CMPUT 379 Lab

ETLC E1003: Tuesday, 5:00 – 7:50 PM.

Tianyu Zhang, Peiran Yao

CAB 311: Thursday, 2:00 – 4:50 PM.

Max Ellis, Aidan Bush

Today’s lab

- Get familiar with the programming environment
- Basic steps for Unix system programming
- System calls used in Assignment 1

Getting started with the programming environment

Producer: Do you have a portable setup?

Me: Yup! See you tomorrow!

CS Linux machines

- Assignments will be graded on these machines
- Make sure your programs can be compiled and run
- Help desk at CSC 1F

1. The full list of servers is on eClass.

Servers:

ugXX.cs.ualberta.ca (XX must be between 00 and 34)

uiXX.cs.ualberta.ca (XX must be between 00 and 22)

udXX.cs.ualberta.ca (XX must be between 00 and 26)

ueXX.cs.ualberta.ca (XX must be between 00 and 26)

ufXX.cs.ualberta.ca (XX must be between 00 and 25)

ucXX.cs.ualberta.ca (XX must be between 01 and 16)

umXX.cs.ualberta.ca (XX must be between 00 and 24)

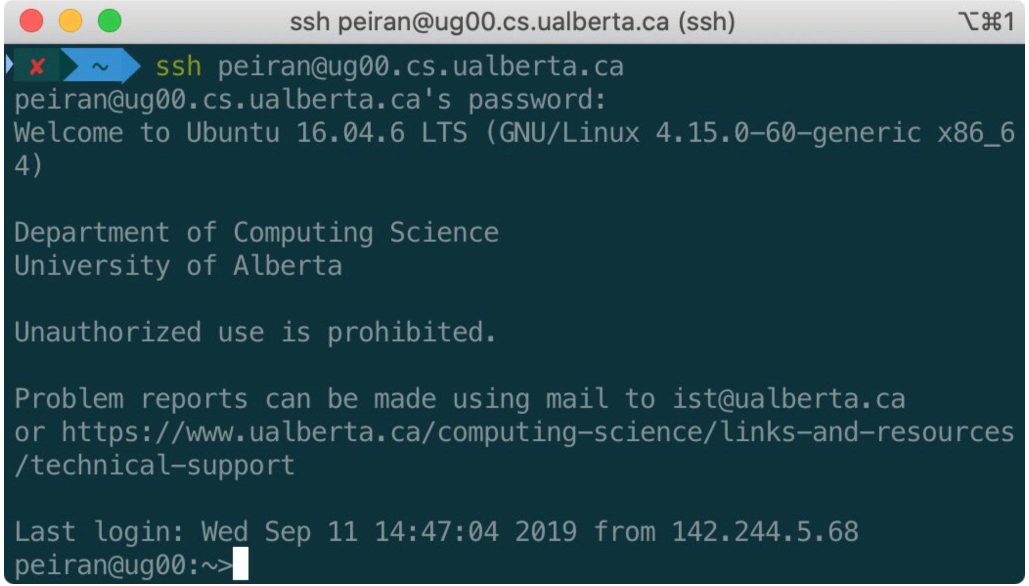
Additionally you might login to and use these general purpose servers:

ohaton.cs.ualberta.ca

innisfree.cs.ualberta.ca

Connect to CS Linux machines with ssh

- Open your terminal simulator
- Run `ssh` [`CSID@uYXX.cs.ualberta.ca`](CSID@uYXX.cs.ualberta.ca)
- Enter the password for your CSID
- Type in **`exit`** to exit



```
 ✘  ~  ssh peiran@ug00.cs.ualberta.ca
peiran@ug00.cs.ualberta.ca's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-60-generic x86_6
4)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources
/technical-support

Last login: Wed Sep 11 14:47:04 2019 from 142.244.5.68
peiran@ug00:~>
```

Alternatively if you use Windows

You can use PuTTY / MobaXTerm

1. https://www.putty.org
2. https://mobaxterm.mobatek.net

Ubuntu virtual machine image

- In case you don't have a Unix environment
- Download the image from eClass and follow the instructions
- **TEST YOUR FINAL PROGRAM ON CS LINUX MACHINES**

1. https://eclass.srv.ualberta.ca/pluginfile.php/5312238/mod_resource/content/1/CS%20ualberta%20VM%20installation..pdf

Transfer files between remote & local machines

Try me.

- On your own computer, run

  ```
  scp -r awesome_code.c CSID@uYXX.cs.ualberta.ca:my_code.c
  ```

  to upload files to the home directory of remote machines

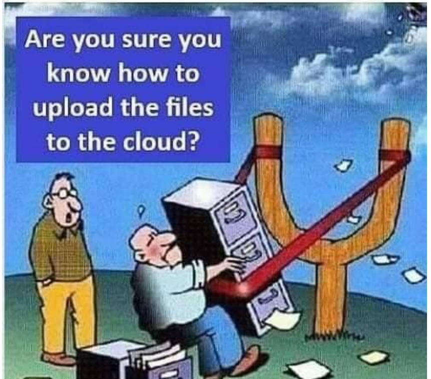
- On your own computer, run

  ```
  scp -r CSID@uYXX.cs.ualberta.ca:results.txt some_results.txt
  ```

  to download files from remote machines

Alternatively you can use **`rsync`**

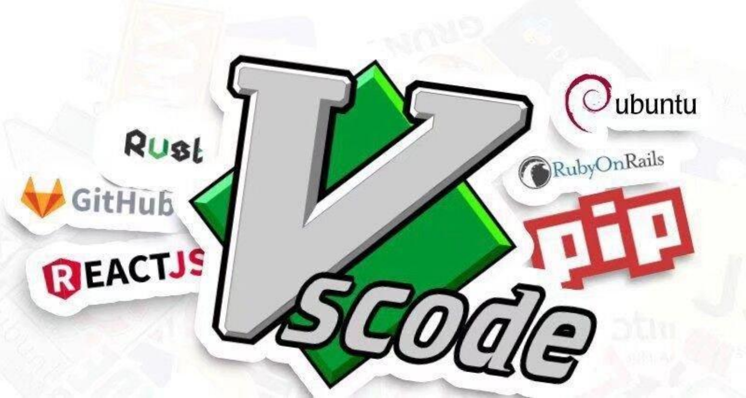
1. https://linux.die.net/man/1/rsync



FYI: Editing files on remote machines

- Text editors like **Vim, Emacs, Nano**
- VS Code Remote Development
  - Modern & elegant GUI application
  - Runs on your own machine!
  - Manage and edit yours remote files through SSH

1. https://danielmiessler.com/study/vim/
2. https://hackaday.com/2016/08/08/editor-wars-the-revenge-of-vim/
3. https://code.visualstudio.com/docs/remote/remote-overview



FYI: Useful tools for development

- Make your sessions persist after you close the SSH connection with **tmux**
- Use **git** to keep track of your code

1. https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/
2. https://git-scm.com/book/en/v2

