Trung Viet Nguyen

Lab 1

CIS 3207

Discrete Event Simulator – ReadMe File

Abstraction:

The goal of the program is to simulate a simple computing system with 1 CPU and 2 Disks. The diagram for the program is shown below. The program will first read from a configuration file (config.txt) containing a seed for the random function and the configurations, then start generating and handling events using the appropriate data structures for each component (priority queue for the event input and 3 FIFO queues for the CPU and Disks). The configurations of the program and the event queue log will be written to a log.txt file, and the statistics of the components (average and the maximum size of each queue, utilization of each server…) will be displayed to the screen.

Files included in the program:

queue.h: The header file for the queue structure/methods.

queue.c: Accompanied with queue.h file to be used by the main.c file.

main.c: Contains the main function and the simulation functions.

config.txt: containing the configurations for the program (SEED, INIT_TIME, FIN_TIME…).

log.txt: The result file created by the program.

To compile the file, put all files in a folder and use this command:

(In UNIX Terminal) -gcc queue.c main.c -o main

Some alterations:

Instead of building a priority queue structure for the event queue, I let it use the same queue structure as the CPU and Disks but implement a priority enqueue function for it.

The PROB_QUIT configuration in my program is in percentage (so 0.2 would be 20 (%)).

Program Explanation:

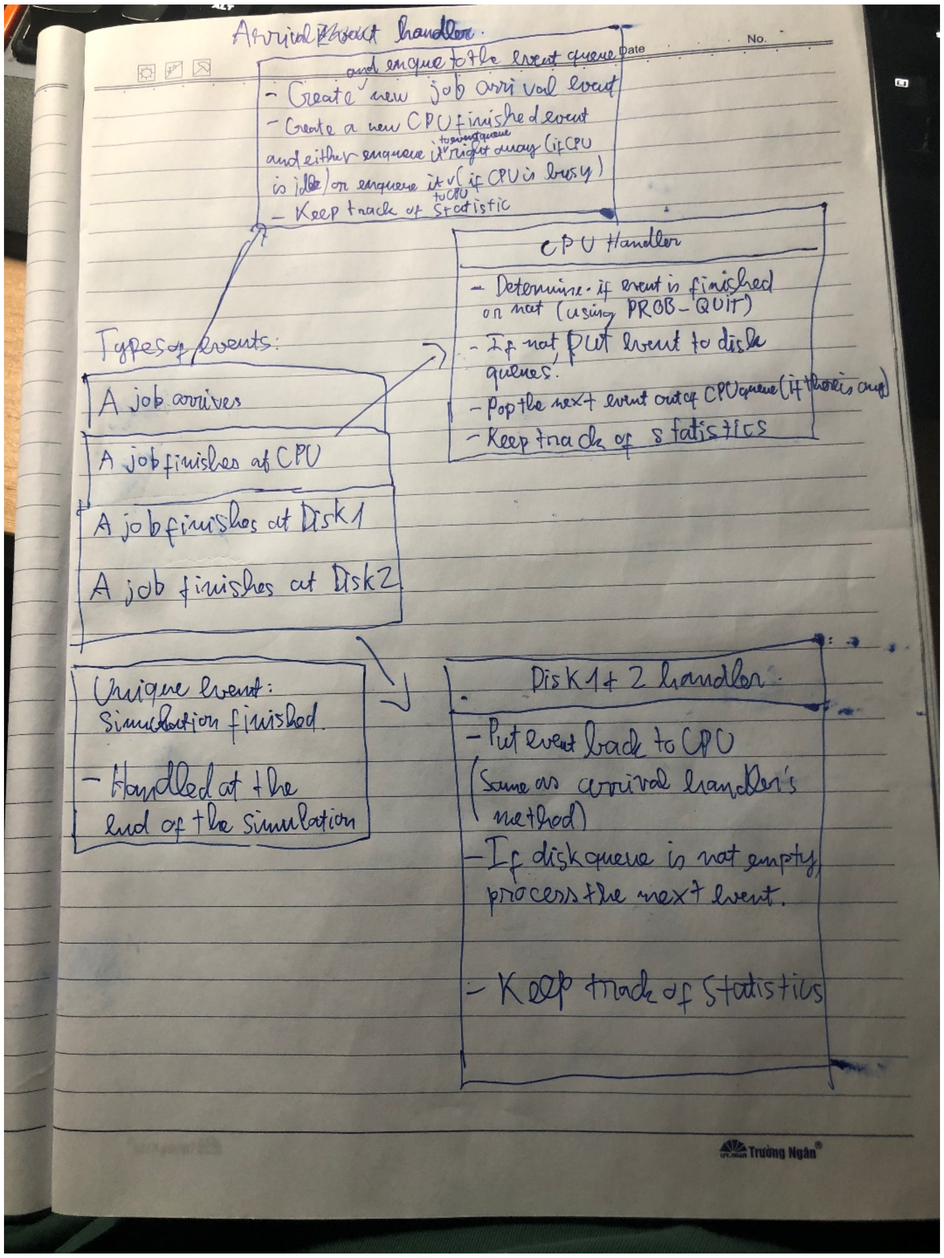
1. **The queue**

The file queue.c and queue.h contains the queue structures and functions that will be used by the main.c file. There are 3 structures inside the queue.h file: queue, node, and event. An event is a structure containing an integer for the job number, an array of character for the event description (E.g.: “arrives”, “finishes at CPU” …), and an integer for the time that the event occurs. A node is defined to contain an event and a pointer that links to another node. The implemented queue contains 2 nodes, head and tail, along with an integer variable to get the queue’s size. The files also contain the standard functions for the queue data structure (enqueue, dequeue, get size), the constructor functions (newEvent, newNode…), and a function to print out the events. To simplify the work, the event queue and component queue share the same data structure. Instead, there are two types of enqueue functions: FIFO enqueue and Priority enqueue. The normal FIFO enqueue is used for the CPU and Disks, while the Priority enqueue is used for the event queue. The priority enqueue will always place the events with smaller time variable above the ones with bigger time variable.

2. **The main file and the actual simulation**

The main.c file contains the simulation and handler functions, along with the file I/O inside the main function. It firsts reads the configurations from config.txt into an array of integer (that will be used by the functions later on), then write the configurations into the log.txt file. The simulation function then creates the 4 queues (event, CPU, disk 1, disk 2), and load the first event into the event queue. It then begins to dequeue the event queue until either the queue is empty (should not happen before reaching finish time) or the finish time FIN_TIME has been reached. For each type of event, a handler function is used. The log for the event generating and handling process is written into the log.txt file. During the process, the handler functions also keep track of the components statistics by modifying the statistic pointers each time it is called. After the final event “Simulation finishes” has been reached, it will print out the statistics to the screen.

A diagram is attached below to clarify how the functions work, and further comments are provided inside the .c files.


Arrival ~~[illegible]~~ handler.
- Create new job arrival event and enque to the event queue
- Create a new CPU finished event and either enqueue it to event queue right away (if CPU is idle) or enqueue it to CPU (if CPU is busy)
- Keep track of statistic
CPU Handler
- Determine if event is finished or not (using PROB-QUIT)
- If not, put event to disk queues.
- Pop the next event out of CPU queue (if there is any)
- Keep track of statistics
Types of events:
A job arrives
A job finishes at CPU
A job finishes at Disk1
A job finishes at Disk2
Unique event: Simulation finished
- Handled at the end of the simulation
Disk 1+2 handler
- Put event back to CPU (Same as arrival handler's method)
- If disk queue is not empty process the next event.
- Keep track of statistics