Assignment-2: Student management system (IPC)

Operating System

August 11, 2023

1 Context

This assignment requires you to reuse various components from previous version. The main context remains same as before. However, the application structure will be different. New application will implement interprocess communication using pipes for interaction among various processes.

2 Problem overview

You are required to develop an application for the same club as before. So there is no need to change the code for database. Operations provided will remain same, and so will their operation id. There will be three kinds of role, administrator, manager, and members as before. However, now these roles will be generating operation requests and the requests generated by them will have different priority.

This application will have following components:

1. **Kernel** The whole application will start with this compoenent (or say process). This process will be the first one to start and it will be the last one to terminate. All remaining components will be forked from this process and will be loaded a new program in their address space to perform corresponding task.

   The Kernel will also manage disk I/O upon request in kernel mode. For this it will load a program to do disk I/O, once loaded the program will keep running until the server is terminated.

2. **Record_generator** This process will generate 100 records as specified in last version.

3. **Request_generator_xyz** After data generation, three req_generator processes will start namely request_generator_admin, req_generator_manager, req_generator_member. Requests generated by these processes will have different priorities. More on these generators are as follows:

(a) **Request_generator_admin** This generator after starting waits for random duration of 2 to 5 seconds and then starts generating requests. It generates a total of 12 requests with random pause in between. After 12 generations it terminates.

(b) **Request_generator_manager** This generator waits for a random duration of 1 to 4 seconds and then starts generating requests. It generates a total of 15 requests with random pauses in between and then terminates.

(c) **Request_generator_member** This generator generates requests at a regular interval of one second. After generating a total of 35 requests it terminates.

Each request_generator will request kernel to establish a connection with 'Server' before it starts request generation process. In reply kernel will send a pair of numbers. Zero reflects that connection can not be established. Non-zero positive numbers will be pipe descriptors (read, write) indicating the established connection with the server.

4. **Server** Once a *generator* requests *kernel* to establish the connection with *Server*, then *kernel* loads server program (if not already running). Followed by which, connection between the *server* and the corresponding *request_generator* is established and requests for an operation along with student id are sent to server. Once a requested operation is completed, server returns a four tuple (request_sequence_no, time_of_submission, time_of completion, total_waiting_time). Requested operation will follow following procedure

   (a) **Service state** When the server is processing a request, we call it *service*. A service can be in one of three states namely *ready*, *running*, or *blocked*. The description of these states are as follows:

      i. *Ready* When a request is waiting to be processed by Server, it is in *ready* state.

      ii. *Running* When a request is being processed by server then it is in *running* state.

      iii. *Blocked* When a process does not find the data in main memory, then it is put in *blocked* state and server raises a disk I/O request with *Kernel*. Once the *kernel* returns with data from disk, the *server* places the *service* back into *ready* state.

   (b) **Priority**

      i. A service associated with admin will be given highest priority. So, such a *service* will be executed after at most one service already present in the *ready* state.

      ii. A service associated with manager will be given higher than *member* but lower than *admin* services. It will wait at most three *services* associated with a member in *ready* state before being served.

5. **Memory** The details of this remain same as specified before. The main-memory will consist of 5 linked-list nodes.

3 Performance Observation

1. **Scenario1** Measure the throughput, mean response time (time taken for a service to reach in *running state* first time), and mean turn-around time (time duration from request generation to service completion) for all request generators.
2. **Scenario2** Measure the impact of increasing the main-memory size by 20% and then 50%
3. **Scenario3** Measure the impact of faster memory technology, if both the memories perform twice as fast. Measure this performance for original memory size and increased memory size.

4 Code Execution/submission

Those members who did not implemented linked list in previous submission, will implement at least one queue using linked list in this submission. The code should be executable with one single command 'sh run.sh'; all the remaining commands such as make or other commands should be mentioned in 'run.sh' script. Clearly, identify different componenets, and divide among yourselves. Aim to develop a modular code. Each file/module must have on top a description mentioning which method is implemented by whom. Keep the variable names readable, and same with method names.

5 Group Details

This is a group assignment. A group of three or four is permitted. The skills required are Linked list and File Handling, makefile, and shell scripting. Each member will implement and maintain a piece of code (assessment will be on your code; not planning or documentation). While implementing the project, keep thinking how the process can be improved, if required in future.