CS 472 Operating Systems Design

Project 6: Parallel Zip

Part 1: Due on November 18, 2020, 11:59pm

Part 2: Due on December 2, 2020, 11:59pm

Part 3: Due on December 9, 2020, 11:59pm

This project is to be done in groups of size one or two (not three or more or zero).

The goal of this project assignment is to let you familiarize yourself with the concurrent programming application using locks and conditional variables for thread control. This is a very **challenging** project, and please start early! Please refer to the following link at GitHub:

https://github.com/remzi-arpacidusseau/ostep-projects/tree/master/concurrency-pzip

to implement a parallel zip application. However, please work on this project by following the instruction in **this** document (*i.e.*, project6.pdf).

It is strongly recommend to watch the following video before working on this project:

https://www.youtube.com/watch?v=WVHRaqom0yo&feature=youtu.be

by Dr. Remzi Arpaci-Dusseau.

To better approach this project, the tasks are divided into three parts, as well as three **deadlines**.

Part 1: Single thread using mmap() and write().

Change the code wzip.c from Project 1 to use mmap() and write(). That is, use mmap() to map a file or files into memory and store the results in the heap memory. Then, use write() to print the results in the heap memory to the standard output. Please see an example in the man page of function “mmap”. Please name your code “zip-mmap.c”.

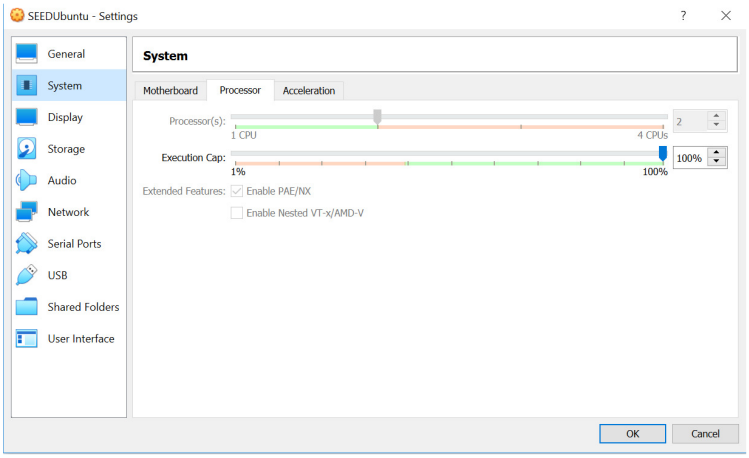
To test the correctness of zip-mmap.c, please copy the “tests” directory from the Project 1 “initial-utilities/wzip/tests”. Change the “tests/3.out” to “**pzip**: file1 [file2 ...]”. A testing script “test-mmap.sh” with other supporting testing files (i.e., run-tests.sh and test-time.sh) can be downloaded from Brightspace. Note that in test-time.sh, “**time** ./run-tests.sh” is used so that the time to run all testing cases is shown.

Part 2: Multiple threads using the static method.

Please name your code “pzip-static.c”, which is enhanced from “zip-mmap.c”.

A file is (roughly) evenly divided into *n* parts, where *n* is the number of processors in SEEDUbuntu VM. Each part of the file is zipped by a thread. Consider to use a proper **data structure** for thread job description and for storing zipped results. Moreover, please refer to Chapter 29 (Lock-based Concurrent Data Structure) for how to apply a data structure for a **queue**.

To change the number of CPUs in VM, please use “Settings” in VirtualBox for the VM, and follow the following path: System -> Processor -> Number of Processors, *i.e.*,



Please make sure to test three cases: number of CPU(s) = 1, 2, or 4.

Moreover, please avoid memory leakage in the heap memory by running the “valgrind” tool.

The testing script “test-static.sh” and other supporting testing files (i.e., run-tests.sh and test-time.sh) can be downloaded from Brightspace.

Part 3: Multiple threads using producer/consumer, but with unbounded buffer.

Please name your code “pzip-dynamic.c”, which is enhanced from “pzip-static.c”.

Instead of being statically assigned with (roughly) equal size from a file, each thread can run a dynamic number of multiple blocks. A big file is divided into blocks with the size up to 1MB or other proper size. The main thread is a producer and produces job into a buffer, whereas threads consumes jobs until all jobs are done. Please refer to Chapter 30 (Condition Variables) for the producer/consumer problem. But here the buffer can be unbounded, so the problem is easier than the example in Chapter 30.

Please make sure to run the final code many times to detect and fix the possible deadlock issue, especially using 4 CPUs for testing.

The testing script “test-dynamic.sh” and other supporting testing files (i.e., run-tests.sh and test-time.sh) can be downloaded from Brightspace.

For each part, please write a README file, which include the following information:

- How the code is implemented? What are main function calls?
- How did you test the code to make sure the correctness of the code?
- If you work with a partner, who is your group member? How much code was contributed by you, and how much was contributed by your group member? Which functions were implemented you or your group member?

Please upload your implement (zip-mmap.c and README-mmap, pzip-static.c and README-static, or pzip-dynamic.c and README-dynamic) to your GitHub project repository under “project-6” directory before each deadline.

Grading rubric:

- Part 1: zip-mmap.c – 50pts
    - Correctness – 30pts
        - Each test 5 pts.
    - Memory leakage – 10pts
        - Each test 5 pts
    - README-mmap file – 10pts

- Part 2: pzip-static – 200pts
    - Correctness with thread creation and completion – 120pts
        - Each test 20 pts
    - Memory leakage – 60pts
        - Each test 30 pts
    - README-static file – 20pts

- Part 3: pzip-dynamic – 150pts
    - Correctness with producer and consumer (condition variables) – 60pts
        - Each test 10 pts
    - Correctness without deadlock – 40pts
    - Memory leakage – 30pts
        - Each test 15pts
    - README-dynamic file – 20pts

Total: 400 pts