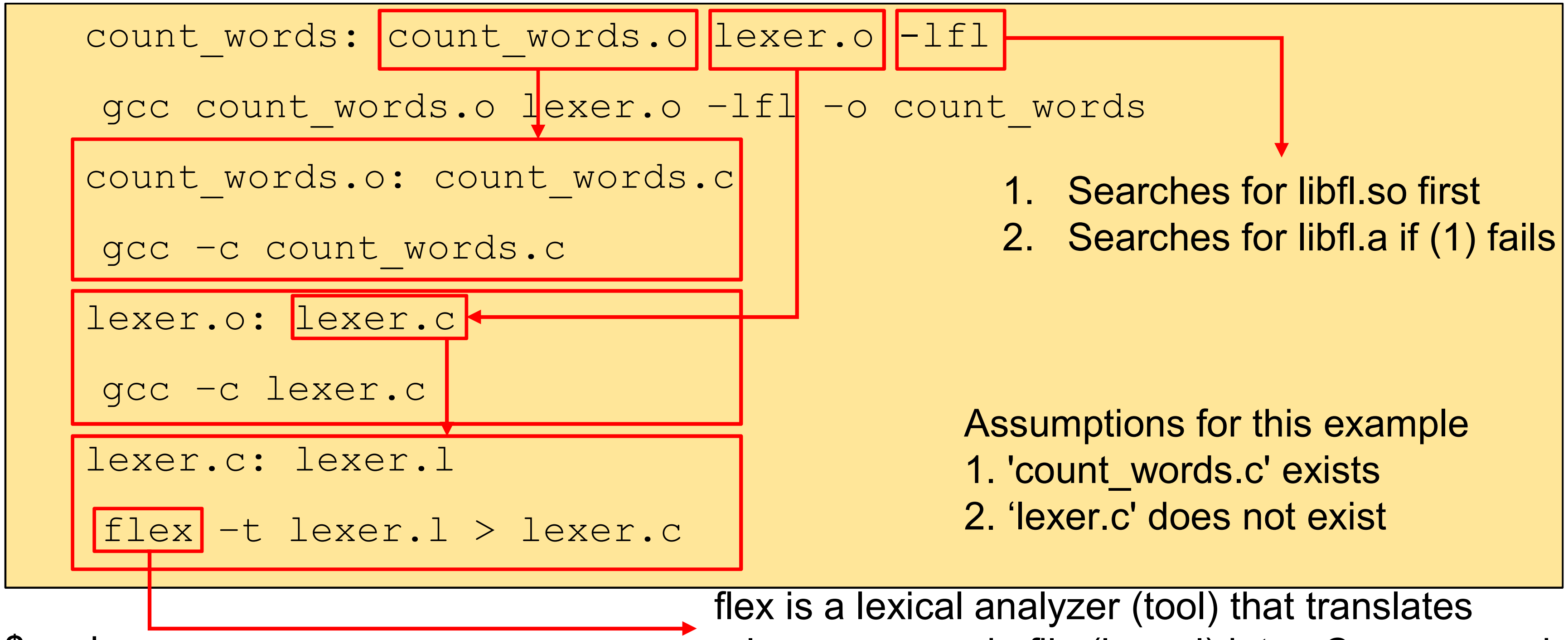
  2. If the prerequisites have associated rules, evaluate them recursively.

```
foo.o: foo.c foo.h
    gcc -c foo.c
```

  3. If any prerequisite is ***newer*** (modified more recently) than the target, the target is rebuilt by executing the command(s).

     - Each command line is run in its own shell.
     - If any command fails, building of the target terminates and make exits.