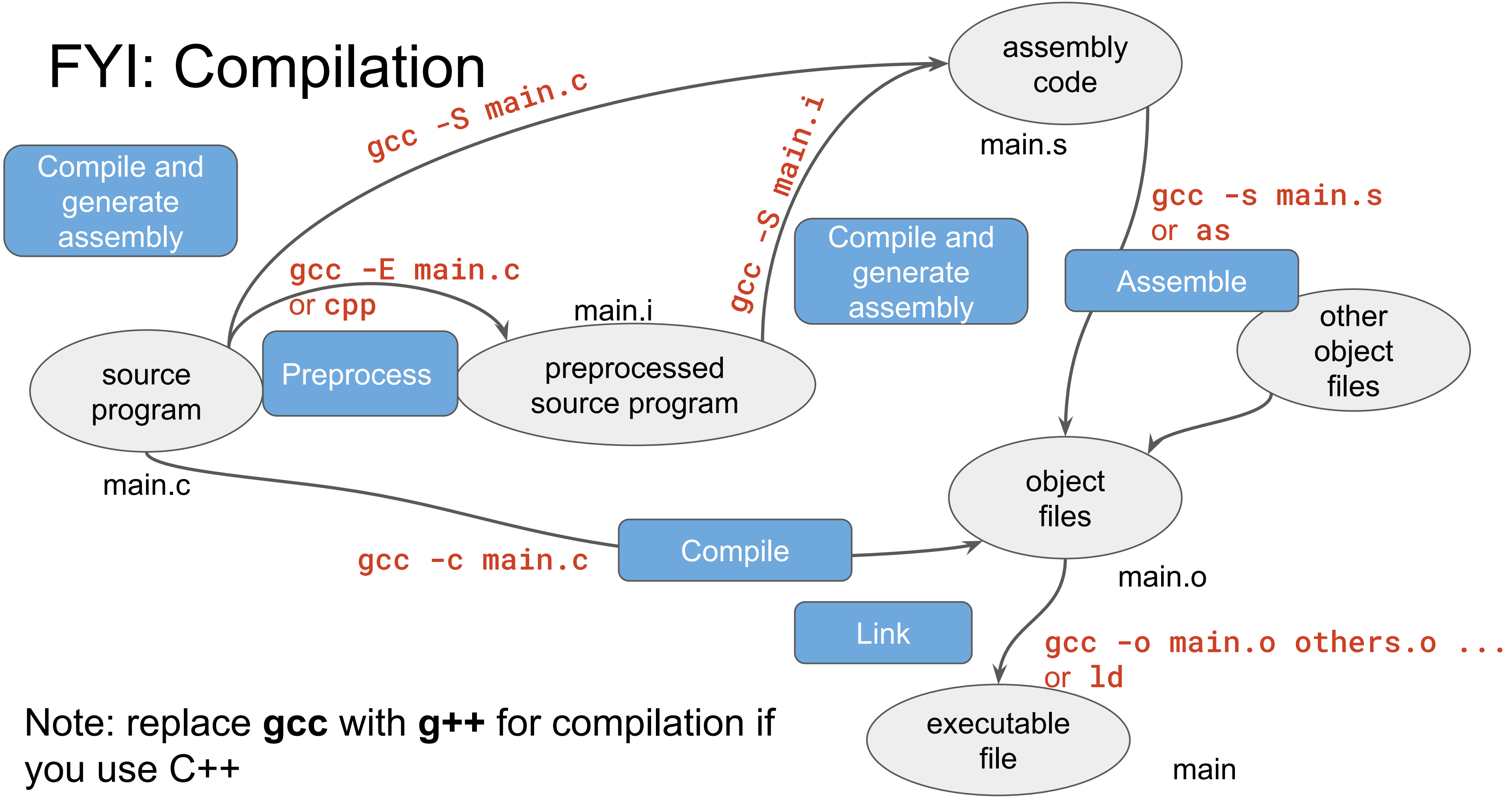
Compression

- You'll need to compress your code to submit it
  - `zip my_code.zip *.c *.h`
  - `tar -caf my_code.tar.gz *.c *.h`

- 🚨 **Type the name of your compressed file first!!!**
- 😭 **Otherwise, you lost your first file**
- **Renaming files does not change their format**

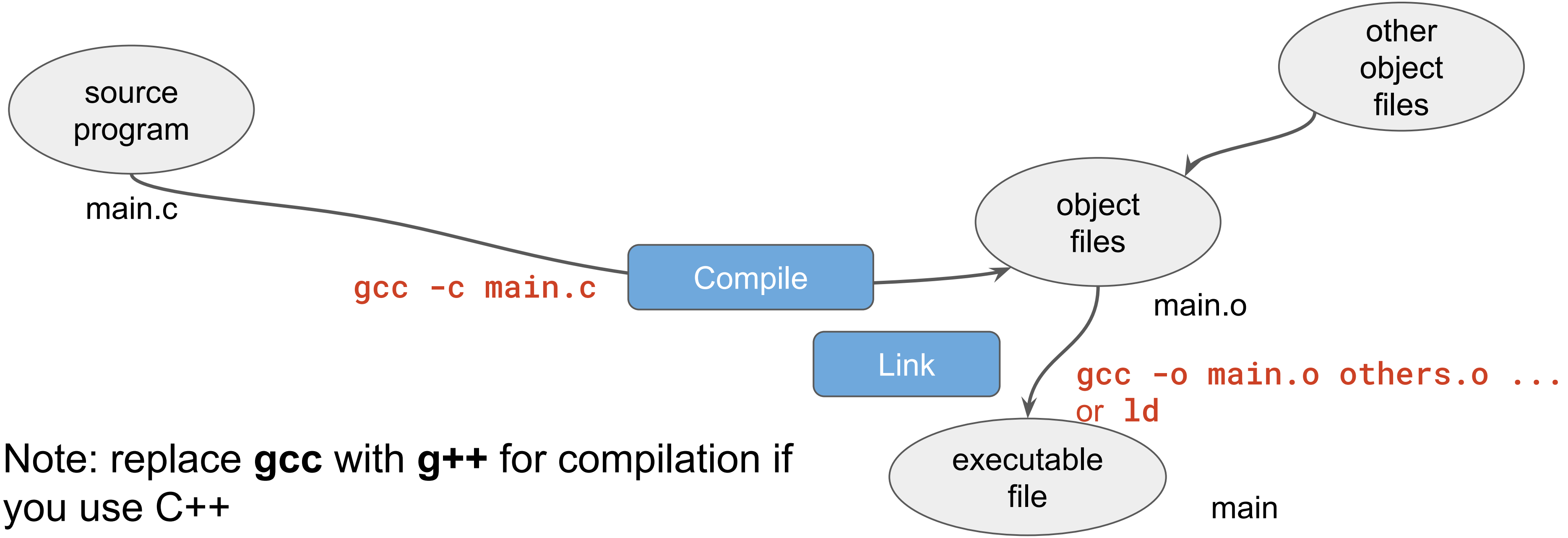
UNIX programming 101

FYI: Compilation



Note: replace **gcc** with **g++** for compilation if you use C++

Compilation - simplified!



source program

main.c

`gcc -c main.c`

Compile

object files

main.o

other object files

Link

`gcc -o main.o others.o ...`
or `ld`

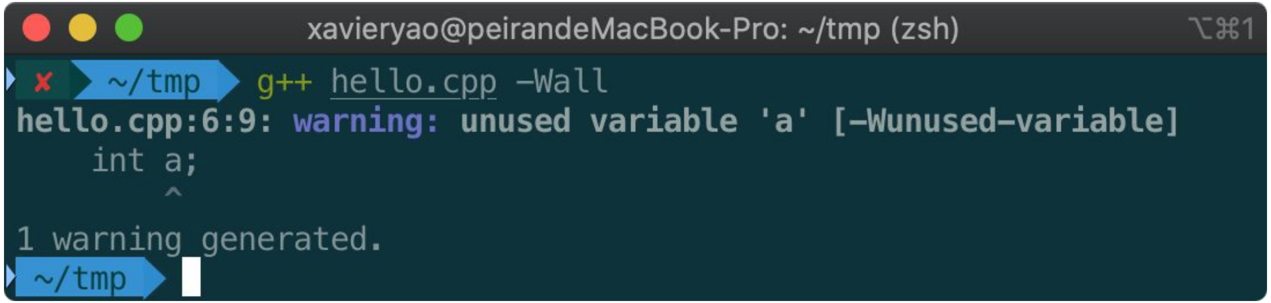
executable file

main

Note: replace **gcc** with **g++** for compilation if you use C++

Compilation - options

- Use command line options to control the behavior of **gcc**
- **-o [output file name]** output file name (create executable if -c not specified)
- **-c [output file name]** create an object file
- **-O** optimize your code
  - Different levels: **-O1 -O2 -O3 -Ofast**
- **-g** keep debugging information
- **-Wall** adds most warnings
- **-Wextra**
- **-Wno…** don't use
- Example:



```
✘  ~/tmp  g++ hello.cpp -Wall
hello.cpp:6:9: warning: unused variable 'a' [-Wunused-variable]
    int a;
        ^
1 warning generated.
 ~/tmp  
```

