CMPT 300 D100 Operating System I

Lecture 2.2 – Debugging

Dr. Hazra Imran

Spring 2022

Admin notes

- OH information available on canvas Team
- Assignment 1 released on canvas

Step 1: GitHub ID FORM (REQUIRED) - Do it before submitting your Assignment 1- SFU Credential is required.

Step 2: Assignment general guideline + rules - READ IT CAREFULLY ✓ build

Step 3: Read the Assignment 1 instructions ✓

Step 4: A1 -Submission GitHub link to submit your files. ✓

Step 5: Assignment 1 - Reflection Survey + Late claim days ✓ 2 days

System Boot

multiple O's

- OS must be available to hardware to start it
- On powered up, the instruction register is loaded with a predefined memory location
  - Typically, the address of the bootstrap loader in ROM
- Bootstrap loader routine
  - Performs diagnostic tests (Power-on Self Testing)
  - Loads a small piece of code from a fixed location (boot block) on the disk into memory, which loads the rest of the loader from disk, which loads kernel itself

boot menu.

- Examples:
  - Grub, LILO (Linux Loader)

troubleshooting

Operating-System Debugging

vscode
zombie

- **Debugging** is finding and fixing errors, or **bugs**

- OS generate **log files** containing error information
    - Failure of an application can generate **core dump** file capturing memory of the process
    - Operating system failure can generate **crash dump** file containing kernel memory

- Beyond crashes, performance tuning can optimize system performance
    - Sometimes using *trace listings* of activities, recorded for analysis
    - Profiling is periodic sampling of instruction pointer to look for statistical trends

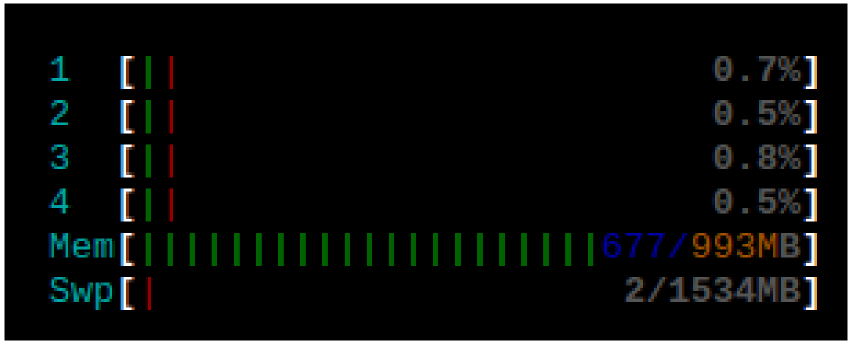
perf $\rightarrow$ Try

OS Debugging and Performance Tuning

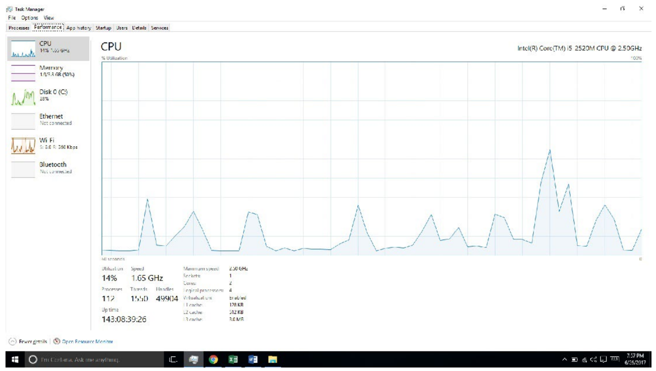
Debugging: finding and fixing errors, or bugs

Tuning: optimize performance by removing bottlenecks

- Common information and approaches
  - Log files containing error information
  - Core dump file capturing the memory of the process
  - Profiling: trace and record activities for later offline analysis
  - Monitoring resource utilization
    - For example, the "top" program or Windows Task Manager



```
1   [||                         0.7%]
2   [||                         0.5%]
3   [||                         0.8%]
4   [||                         0.5%]
Mem[|||||||||||||||||||||677/993MB]
Swp[|                      2/1534MB]
```



Tracing

- Collects data for a specific event, such as steps involved in a system call invocation

- Tools include
  - strace – trace system calls invoked by a process
  - gdb – source-level debugger ✓
  - perf – collection of Linux performance tools
  - tcpdump – collects network packets

read

gcc

GUI CLI

- gcc is the C and C++ compiler developed by GNU project. It is widely adopted as the default compiler of UNIX-like systems.