An Example for Evaluating Rules



```
count_words: count_words.o lexer.o -lfl
  gcc count_words.o lexer.o -lfl -o count_words
count_words.o: count_words.c
  gcc -c count_words.c
lexer.o: lexer.c
  gcc -c lexer.c
lexer.c: lexer.l
  flex -t lexer.l > lexer.c
```

1. Searches for libfl.so first
2. Searches for libfl.a if (1) fails

Assumptions for this example
1. 'count_words.c' exists
2. ‘lexer.c' does not exist

flex is a lexical analyzer (tool) that translates a lex source code file (lexer.l) intoa C source code

- $ make

```
gcc -c count_words.c
flex -t lexer.l > lexer.c
gcc -c lexer.c
gcc count_words.o lexer.o -lfl -o count_words
```

A Few Tips on Rules

- A rule can have multiple targets.
  - Each target has the same set of prerequisites.

```
vpath.o variable.o: make.h config.h getopt.h gettext.h dep.h
```

```
vpath.o: make.h config.h getopt.h gettext.h dep.h
variable.o: make.h config.h getopt.h gettext.h dep.h
```

- Not all prerequisites need to be defined ***all at once.***

```
vpath.o: make.h config.h getopt.h gettext.h dep.h
vapth.o: filedep.h hash.h job.h commands.h variable.h vapth.h
```

```
vpath.o: lexer.c
…
vpath.o: vpath.c
```

Whenever lexer.c is updated, vpath.o must be updated.
The prerequisite is always updated before the target is updated.

Wildcards and Variables

- `make` supports wildcards. (*, ~, ?, […], [^…])
  - *: replaced with all file names in the current directory. (*.c: all '.c' files in current directory)
  - ~: replaced with the home directory.
- `$(variable-name)` : expand the variable whose name is variable-name.
  - Format: variable-name = value
  - Variables can contain almost any text
- Automatic variables
  - **$@**: the filename that represents the target.
  - $<: The filename of the first prerequisite.
  - $? : the names of all prerequisites newer than the target, separated by spaces.
  - **$^**: the filename of all the prerequisites, separated by spaces. (with duplicate files removed)
  - $+: Similar to $^, but allows duplicate files.
  - $*: the stem of the target filename, "stem" = a filename without its suffix.

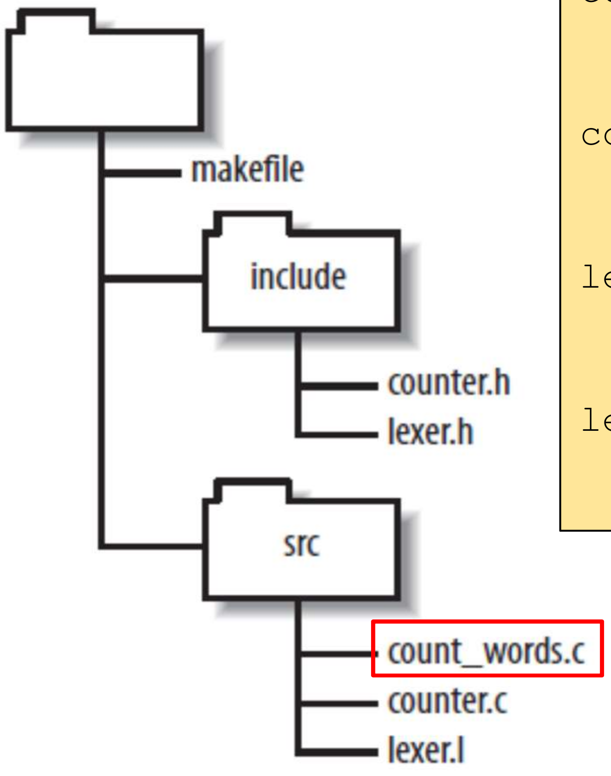
```
CC = gcc
prog: *.c
    $(CC) -o $@ $^
```

gcc –o prog *.c

Practicing with Wildcards

```
count_words: count_words.o counter.o lexer.o -lfl
    gcc $^ -o $@        →  gcc count_words.o counter.o lexer.o –lfl –o count_words
count_words.o: count_words.c
    gcc -c $<           →  gcc –c count_words.c
counter.o: counter.c
    gcc -c $<           →  gcc –c counter.c
lexer.o: lexer.c
    gcc -c $<           →  gcc –c lexer.c
lexer.c: lexer.l
    flex -t $< > $@     →  flex –t lexer.l > lexer.c
```