Compilation - compile and link

- Compilation and linking can actually be combined together (DON’T DO THIS)

  -
    ```
    gcc *.c *.h -o main
    ```
- But some files don’t need to be recompiled
- Object files can be reused / shared
- Use **make** to help you automate this process - covered later

Compilation - compiler version

- **gcc** 5.4.0 on CS Linux machines
- Supports C++11 & C++14 & C99
- You can use **clang**, but make sure your code can compile using **gcc** on CS Linux machines (clang may be aliased as gcc on MacOS)

1. https://www.gnu.org/software/gcc/projects/cxx-status.html

Make - Introduction

- **Make**: a tool to automate compilation
- **REQUIRED FOR ASSIGNMENTS** (CMake also acceptable, covered later)
- When properly setup it should only recompile outdated files
- You’ll need a **Makefile** in your project folder

1. https://opensource.com/article/18/8/what-how-makefile
2. http://nuclear.mutantstargoat.com/articles/make/

make - Makefile basics

- Hello world in Makefile

```
say_hello:
    echo "Hello World"
```

- Run it in shell

```
$ make
echo "Hello World"
Hello World
```

make - Makefile basics

- Syntax to define rules

```
target: prerequisites
<TAB> recipe
```

- Run it in shell

```
$ make <target>
```

make - Makefile basics

- When we run `make <target>` in the shell, `make` will
  - Check the dependencies of the target
    - If any of the dependencies have been modified since the last time you the **target** was generated
      - Run the recipe line by line

- You need at least 4 targets in your Makefile for your assignments
  - compile, link, clean, compress

make - Makefile basics

- Dependencies can be another rule’s target!
- Putting them all together

```
code_piece1.o: code_piece1.c
    gcc -c code_piece1.c -o code_piece1.o

code_piece2.o: code_piece2.c
    gcc -c code_piece2.c -o code_piece2.o

awesome_app: code_piece1.o code_piece2.o
    gcc -o awesome_app code_piece1.o code_piece2.o
```

make - Variables

Define and assign variables

```
CC      = gcc
CFLAGS  = -Wall -O2
OBJECTS = code_piece1.o code_piece2.o

code_piece1.o: code_piece1.c
    $(CC) $(CFLAGS) -c code_piece1.c -o code_piece1.o

code_piece2.o: code_piece2.c
    $(CC) $(CFLAGS) -c code_piece2.c -o code_piece2.o

awesome_app: $(OBJECTS)
    $(CC) -o awesome_app $(OBJECTS)
```

Use variables

Equivalent to

```
    gcc -Wall -O2 -c code_piece1.c -o code_piece1.o
```

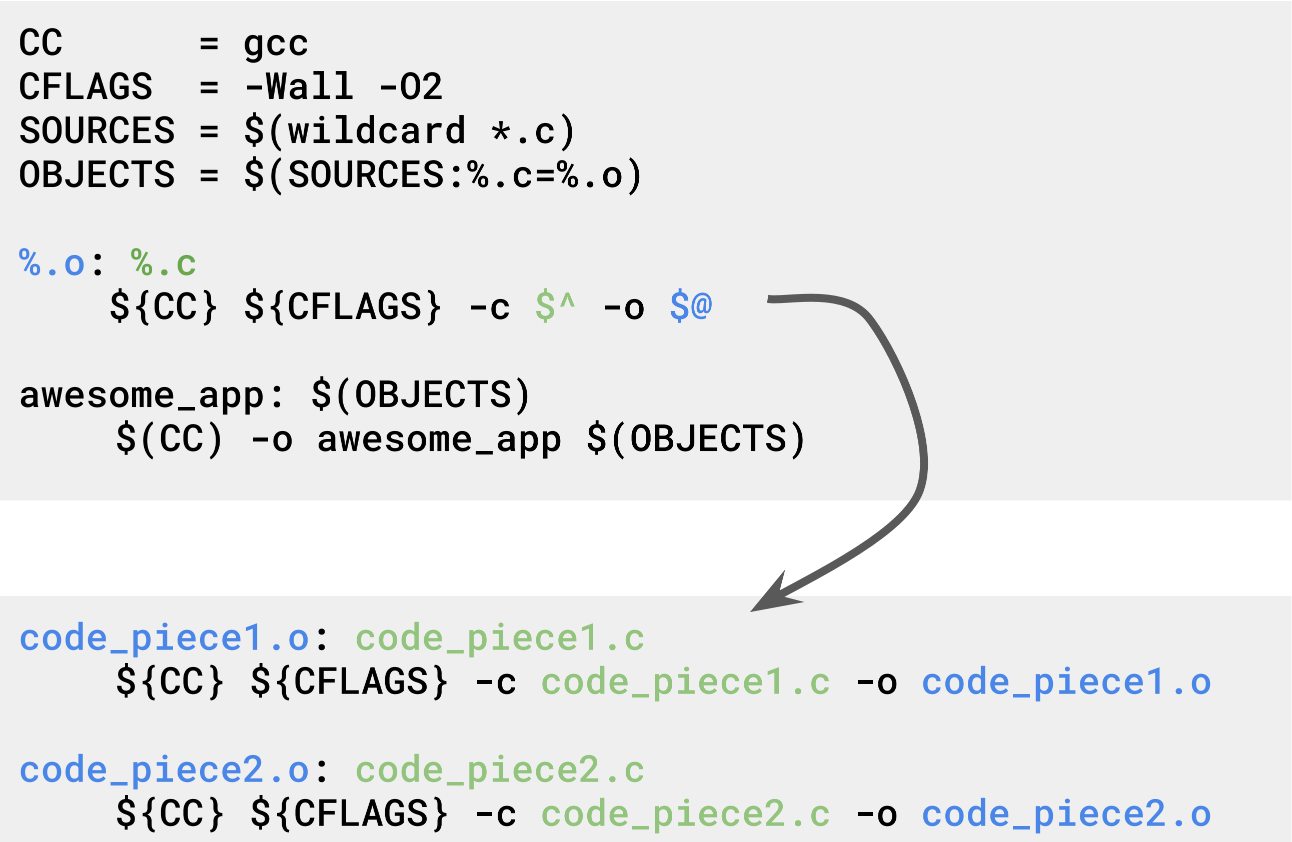
make - Patterns and functions

```
CC      = gcc
CFLAGS  = -Wall -O2
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:%.c=%.o)

code_piece1.o: code_piece1.c
    $(CC) $(CFLAGS) -c code_piece1.c -o code_piece1.o

code_piece2.o: code_piece2.c
    $(CC) $(CFLAGS) -c code_piece2.c -o code_piece2.o

awesome_app: $(OBJECTS)
    $(CC) -o awesome_app $(OBJECTS)
```

- `$(wildcard *.c)` is a function that matches any file ending with .c
- `$(SOURCES:%.c=%.o)` substitutes all file names in **SOURCES** that ends with **.c** to **.o**

```
SOURCES = code_piece1.c code_piece2.c
OBJECTS = code_piece1.o code_piece2.o
```

make - Automatic variables



```
CC      = gcc
CFLAGS  = -Wall -O2
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:%.c=%.o)

%.o: %.c
    ${CC} ${CFLAGS} -c $^ -o $@

awesome_app: $(OBJECTS)
    $(CC) -o awesome_app $(OBJECTS)
```

```
code_piece1.o: code_piece1.c
    ${CC} ${CFLAGS} -c code_piece1.c -o code_piece1.o

code_piece2.o: code_piece2.c
    ${CC} ${CFLAGS} -c code_piece2.c -o code_piece2.o
```

- `%.o` matches any file ending with **.o**
- `%.c` matches any file ending with **.c**
- `$^` is replaced by the name of all the prerequisites
- `$@` is replaced by the name of the target

[1] https://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html

make - Phony targets

```
CC      = gcc
CFLAGS  = -Wall -O2
SOURCES = $(wildcard *.c)
OBJECTS = $(SOURCES:%.c=%.o)