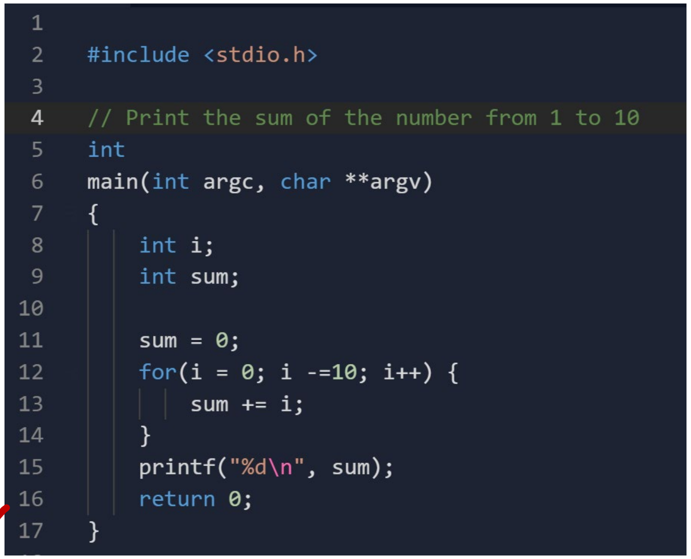
- The basic way of compiling first.c into an executable file called “first" is: gcc -o first first.c

- If the program is compiled without errors, you can execute the program by typing "./first"

- If you are interested in how the assembly code of first.c looks like, you can also generate the assembly code by replacing the "-o first" option with "-S" as gcc -S first.c

first.s

text editor



```c

#include <stdio.h>

// Print the sum of the number from 1 to 10
int
main(int argc, char **argv)
{
    int i;
    int sum;

    sum = 0;
    for(i = 0; i -=10; i++) {
        sum += i;
    }
    printf("%d\n", sum);
    return 0;
}
```

gcc

- In addition to the basic usage, gcc also provides options that help you optimize or debug your code.
- For example, you may :
  - Compile your code with debugging information:
  - ✓ `gcc -g -o first first.c`

- Compile your code with optimizations:
  - `gcc -On -o first first.c`
  - *n* is usually a number from 1 to 3. The larger the number, the more optimizations are performed while compiling the code. The optimization options may differ in each platform.

- For other optimization/debug options, you may use ✓
  - `man gcc`
- under any UNIX-like system.

Debugging using GDB

- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- GDB can step through your source code line-by-line or even instruction by instruction. You may also watch the value of any variable at run-time.
- In addition, it also helps to identify the place and the reason making the program crash.

Segmentation fault

gdb

- All program to be debugged in gdb must be compiled by gcc with the option "-g" turning on.
- If you want to debug the program “first", we can simply start gdb by: gdb ./first
- Then, you will see the gdb environment similar as following:

```
GNU gdb Red Hat Linux (6.3.0.0-1.162.el4rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"..."/home/h1tseng/garbage":
not in executable format: File format not recognized

(gdb)
```

To start running and debugging the program, we can simply type the "run" command after the (gdb) prompt as below: (gdb) run

If the program takes arguments such as "garbage arg_1 arg_2", we may start running the program with these arguments as: (gdb) run arg_1 arg_2 ✓

Debugging using GDB

- Compile source files with –g option to enable debugging
  - -g option tells compiler to put debug information in the object file

Tasks	GDB commands
Compile with debugging symbols	gcc –g firstProg.c –o firstProg
Run programs in GDB	gdb ./firstProg
Exit debugger	quit

Breakpoints

- **Break by line**: break source_filename:line_number
  - (gdb) break first.c:8
  - Breakpoint 1 at 0x1f7b: file first.c, line 8.