Files in Different Directories?



```
count_words: count_words.o counter.o lexer.o -lfl
    gcc $^ -o $@
count_words.o: count_words.c include/counter.h
    gcc -c $<
counter.o: counter.c include/counter.h include/lexer.h
    gcc -c $<
lexer.o: lexer.c include/lexer.h
    gcc -c $<
lexer.c: lexer.l
    flex -t $< > $@
```

```
$ make
make: *** No rule to make target 'count_words.c’,
needed by 'count_words.o’. Stop.
```

How to fix this problem?

Why? ‘count_words.c‘ is not found in the current directory (it’s in src/) nor there is a rule to build the file. The error messages refers to the latter.

VPATH & CPPFLAGS

```
VPATH = src
CPPFLAGS = -I include
count_words: count_words.o counter.o lexer.o -lfl
    gcc $(CPPFLAGS) $^ -o $@
count_words.o: count_words.c include/counter.h
    gcc $(CPPFLAGS) -c $<
counter.o: counter.c include/counter.h include/lexer.h
    gcc $(CPPFLAGS) -c $<
lexer.o: lexer.c include/lexer.h
    gcc $(CPPFLAGS) -c $<
lexer.c: lexer.l
    flex -t $< > $@
```

- `VPATH`: environment variable. tells make where to look for the files.
- `CPPFLAGS`: provides an option to gcc for finding the header files.
  - `CPPFLAGS`: preprocessor options
  - `CFLAGS`: c compiler options
- `vpath` directive – more precise

```
vpath %.c src
vpath %.l src
vpath %.h include
```

  - Look for .c or .l files in “src/”.
  - Look for .h in “include/” .
    - No need for include/X in prerequisites.

Phony Targets

- Phony target: a target that does not represent a file.
  - Always out of date, so always evaluate the rule.

```
clean:
    rm -f *.o lexer.c
```

  - `$ make clean //  always executes 'rm -f *.o lexer.c'.`

- What if there happens to be a file, 'clean', in the current directory?
  - $ make clean

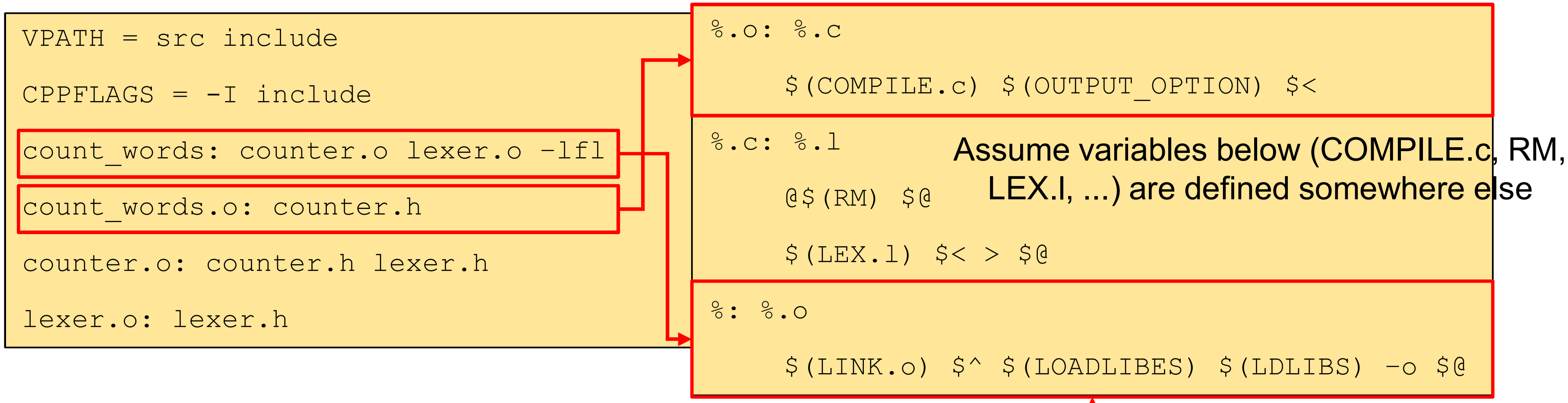
    make: 'clean' is up to date.
  - Problem: make does not know whether a target is phony or not.
  - `.PHONY` to explicitly tell make that 'clean' is a phony target.

```
.PHONY: clean
clean:
    rm -f *.o lexer.c
```

