CS347M Operating Systems
Spring 2020-21
Lab3

This lab has the following aspects,

- You can choose any one of the following statements for developing an implementation based solution.
- Each statement is a high level description of a potential problem. Part of the lab's requirement is to define a workable and interesting problem scope/statement.
- The design and implementation should be tailored to this scope.
- A report explaining the exact problem statement and scope, highlights of design and implementation is required.
- You are required to demonstrate correctness and features of your solution and will be considered an important part of the evaluation.
- Evaluation of the lab will be conducted via an online viva.
- The lab is to be attempted as an individual assignment.

- **Submission deadline: 20th April 2021**

<u>List of problem statements</u>

1. Implement two different locking mechanisms in xv6---blocking and non-blocking and compare them to report performance with different workloads/usages. The locks are to be used by user processes via system calls, and implemented in the operating system.

2. Implement two-different scheduling algorithms to schedule processes in xv6, other than the default xv6 scheduler. Compare and report performance with different workloads/usages.

3. Write a user-space file system that uses a file as a secondary storage device to store files and directories. Provide a simple command line interface for file operations---create, read, write, delete, redirect etc. All meta-data and data of files manipulated via the interface are stored is a single large file.

4. Implement a lazy memory allocation (and mapping) optimization in xv6. Demonstrate correctness in terms of allocations and reclamations. Swapping need not be implemented. Lazy allocation --- physical memory is allocated to process, only when accessed and not on request for allocation.

5. Dynamic memory management of virtual memory
   Implement the ***salloc*** and ***sfree*** functions to emulate management of the virtual address space of a process.
   You can start by allocating some fixed large size (say 512MB) and then manage this memory using **salloc** and **sfree**. Implement two heuristics (best-fit, first-fit, buddy allocation etc.) to manage allocation and freeing of the address space and compare their performance.
   You also need to implement a command to display the status of the address space (such as the amount of free memory, amount of utilised memory, allocated regions etc.)

6. Using the pthread library implement a set of primitives for managing threads,
   a. Thread **ordering**
      A set of threads (each with an ID) execute in a specified order.
   b. **Barrier** synchronization
      A set of threads wait at a ***barrier***, a point in execution till where all threads need to reach before they can execute further.
   c. **Priority** synchronization
      Some threads have higher priority than other threads during synchronization and get access to lock before other threads with lower priority.
   d. your own new synchronization primitive

7. Add signal handling capabilities in xv6. A process can make a system call to send a signal to another process. The signaled process on returning to the user process first executes the signal handler and then returns to its original return address via the operating systems. (Needs understanding of process execution stack, the trapframe, of xv6).

8. Implement a shared memory service between processes in xv6. The memory region (to be shared) needs procedures to be allocated, shared and possibly an index to identify/reference the region.

9. Your own cool idea.