- **Break by function:** break source_filename:function_name()
  - (gdb) break first.c:main
  - Breakpoint 1 at 0x1f7b: file first.c, line 8.

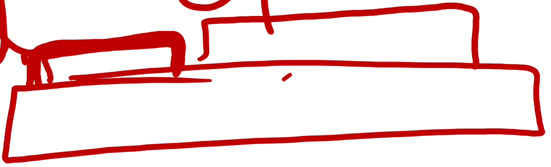
- **Break by instruction:** break *PC
  - (gdb) break *0x1f7b
  - Breakpoint 1 at 0x1f7b: file first.c, line 8.

— machine instructions

operation)

stop
load
add

Operand



Breakpoints

- To show the current breakpoints we have, we may use the "info breakpoint" command as:
- (gdb) info breakpoint ✓

Num	Type	Disp	Enb	Address	What
1	breakpoint	yes	y	0x00001f7b	in main at first.c:8

- To **disable** a breakpoint: "disable breakpoint_number".
- To **re-enable** disabled breakpoint: "enable breakpoint_number".
- To **remove** a breakpoint: "delete breakpoint_number" or replace the "break" with "clear" command as we create these breakpoints.
  - (gdb) clear first.c:8
  - Deleted breakpoint 1

Executing GDB

- GDB allows
  - to execute and stop your program at specified points (break points)
  - examine what has happened and inspect your program after breakpoint
  - make changes to variables and run

Tasks	**GDB Commands**
Restart execution ✓	continue
See where the program stopped	list
Step through code line-by-line	Next step
Examine variables	print ✓

GDB stack-breakpoints-watchpoints

Tasks	**GDB Commands**
Get a backtrace	backtrace
Change stack frame	frame
Set breakpoint on line/function	Break line_number/function_name ✓
Set watchpoint on variable	Watch variable
Get list of breakpoints/watchpoints	Info breakpoints ✓
Disable breakpoints	Disable/clear breakpoints ✓

(Some) Unix Commands You Should Know

- pwd ✓
- ls ✓
- ls -l
- ls -la
- touch ✓ create file
- cat, more, less ✓
- mkdir, rmdir
- rm
- rm -rf
- cd
- wc
- top ✓
- make