.PHONY: all clean

all: awesome_app

clean:
    rm *.o awesome_app

%.o: %.c
    ${CC} ${CFLAGS} -c $< -o $@

awesome_app: $(OBJECTS)
    $(CC) -o awesome_app $(OBJECTS)
```

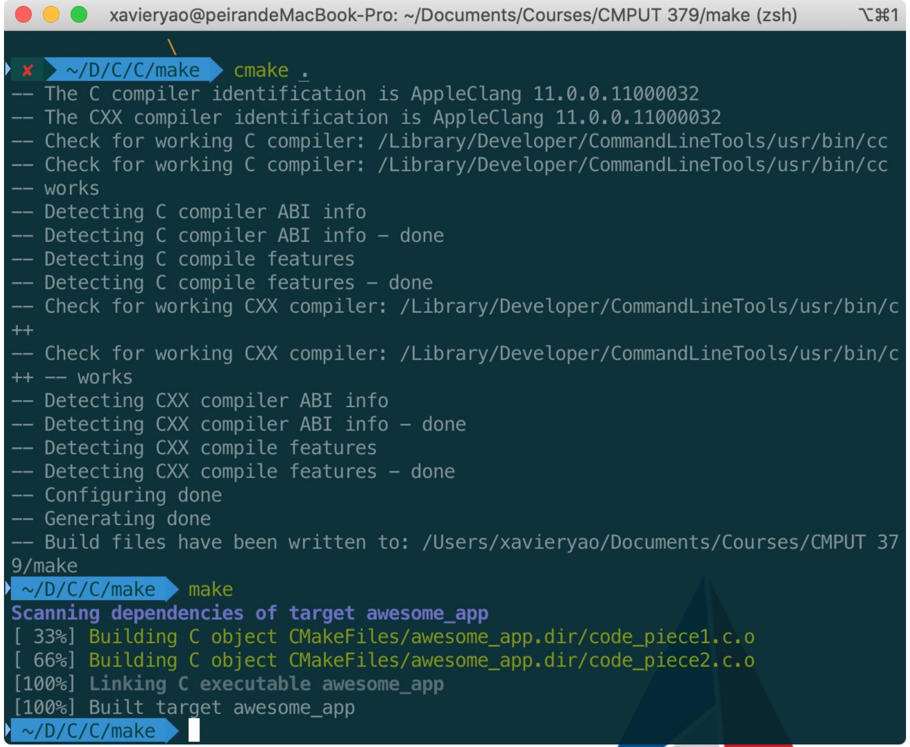
- You can have non-file targets - put them in **.PHONY**
- **all**: conventional entry point
- **clean**: clean all files generated by make
  - **REQUIRED FOR ASSIGNMENTS**

[1] https://www.gnu.org/software/make/manual/html_node/Standard-Targets.html

FYI: CMake

- A tool that can manage dependencies and generate Makefiles

```
cmake_minimum_required(VERSION 3.5)

project(awesome_app)

set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O2
-Wall")

file(GLOB SOURCES *.c)
add_executable(awesome_app ${SOURCES})
```



xavieryao@peirandeMacBook-Pro: ~/Documents/Courses/CMPUT 379/make (zsh)
⌥⌘1

```
              \
 ✘  ~/D/C/C/make  cmake .
-- The C compiler identification is AppleClang 11.0.0.11000032
-- The CXX compiler identification is AppleClang 11.0.0.11000032
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc
-- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c
++
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c
++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/xavieryao/Documents/Courses/CMPUT 37
9/make
 ~/D/C/C/make  make
Scanning dependencies of target awesome_app
[ 33%] Building C object CMakeFiles/awesome_app.dir/code_piece1.c.o
[ 66%] Building C object CMakeFiles/awesome_app.dir/code_piece2.c.o
[100%] Linking C executable awesome_app
[100%] Built target awesome_app
 ~/D/C/C/make 
```

1. https://cmake.org
2. https://github.com/ttroy50/cmake-examples/tree/master/01-basic

GDB

- Debug your program
- Use it to find errors that hard to address
- Need to add flag “-g” when compile your program

```
$(CC) $(CFLAGS) your_c.c -o output -g
```

- Then, use gdb by:

```
gdb ./output
```

- Important commands:

run / continue / next / step / until / print / call / quit / break + line # / etc..

GDB example

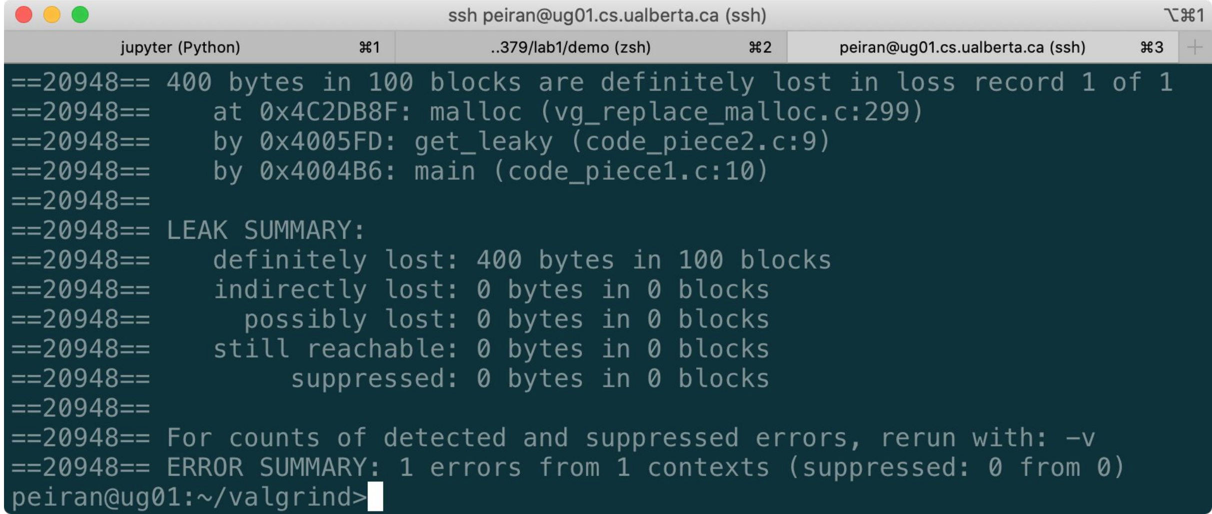
```c
#include <stdio.h>
int foo = 0;

void mystery(void) {
    foo++;
    if (foo == 19234) {
        int *p = 0;
        *p = 0;
    }
}
```

```c
int fib(int n) {
    mystery();
    if (n < 2) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}

int main(int argc, char *argv[]) {
    printf("%d\n", fib(20));
}
```

Valgrind

- Check the memory leak of your program
- Need to add flag “-g” if you want line numbers of where problem originate
- Use it to check after generate your executable file

```
valgrind --leak-check=yes ./your_prog arg1 arg2
```


ssh peiran@ug01.cs.ualberta.ca (ssh)
⌥⌘1
jupyter (Python)
⌘1
..379/lab1/demo (zsh)
⌘2
peiran@ug01.cs.ualberta.ca (ssh)
⌘3

```
==20948== 400 bytes in 100 blocks are definitely lost in loss record 1 of 1
==20948==    at 0x4C2DB8F: malloc (vg_replace_malloc.c:299)
==20948==    by 0x4005FD: get_leaky (code_piece2.c:9)
==20948==    by 0x4004B6: main (code_piece1.c:10)
==20948==
==20948== LEAK SUMMARY:
==20948==    definitely lost: 400 bytes in 100 blocks
==20948==    indirectly lost: 0 bytes in 0 blocks
==20948==      possibly lost: 0 bytes in 0 blocks
==20948==    still reachable: 0 bytes in 0 blocks
==20948==         suppressed: 0 bytes in 0 blocks
==20948==
==20948== For counts of detected and suppressed errors, rerun with: -v
==20948== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
peiran@ug01:~/valgrind>
```

Valgrind example

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int *p = (int *)malloc(50 * sizeof(int));
    printf("%d\n", p[0]);

    return 0;
}
```