Popular Phony Targets

- Typical phony targets & typical meaning
  - `all`: perform all tasks to build the application.
  - `install`: create an installation of the application from the compiled binaries.
  - `clean`: delete the binary files, temporary files generated from source files.
  - `distclean`: delete all the generated files not in the original source distribution.
  - `TAGS`: create a tag tale for use by editors. (ctags/etags)
  - `info`: create GNU info files from their Texinfo sources.
  - `check`: Run any tests associated with this application.

Pattern Rules



```
VPATH = src include
CPPFLAGS = -I include
count_words: counter.o lexer.o -lfl
count_words.o: counter.h
counter.o: counter.h lexer.h
lexer.o: lexer.h
```

```
%.o: %.c
	$(COMPILE.c) $(OUTPUT_OPTION) $<
%.c: %.l
	@$(RM) $@
	$(LEX.l) $< > $@
%: %.o
	$(LINK.o) $^ $(LOADLIBES) $(LDLIBS) -o $@
```

Assume variables below (COMPILE.c, RM, LEX.l, ...) are defined somewhere else

- Can simplify makefile with built-in rules for well-known file types.
  - Built-in rules = **implicit** pattern rules
  - Pattern rules = normal rules except the **stem** of the file is expressed as %.

a name without a suffix

- The first target expanded by the implicit pattern rule, ‘**%: %.o**’.

```
count_words: count_words.o counter.o lexer.o -lfl
```

  - % = count_words, %.o = count_words.o
- Next target = count_words.o, its rule expanded by ‘**%.o: %.c**’.

```
count_words.o: count_words.c counter.h
```

  - src/count_words.c exists, and it does not have a prerequisite => run the command & move on.

Pattern Rules

```
VPATH = src include
CPPFLAGS = -I include
count_words: counter.o lexer.o -lfl
count_words.o: counter.h
counter.o: counter.h lexer.h
lexer.o: lexer.h
```

```
%.o: %.c
        $(COMPILE.c) $(OUTPUT_OPTION) $<
%.c: %.l
        @$(RM) $@
        $(LEX.l) $< > $@
%: %.o
        $(LINK.o) $^ $(LOADLIBES) $(LDLIBS) -o $@
```

- Next target = counter.o

```
counter.o: counter.c counter.h lexer.h
```

(expanded by **%.o: %.c)**

- Next target = lexer.o (expanded by **%.o: %.c**)

```
lexer.o: lexer.c lexer.h
```

  - But src/lexer.c does **NOT** exis
  - This triggers the rule, '**%.c: %.**

```
lexer.c: lexer.l
    @$(RM) $@
    $(LEX.l) $< > $@
```

Predefined Variables in Built-in Rules

```
%.o: %.c
    $(COMPILE.c) $(OUTPUT_OPTION) $<
%.c: %.l
    @$(RM) $@
    $(LEX.l) $< > $@
%: %.o
    $(LINK.o) $^ $(LOADLIBES) $(LDLIBS) -o $@
```

- `$ make -p // will show all predefined/default variables`
  - COMPILE.c = $(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) –c
  - CC = cc, can be redefined to the path of an alternate C compiler.
  - CFLAGS: options for $(CC) command. none by default.
  - CPPFLAGS: options for cpp. none by default .
  - TARGET_ARCH: architecture-specific options. none by default.
  - LINK.o = $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
  - LDFLAGS: options for ld; none by default.
  - LEX.l = $(LEX) $(LFLAGS) –t
  - LEX = lex
  - LFLAGS = options for lex, none by default.
  - RM = rm –f