Editors

- emacs
- Vim
- VScode

Check [Some Useful commands_Linux.pdf](#) on canvas

Summary

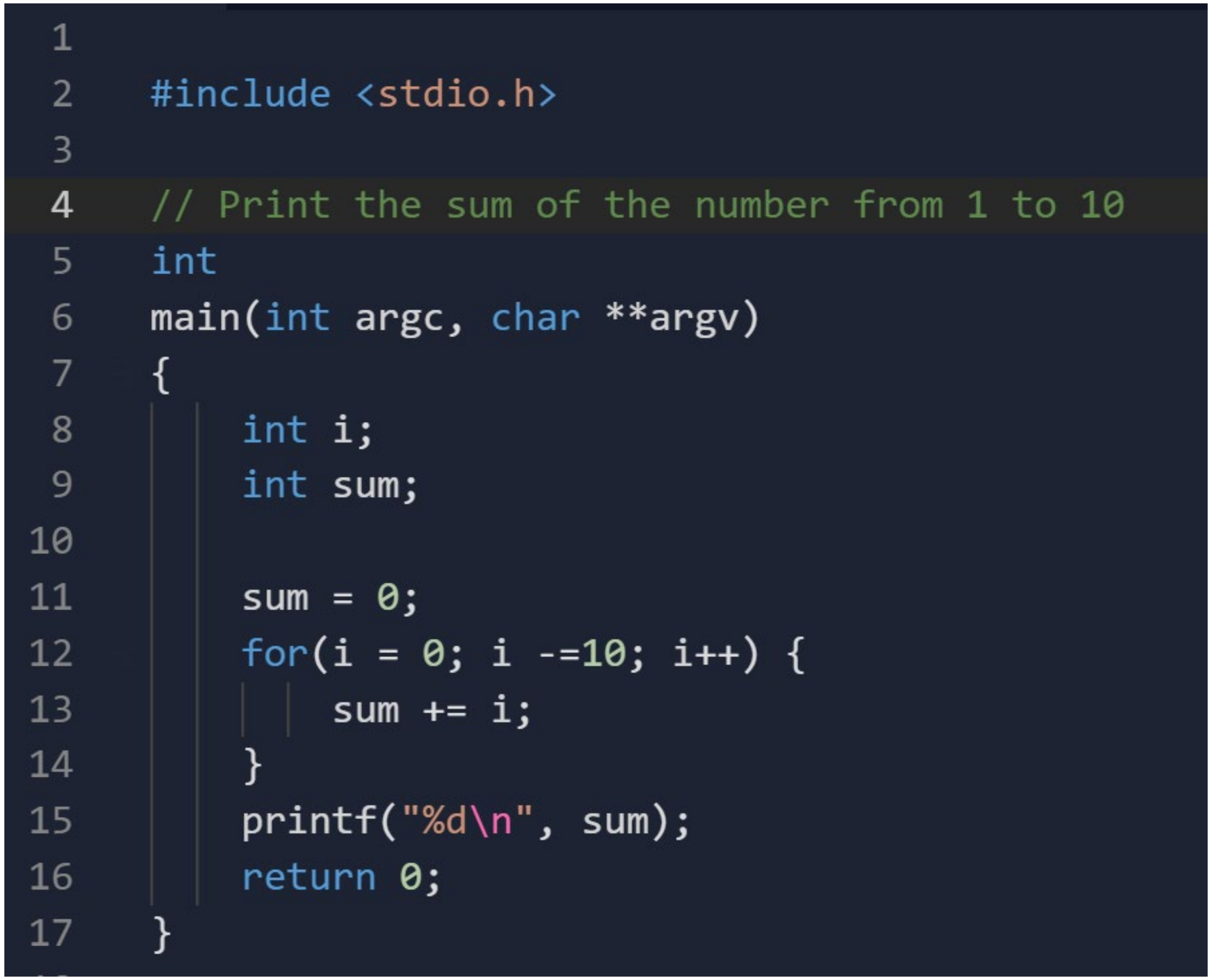
- OS Structures and their pros and cons
  - Monolithic – no structure
  - Layered
  - Microkernel
  - Modular
- Modern OSes are often hybrids ✓
  - Exhibit properties from multiple structures
- OS bootstrapping
- OS needs to be debugged and tuned – optimization
  - Various tools – perf, core dumps, top, tcpdump…

To do

1. Type the code and save it as sumdemo.c
2. Compile program using gcc
   gcc [flags] <source files> -o <output file> *search for –Wall and Werror*
3. Add –g option to enable built in support
   gcc *[flags] –g <source files> -o <output file*
4. start gdb
5. Run sumdemo (gdb) run
6. Use breakpoints (gdb) break sumdemo.c :12
7. Print sum (gdb) sum
8. Delete breakpoints
9. Fix the code
10. Run it again

Try breakpoint commands from **Some Useful commands_Linux.pdf**

To do



```

#include <stdio.h>

// Print the sum of the number from 1 to 10
int
main(int argc, char **argv)
{
    int i;
    int sum;

    sum = 0;
    for(i = 0; i -=10; i++) {
        sum += i;
    }
    printf("%d\n", sum);
    return 0;
}
```

Next

Process – Chapter 3