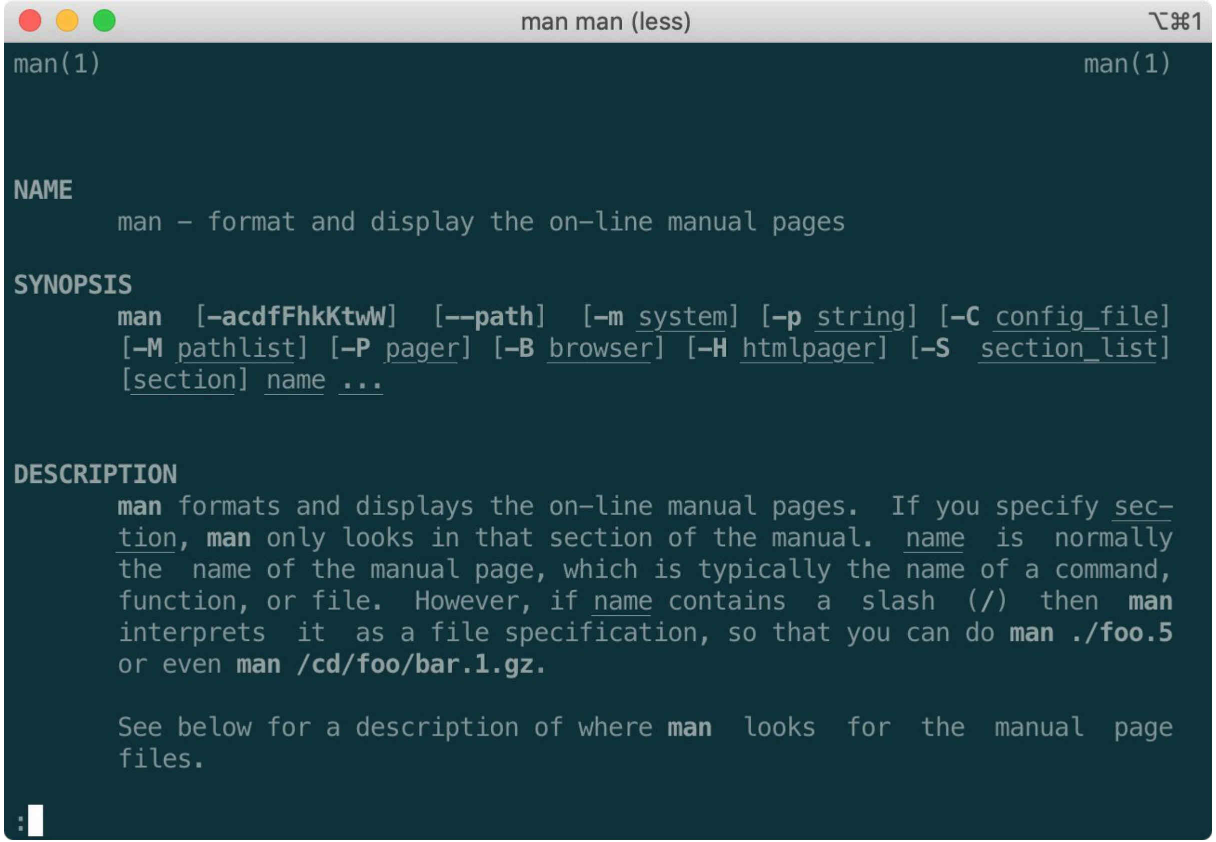
Valgrind example

```
==10765== HEAP SUMMARY:
==10765==     in use at exit: 200 bytes in 1 blocks
==10765==   total heap usage: 2 allocs, 1 frees, 1,224 bytes allocated
==10765==
==10765== LEAK SUMMARY:
==10765==    definitely lost: 200 bytes in 1 blocks
==10765==    indirectly lost: 0 bytes in 0 blocks
==10765==      possibly lost: 0 bytes in 0 blocks
==10765==    still reachable: 0 bytes in 0 blocks
==10765==         suppressed: 0 bytes in 0 blocks
==10765== Rerun with --leak-check=full to see details of leaked memory
==10765==
==10765== For counts of detected and suppressed errors, rerun with: -v
==10765== Use --track-origins=yes to see where uninitialised values come from
==10765== ERROR SUMMARY: 5 errors from 5 contexts (suppressed: 0 from 0)
```

Get help - **`man(1)`**

- Synopsis: man [section] name
- Sections
  - 1 user commands (e.g. **`man man`** “What is man and how to use it”)
  - 2 system calls (e.g. **`man 2 open`** “How to open a fd”)
  - 3 C standard library (e.g. **`man 3 printf`** “How to Hello World”)
  - 7 miscellaneous (e.g. **`man 7 signal`** “What are all the Linux signals”)

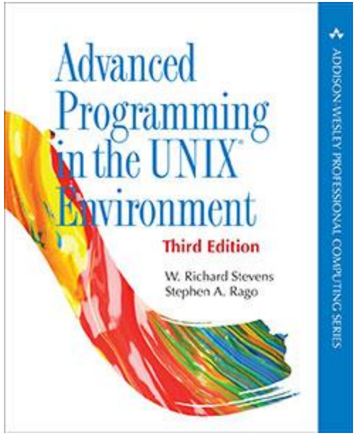
Get help - **man(1)**



```
man(1)                                                                man(1)



NAME
       man - format and display the on-line manual pages

SYNOPSIS
       man  [-acdfFhkKtwW]  [--path]  [-m system] [-p string] [-C config_file]
       [-M pathlist] [-P pager] [-B browser] [-H htmlpager] [-S  section_list]
       [section] name ...


DESCRIPTION
       man formats and displays the on-line manual pages.  If you specify sec-
       tion, man only looks in that section of the manual.  name  is  normally
       the  name of the manual page, which is typically the name of a command,
       function, or file.  However, if name contains  a  slash  (/)  then  man
       interprets  it  as a file specification, so that you can do man ./foo.5
       or even man /cd/foo/bar.1.gz.

       See below for a description of where man  looks  for  the  manual  page
       files.

:
```

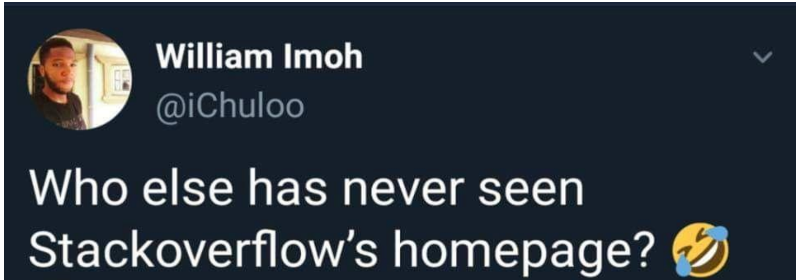
Get help

- Ask TAs during lab sessions
- Post your questions on eClass forums (but do not share code!)
- Send an email to cmput-379-f19@googlegroups.com
- Refer to Advanced Programming in the UNIX® Environment, Third Edition
  - ○ Electronic version available via UofA library


Advanced Programming in the UNIX® Environment
Third Edition
W. Richard Stevens
Stephen A. Rago
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

FYI: Get help - StackOverflow.com

⚠️ **NO PLAGIARISM ALLOWED**



William Imoh
@iChuloo

Who else has never seen Stackoverflow’s homepage? 🤣

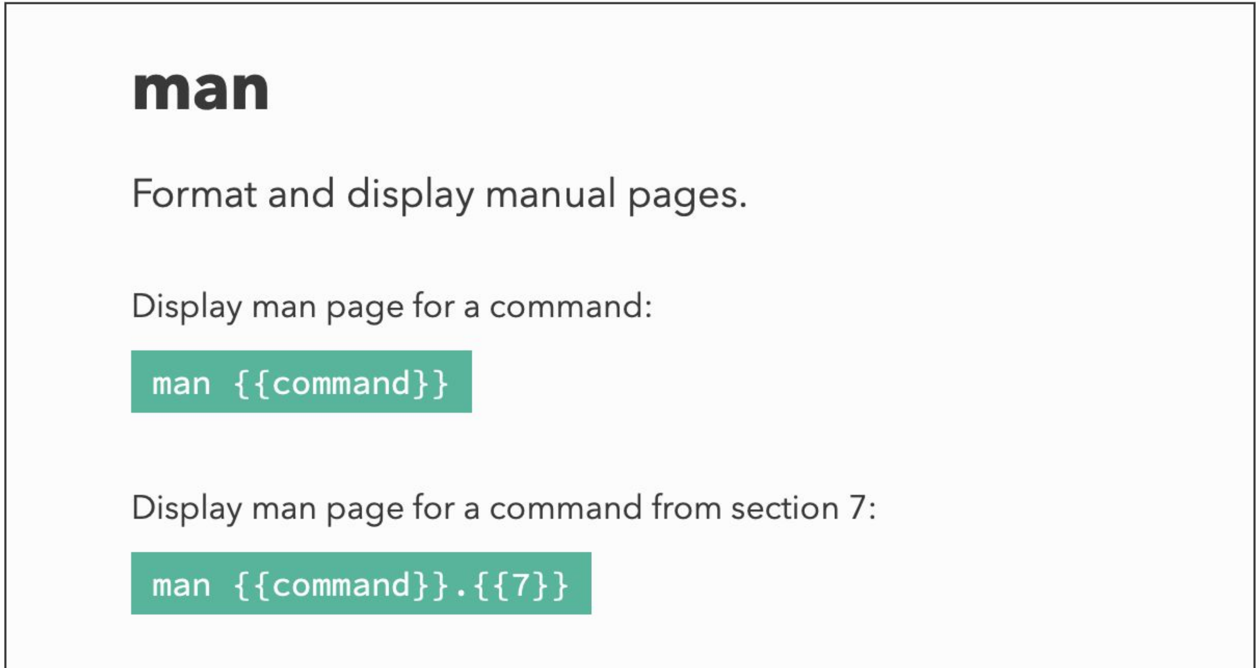
20 Jul 19 · Twitter for iPhone

etweets **626** Likes

Stack Overflow finally released their own keyboard!