Practice Writing Makefile

- Want to build *progK.* (program binary name)
  - C source code consists of fileA.c, fileB.c, and fileC.c
  - fileA.c includes a1.h and a2.h, fileB.c includes b1.h, and fileC.c includes c1.h.
  - All files (.c or .h) can be modified any time.
- Let's write reasonable makfefile.
- Version 1:

```
all:

    gcc -c fileA.c fileB.c fileC.c -o progK
```

- Problem: no prerequisites – don’t recompile even if any files are modified.
- Version 2:

```
all: fileA.c fileB.c fileC.c a1.h a2.h b1.h c1.h
    gcc -c fileA.c fileB.c fileC.c -o progK
```

- Problem? Recompile every file if any prerequisite files are modified.

Practice Writing Makefile (Continued)

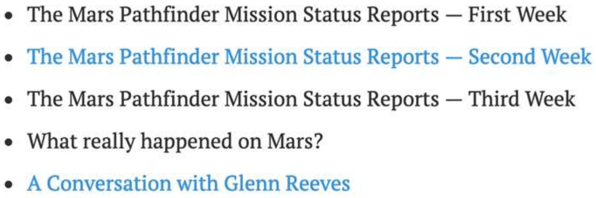
- Version 3:

```
OBJFILES = fileA.o fileB.o fileC.o
CC = gcc
CFLAGS = -Wall –Werror
All: $(OBJFILES)
    $(CC) $(OBJFILES) –o progK
fileA.o: a1.h a2.h
fileB.o: b1.h
fileC.o: c1.h
```

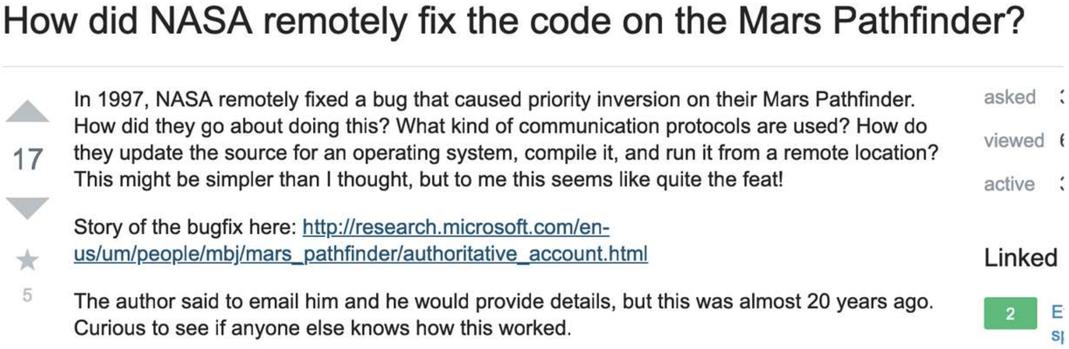
- A reasonable one that does
  - Partial/incremental build.
  - Properly uses built-in pattern rules.
  - Properly overrides CFLAGS with –Wall (print all warnings) –Werror (make all warnings into errors).

Debugging is not easy.

In 1997, Pathfinder on Mars has stopped. OS has crashed due to the priority inversion.



- The Mars Pathfinder Mission Status Reports — First Week
- The Mars Pathfinder Mission Status Reports — Second Week
- The Mars Pathfinder Mission Status Reports — Third Week
- What really happened on Mars?
- A Conversation with Glenn Reeves



How did NASA remotely fix the code on the Mars Pathfinder?

In 1997, NASA remotely fixed a bug that caused priority inversion on their Mars Pathfinder. How did they go about doing this? What kind of communication protocols are used? How do they update the source for an operating system, compile it, and run it from a remote location? This might be simpler than I thought, but to me this seems like quite the feat!

Story of the bugfix here: http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/authoritative_account.html

The author said to email him and he would provide details, but this was almost 20 years ago. Curious to see if anyone else knows how this worked.