CSCI 347 Computer Systems II
Summer 2018
Assignment 3

Submitting Your Work

This assignment is worth 15% of the grade for the course. Save your program files (including header files and makefile) in a zipped tar file and submit the file via the **Assignment 3 Submission** item on the course web site. You must submit your assignment by 2:00 pm on Monday, August 20.

Your assignments will be evaluated on correct functionality and conformance to the coding standards described at the end of this assignment specification.

Introduction

You are to write one or more multi-threaded C programs, using the pthreads library to simulate a disk scheduler and use those programs to compare the performance of disk operations under three different scheduling strategies.

Disk Scheduling

A computer's operating system receives requests for disk I/O operations from the processes running on the computer. It will use a disk scheduling strategy to determine which request to service next, with the aim of optimizing that aspect of the computer's operations. There are two performance measures that matter:

- **Throughput**: the number of disk I/O requests completed per minute (or some other unit of time). For good performance, we want to maximize throughput.
- **Fairness**: ensure that there is a low variance in the time taken to complete individual requests. For fairness, we want to minimize the variance of service times.

Disk I/O Performance

Each request for a disk I/O operation specifies the cylinder, head and sector to be written or read. The read/write head must first move to the required cylinder, the required read/write head is activated and the operation is performed when the required sector passes under the read/write head.

The time taken for any I/O operation on a (spinning) disk drive is the sum of several parts:

- **Seek time**: the time for the read/write heads to move to the required cylinder. This varies with the quality of disk drive: low-end drives may have a seek time of 20ms, high-end enterprise drives may be around 5ms.

- **Head switch time**: typically 1 or 2ms.

- **Rotational latency**: the time for the disk to rotate so that the required sector passes under the read/write head. This depends on the disk drives spindle speed. For a 7,200 rpm disk, rotational latency is around 4ms.

- **Data transmission**: the time taken to move the data from the disk drive to the computer's memory. This is less than 1ms for a 1Kbyte data transfer.

All of these amounts are constant among all I/O requests, except for seek time, which will vary according to how far the disk heads have to travel to get from their current cylinder to the required cylinder.

Your Task

You are to implement a model of the disk scheduling system to study the performance of three scheduling strategies:

1. **First Come First Served (FCFS)**: requests are serviced strictly in the order of receipt by the scheduler.

2. **Shortest Seek Time First (SSTF)**: the next request to be serviced is the one whose cylinder is closest to the cylinder of the request most recently serviced.

3. **Circular Scan (CSCAN)**: this is the (one-way) elevator algorithm, in which requests are only serviced as the read/write head travels in one direction, servicing the requests in the order of increasing cylinder number. Any new requests for a cylinder number greater than or equal to the read/write head's current position are serviced in the current sweep from low-number cylinders to high-number cylinders. Any new requests for a cylinder number less than the read/write head's current position are queued for the next sweep. No requests are serviced between sweeps, as the read/write heads move back to cylinder 0.

Suggested Model

The following model is suggested for this simulation.

1. There are multiple clients of the disk scheduler. These clients are the processes running on the computer. Each process will repeatedly sleep for some random time, make an I/O request for some random cylinder and wait for that request to be completed.

2. There is one disk scheduler which receives and manages requests from the clients.

3. There is one device driver which performs one disk I/O operation at a time. The device driver repeatedly asks the disk scheduler for the next request, services that request and then informs the disk scheduler of completion of the request. If there are no requests pending, the disk drive must wait until the disk scheduler receives a request.

Assumptions and Requirements

The following assumptions and requirements apply:

1. Each client process sleeps for a random period, uniformly distributed between 5 and 15 milliseconds, between making I/O requests.

2. Each I/O request will be for a cylinder chosen at random from a uniform distribution of the 16,000 cylinders in the disk drive. The head and sector are not specified as these do not provide a variable component in the time taken for a disk I/O operation.

3. The time taken by the device driver to service a request is equal to 10ms plus 0.1ms per cylinder that the heads have to move from their current position to the cylinder of the request.

4. The disk scheduler's data structures, used for management of I/O requests, are shared resources. They must only be changed by atomic operations, ensured by the use of a mutex.

5. The number of client processes and the number of I/O operations to be performed by each client are to be specified as command-line arguments for the main() method.

6. The clients, disk scheduler and device driver must be implemented as threads.

7. The disk scheduler and device driver must wait for all the clients to complete their iterations and then terminate.

8. The output from the simulation must include a measure of throughput of I/O operations as well as the mean and variance of individual request service times.

Saving your files in a zipped tar file

You need to submit all your C and header files, as well as your makefile. First you need to bundle them up into a single *tar* file. The term "tar" is an abbreviation of "tape archive" and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

1. Use the command:

   ```
   tar -cf myfiles.tar *.c *.h Makefile
   ```

   The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

   Following the name of the tar file, we list the files to be included in the tar file. In this case, we want all the files in your current directory whose names end with `".c" or ".h",` as well as `Makefile`.

2. If you now use the command `ls` you should now see the file myfiles.tar in your directory.

3. If you use the command

   ```
   tar -tf myfiles.tar
   ```

   it will list the files within the tar file.

4. Now compress the tar file using the gzip program:

   ```
   gzip myfiles.tar
   ```

5. By using the `ls` command again, you should see the file myfiles.tar.gz in your directory. This is the file that you need to submit through the **Assignment 3 Submission** link in the course web site.

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.
2. Avoid the repeated use of numeric constants. For any numeric constants used in your program, use #define preprocessor directives to define a macro name and then use that name wherever the value is needed.
3. Use comments at the start of the program to identify the purpose of the program, the author and the date written.
4. Use comments at the start of each function to describe the purpose of the function, the purpose of each parameter to the function, and the return value from the function.
5. Use comments at the start of each section of the program to explain what that part of the program does.
6. Use consistent indentation.
7. Do not let functions get large. Any more than 20 lines of code in a function probably means that the function is too complex and you need to break its task into subtasks, each handled by a separate function.