FYI: Get help - tldr.sh

- “Take out” version of **man**
- **https://tldr.sh**



man

Format and display manual pages.

Display man page for a command:

```
man {{command}}
```

Display man page for a command from section 7:

```
man {{command}}.{{7}}
```

Tools - objdump

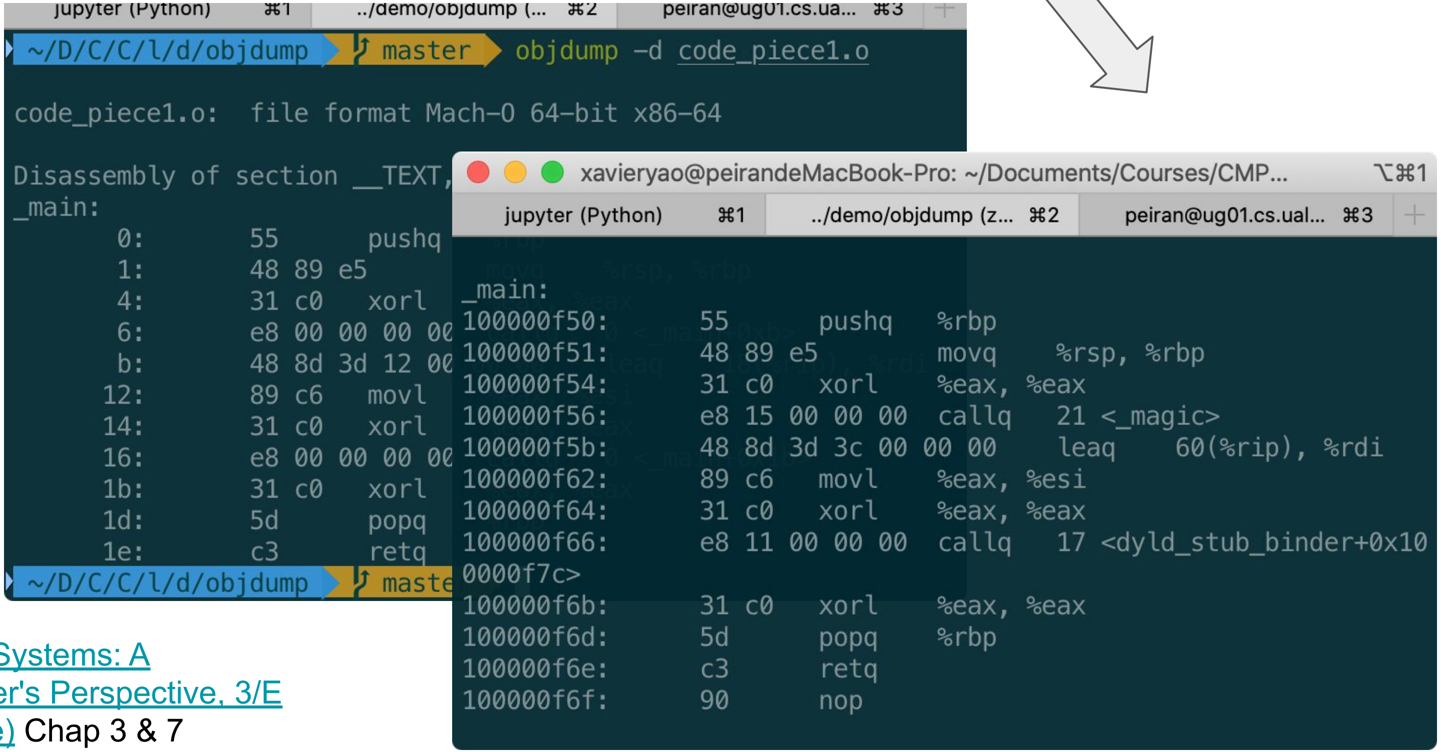
- **objdump** displays information about one or more object files
- **objdump -h ./awesome_app**
- **objdump -D ./code_piece2.o**

1. https://medium.com/@holdengrissett/linux-101-understanding-the-insides-of-your-program-2be2480ba366
2. https://sourceware.org/binutils/docs/binutils/objdump.html

Tools - objdump

- Segment relocation

After linking, addresses are different



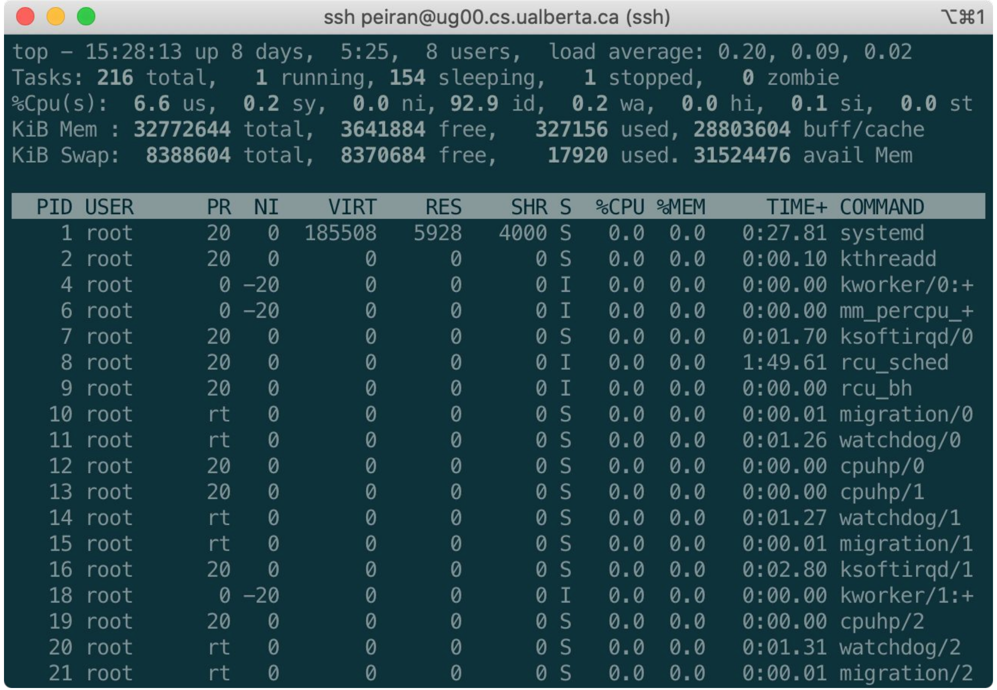
```
~/D/C/C/l/d/objdump  master  objdump -d code_piece1.o

code_piece1.o:  file format Mach-O 64-bit x86-64

Disassembly of section __TEXT,
_main:
       0:       55      pushq
       1:       48 89 e5
       4:       31 c0   xorl
       6:       e8 00 00 00 00
       b:       48 8d 3d 12 00
      12:       89 c6   movl
      14:       31 c0   xorl
      16:       e8 00 00 00 00
      1b:       31 c0   xorl
      1d:       5d      popq
      1e:       c3      retq
~/D/C/C/l/d/objdump  maste
```

```
_main:
100000f50:      55      pushq   %rbp
100000f51:      48 89 e5        movq    %rsp, %rbp
100000f54:      31 c0   xorl    %eax, %eax
100000f56:      e8 15 00 00 00  callq   21 <_magic>
100000f5b:      48 8d 3d 3c 00 00 00    leaq    60(%rip), %rdi
100000f62:      89 c6   movl    %eax, %esi
100000f64:      31 c0   xorl    %eax, %eax
100000f66:      e8 11 00 00 00  callq   17 <dyld_stub_binder+0x10
0000f7c>
100000f6b:      31 c0   xorl    %eax, %eax
100000f6d:      5d      popq    %rbp
100000f6e:      c3      retq
100000f6f:      90      nop
```

