CS307 – System Practicum

Feb-June 2021

Assignment 2

Course Instructor: *Dr. Aditya Nigam*

Date: 09th March 2021

Instructions

- Plagiarism is strictly prohibited. In case of violation, a zero will be awarded for this assignment as a warning and a quick F grade if repeated later.
- In Moodle, students need to submit below given files in a *zip* archive named as *{Group_id}*_assignment_1.zip:
  - A **readme.md** file which provides the instructions for running your codes in detail including the versions of programming language and all the modules that have been used.
  - **Makefile** for compiling and executing the given code, make sure to *use gcc <= 9.3.*
  - The **code** and **report** for the assignment.
  - Give appropriate names to all the submitted files.
- Submit a single report for the assignment. The report should be concise.
- Students using Windows or any other OS are requested to make sure that their code runs perfectly on Linux as mentioned in each problem. Your evaluation will be done on computers having Ubuntu 20.04.
- Students must write their codes in C/C++ with gcc <= 9.3 programming language.
- The deadline for submission is **22nd March 2021**. No late submissions will be entertained.
- Contact **Dheeraj** [*b17041@students.iitmandi.ac.in*] , **Daksh** [*dak.thapar@gmail.com*] , **Uttkarsh** [*b17029@students.iitmandi.ac.in*] , or Ashima [*b17031@students.iitmandi.ac.in*] for any queries.

Problem 1 – Compiling the Linux kernel from source and building a loadable module
(15 marks)

In this problem, you will compile a custom Linux kernel from its source. Then you will build a simple loadable module in the new kernel and try out some ways of crashing the system. Some benefits of a custom Linux kernel are:

1. Support a wide range of hardware including the latest hardware.
2. Remove unwanted drivers from the kernel.
3. Faster boot time due to small kernel size.
4. Increased security due to additional or removed modules/drivers/features.
5. You will learn about the kernel and advanced usage.
6. Always run the cutting edge latest kernel.
7. Lower memory usage.

Note: Using a virtual machine is not necessary to compile/install a custom Linux kernel. Doing it directly on your work system can be dangerous, especially when you are writing kernel modules to deliberately crash the system.

Custom Linux Kernel

1. First, decide which Linux kernel you want to install. For this assignment, you should install 5.9 or 5.10.
2. Setup a virtual machine, preferably using VirtualBox. You can use Lubuntu, Ubuntu or some other Linux distribution. Kernel compilation takes too much space, so try to allocate at least 40-50GBs to the virtual machine.
3. Make sure your VirtualBox supports the kernel you want to build. Check the release notes here: https://www.virtualbox.org/wiki/Changelog. If not, upgrade your VirtualBox installation.
4. Download the source code of the specific kernel version you want to build.
5. To build and install a custom kernel, you can check these resources:
    a. https://help.ubuntu.com/community/Kernel/Compile
    b. https://www.linode.com/docs/guides/custom-compiled-kernel-debian-ubuntu/
    c. DuckDuckGo/Google/Bing
6. Try to reduce the size of the custom kernel and compare the size of your custom kernel with the default kernel. Report your findings. Hint: Do you need all the modules? What about debug symbols?

Building a simple loadable module

1. After the installation is complete, you will build a simple loadable module in Linux. You can follow the steps from:

a. http://www.iitk.ac.in/LDP/HOWTO/pdf/Module-HOWTO.pdf
b. https://scottc130.medium.com/writing-your-first-kernel-module-98ae68edf0e
c. http://www.hitchhikersguidetolearning.com/2018/08/19/writing-a-simple-linux-kernel-module/
d. DuckDuckGo/Google/Bing

2. Introduce some errors in the loadable module e.g. division by 0, returning a value other than zero from init mod, etc. See if you are able to crash the system. Try atleast 3-4 ways and return values and infer the results.
3. Store the bad modules for demonstration.

Problem 2 – Round-Robin Scheduling

(15 marks)

Input

1. N = Number of processes
2. Arrival times of each process. If all processes arrive at each time, this can be set to 0 for all
3. Burst time for each process
4. Time quantum

Output

1. Start Time, Completion Time, Turnaround Time, Waiting Time and Response Time for each process
2. Average Turnround Time (ATT), Average Waiting Time (AWT) and Average Response Time (ART)
3. Throughput and CPU utilization

Experiment with different values of N and time quantum. Compare the ATT, AWT and ART for different time quantum for each N.

What do you observe for very small or very large values of time quantum? Write your results and inferences in your report.

Problem 3 – Merge Sort using threads

(15 marks)

You need to implement Merge Sort **using threads**. The sorting should be **in place.** You are free to allocate some additional memory to keep temporary data if needed.

Rules:

1. The number of threads (first argument) and size of input array (second argument) will be passed as command line arguments. **Please do not hard code the number of threads or the size of the array.** Rounding the number of threads down to the nearest power of 2 is acceptable.
2. Using the size of the array, create a randomly generated integer-type array.

3. Output the initial unsorted array and the final sorted array along with the total execution time.

Input/Output:

```
>> ./myprogram <number_of_threads> <array_size>

 Initial array: [some randomly generated array of size <array_size>]

Final array: [sorted initial array]
 Total time: [total execution time]
```

[Ungraded] Explore OpenMP and see how it handles threading. If possible, try running your merge sort code using OpenMP. You will be amazed to realise how OpenMP makes work sharing among multiple threads so easy that it almost feels like cheating.

The End