1. Computer Systems: A Programmer's Perspective, 3/E (CS:APP3e) Chap 3 & 7

Tools - top / htop / pstree

- **top** - show running processes
- **htop** - fancy version of top
- **pstree** - show running processes as a tree
- **ps** - get list of running processes (-el, -aux)
- All information comes from **/proc** (see demo)



```
top - 15:28:13 up 8 days,  5:25,  8 users,  load average: 0.20, 0.09, 0.02
Tasks: 216 total,   1 running, 154 sleeping,   1 stopped,   0 zombie
%Cpu(s):  6.6 us,  0.2 sy,  0.0 ni, 92.9 id,  0.2 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 32772644 total,  3641884 free,   327156 used, 28803604 buff/cache
KiB Swap:  8388604 total,  8370684 free,    17920 used. 31524476 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0  185508   5928   4000 S   0.0  0.0   0:27.81 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.10 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:+
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_+
    7 root      20   0       0      0      0 S   0.0  0.0   0:01.70 ksoftirqd/0
    8 root      20   0       0      0      0 I   0.0  0.0   1:49.61 rcu_sched
    9 root      20   0       0      0      0 I   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:01.26 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0.0  0.0   0:01.27 watchdog/1
   15 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/1
   16 root      20   0       0      0      0 S   0.0  0.0   0:02.80 ksoftirqd/1
   18 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/1:+
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/2
   20 root      rt   0       0      0      0 S   0.0  0.0   0:01.31 watchdog/2
   21 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/2
```

Tools - time

- **time** - execute and time a program
- **`time ./awesome_app`**

```
$ time ./awesome_app
Hello world. 42
./awesome_app  0.00s user 0.00s system 1% cpu 0.296 total
```

Processes

- **Zombie process**

  Process terminated, but parent process not yet handle it

- **Orphan process**

  Parent process terminated, but the process still running

- **Daemon process**

  Background process that is not directly controlled by user

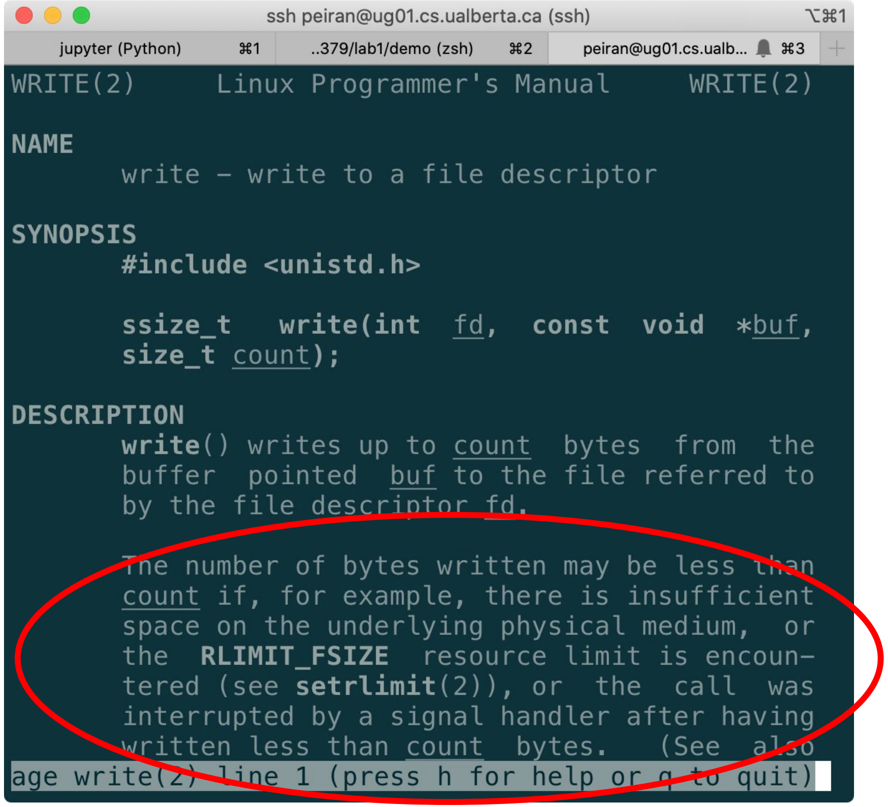
Using system calls

- Assignment 1
    - chdir()
    - fork()
    - execve()
    - _exit()
    - wait()
    - open()
    - close()
    - dup2()
    - pipe()
    - kill()
    - sigaction()

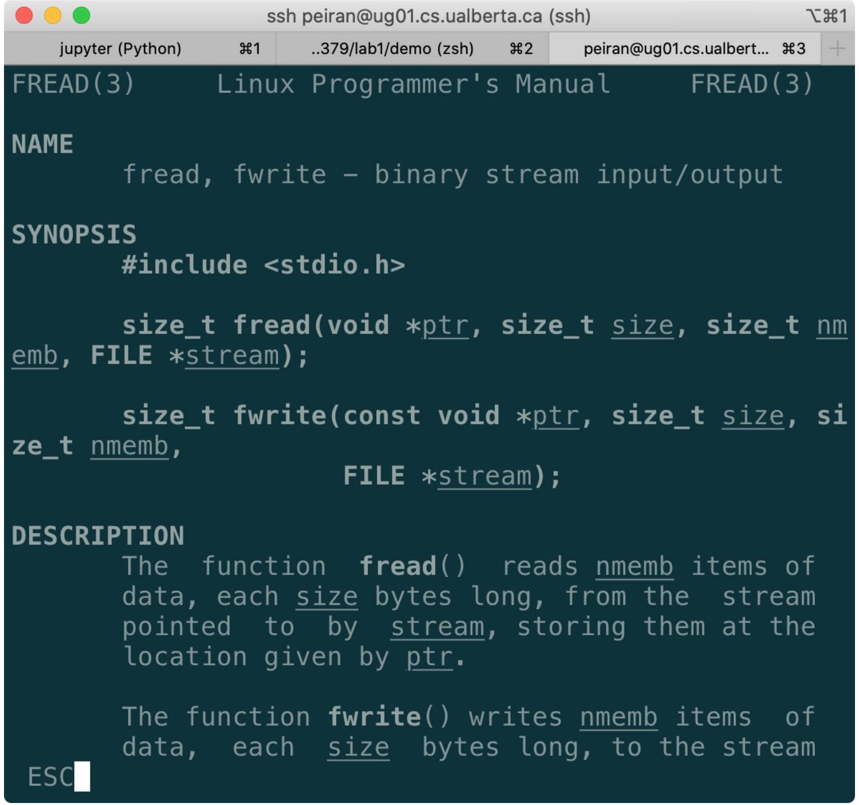
Using system calls: syscall vs. C standard library

- **System calls** (usually) are the standard interfaces that a program can interact with the OS
- **C standard library** provides wrappers of system calls to make them easier to use, plus many other utility functions.

Using system calls: syscall vs. C standard library


ssh peiran@ug01.cs.ualberta.ca (ssh)
⌥⌘1
jupyter (Python) ⌘1
..379/lab1/demo (zsh) ⌘2
peiran@ug01.cs.ualb... ⌘3

```
WRITE(2)      Linux Programmer's Manual      WRITE(2)

NAME
       write - write to a file descriptor

SYNOPSIS
       #include <unistd.h>

       ssize_t   write(int  fd,  const  void  *buf,
       size_t count);

DESCRIPTION
       write() writes up to count  bytes  from  the
       buffer  pointed  buf to the file referred to
       by the file descriptor fd.

       The number of bytes written may be less than
       count if, for example, there is insufficient
       space on the underlying physical medium,  or
       the  RLIMIT_FSIZE  resource limit is encoun-
       tered (see setrlimit(2)), or  the  call  was
       interrupted by a signal handler after having
       written less than count  bytes.   (See  also
age write(2) line 1 (press h for help or q to quit)
```


ssh peiran@ug01.cs.ualberta.ca (ssh)
⌥⌘1
jupyter (Python) ⌘1
..379/lab1/demo (zsh) ⌘2
peiran@ug01.cs.ualbert... ⌘3

```
FREAD(3)      Linux Programmer's Manual      FREAD(3)

NAME
       fread, fwrite - binary stream input/output

SYNOPSIS
       #include <stdio.h>

       size_t fread(void *ptr, size_t size, size_t nm
emb, FILE *stream);

       size_t fwrite(const void *ptr, size_t size, si
ze_t nmemb,
                     FILE *stream);

DESCRIPTION
       The  function  fread()  reads nmemb items of
       data, each size bytes long, from the  stream
       pointed  to  by  stream, storing them at the
       location given by ptr.

       The function fwrite() writes nmemb items  of
       data,  each  size  bytes long, to the stream
 ESC
```

Using system calls: syscall vs. C standard library

- For the assignments, you CAN NOT use C standard library version for the system calls mentioned in the previous slide.
- Check **man** section to see if you are using the current function. (See demo)

Using system calls: error handling

- For most system calls, return value < 0 indicates an error
- See **man errno** to see all possible errors.
- Check the variable **errno** to see what the error is. `#include<errno.h>`
- Use **perror()** to print error detail. `#include<stdio.h>`

```
#include <errno.h>
#include <stdio.h>
if (somecall() == -1) {
    perror("somecall");
    if (errno == ...) { ... }
}
```

Assignment 1 - functions

```
#include <unistd.h>

int chdir(const char *path);

// Change current working directory
```

- Parameter:
  - path - which the user want to make the current working directory
- Return Value:
  - 0 - success
  - -1 - an error occurs and **errno** is set appropriately

chdir example

```
#include <stdio.h>
#include <unistd.h>
#include <limits.h>

int main(int argc, char *argv[]) {
    char s[PATH_MAX];

    printf("%s\n", getcwd(s, PATH_MAX));
    chdir("..");
    printf("%s\n", getcwd(s, PATH_MAX));
    return 0;
}
```

Assignment 1 - functions

```
#include <sys/types.h>

#include <unistd.h>

pid_t fork(void);

// Create a new process
```

- Return Value:
  - -1 - creation of a child process was unsuccessful
  - 0 - Returned to the newly created child process
  - >0 - Returned to parent or caller. The value contains process ID of newly created child process

fork example

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

Assignment 1 - functions

```
#include <unistd.h>

int execve(const char *path, char *const argv[], char *const envp[]);

// Replace current process image with a new one
```

- Parameter:
  - path - the path of the file being executed
  - argv - null terminated array of the arguments for the program being executed
  - envp - array of strings, conventionally of the form **key=value**
- Return Value:
  - No return - success
  - -1   - an error occurs and **errno** is set appropriately

execve example

```c
#include <stdio.h>
#include <unistd.h>

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

    char *argv1[]={"ls", "-al", NULL};
    char *envp[]={NULL};
    if (execve("/bin/ls", argv1, envp) ==-1){
        perror("execve");
    }
    printf("End on demo");

    return 0;
}
```

Assignment 1 - functions

```
#include <unistd.h>

int _exit(int status);

// Terminate process and return status to the parent
```

- Parameter:
  - Status - value returned to the parent process

exit example 1

```
#include <stdio.h>
#include <unistd.h>

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

    printf("START");
    fflush(stdout);
    _exit(0);
    printf("End of program");
}
```

exit example 2

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    for (int i = 0; i < 5; i++) {
        if (fork() == 0) {
            printf("Child %d with pid %d\n", i, getpid());
            _exit(0);
        }
    }
    printf("hello\n");
    return 0;
}
```

Assignment 1 - functions

```
#include <sys/types.h>

#include <sys/wait.h>

pid_t wait(int *status);

// Wait until one of its children terminates
```

- Return Value:
  - Terminated process ID - success
  - 0 or -1   - error

wait example

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    for (int i = 0; i < 5; i++) {
        if (fork() == 0) {
            printf("Child %d with pid %d\n", i, getpid());
            exit(0);
        }
    }
    while (wait(NULL) > 0);
    printf("hello\n");
    return 0;
}
```

Assignment 1 - functions

```
#include <sys/stat.h>

#include <fcntl.h>

int open(const char *path, int oflags, mode_t mode);

// Open a file
```

- Parameter:
    - Oflags - O_RDONLY, O_WRONLY,O_RDWR, O_APPEND, O_CREAT, etc.
    - mode   - S_IRUSR, S_IWUSR, S_IXUSR, etc.
- Return Value:
    - -1   - error
    - Others - file descriptor

Assignment 1 - functions

```
#include <unistd.h>

int close(int fildes);

// Close a file
```

- Parameter:
  - Fildes - The file descriptor to be closed
- Return Value:
  - -1 - error
  - 0 - success

File example - **write**

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int a[100];
    for (int i = 0; i < 100; ++i) a[i] = i;
    // create file "data", truncate it, open it for write
    // operation, set user permissions to rw
    int fd = open("data", O_CREAT | O_TRUNC | O_WRONLY, S_IRUSR | S_IWUSR);

    if (fd < 0)
        perror("encountered open error");
    int length = 100 * sizeof(a[0]);
    if (write(fd, a, length) != length)
        perror("encountered write error");
    if (close(fd) < 0)
        perror("encountered close error");
    return 0;
}
```

File example - **read**

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int a[100];
    int fd = open("data", O_RDONLY);

    if (fd < 0)
        perror("encountered open error");
    int length = 100 * sizeof(a[0]);
    if (read(fd, a, length) != length)
        perror("encountered write error");
    if (close(fd) < 0)
        perror("encountered close error");
    for (int i = 0; i < 100; ++i) printf("%d ", i);
    return 0;
}
```

Assignment 1 - functions

```
#include <unistd.h>

int dup2(int fildes, int fildes2);

// Duplicates one file descriptor, making them aliases, and then deleting the old file descriptor
```

- Parameter:
  - Fildes - source file descriptor
  - Fildes2 - target file descriptor
- Return Value:
  - <0   - error
  - Others - second file descriptor

dup2 example

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    int file_desc = open("test.txt", O_CREAT | O_WRONLY);
    if(file_desc < 0)
        printf("Error opening the file\n");
    dup2(file_desc, 1);
    printf("Tester\n");
    close(file_desc);

    return 0;
}
```

Assignment 1 - functions

```
#include <unistd.h>

int pipe(int pipefd[2]);

// creates a unidirectional data channel for interprocess communication
```

- Parameter:
  - Pipefd - two file descriptors, read/write ends of the pipe
- Return Value:
  - -1 - error
  - 0 - success

pipe example

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
    char msg[] = "Tester", buf[100];
    int p[2];

    pipe(p);

    write(p[1], msg, sizeof(msg));
    read(p[0], buf, sizeof(msg));
    printf("%s\n", buf);

    return 0;
}
```

Assignment 1 - functions

```
#include <sys/types.h>

#include <signal.h>

int kill(pid_t pid, int sig);

// Send signal to the target process
```

- Parameter:
  - Pid - target process
  - Sig - signal want to send
- Return Value:
  - 0 - success
  - -1 - error

kill example

```c
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child with pid %d\n", getpid());
        while (1) printf("*");
    } else {
        sleep(1);
        printf("\nParent killing pid %d\n", pid);
        kill(pid, SIGKILL);
    }
    printf("hello\n");
    return 0;
}
```

Practice - process management

Write a program **timer.c** that reads the path of a program from **stdin** (you can use **scanf()** or **cin**), and use **fork()** and **execv()** to run that program located at the path as a subprocess. Use **kill()** to send **SIGKILL** to that subprocess after **1 second** to stop it.

Hint:

```
unsigned int sleep(unsigned int seconds);   // unistd.h
```

Practice 2

Write a program **primes.c** that is supposed to find all primes between a and b (inclusive with maximum b of 40,000,000) **using n processes**, and write the result to a local output **out.txt** with **sorted order** (parent wait until all children finished and then sort). You can safely assume the number of primes < 2,500,000.

Measure the speedup for n=1, 2, 3, 4, 5, 6, 7, 8 on the lab computers for a = 1 and b = 40,000,000 and discuss what the speedup results mean.

Hint:

Use **fork()** to create processes.

Use **pipe()** to send the results back the the main process.