Tuan Nguyen

**README**

In the project, we implemented a system which includes:

1) priority queue to represent the Job Scheduler

2) FIFO Queue to represent the CPU

3) FIFO Queue to represent two Disks: Disk1 , Disk2

-When it is running, the system is responsible for scheduling jobs in different components based on their event types and write all the events into a log file which also includes jobID and time of each events, we have 6 different types of events:

\*Arrive at CPU

\*Finish at CPU

\*Arrive at Disk

\*Finish at Disk 1/2

\*Exits the system

\*Simulation Finished

-At each iteration, a Job will be popped from the Scheduler. If event type is Arrive at CPU. The CPU also will create a new Job with event type Arrive and send it back to the Scheduler. The arrival time will be decided by generate a random number from a specified range and add it to the currentTime variable (We need to read the parameters that specify the range from a (file.txt): config file. The Job will go to the CPU to wait for processing. There are 2 cases:

    -If the CPU is busying processing other jobs:

we'll do nothing

    -Else we will pop the task at the top of the CPU queue. Create a new event with type equals Finish at CPU and determine their finished time the same way we did previously. We switch the CPU to busy state

\*If event type is Finish at CPU, we will do the following:

    -Set CPU state to not busy

    -with some probability p, we will determine whether a job should leave the system or will be sent to disk for further I/O processing

\*If event type is Arrive at Disk, we will select the Disk with less capacity and push the new job that was created previously to that disk. If the disk is of equal capacity, we select randomly. Also, we need to generate a Disk finished event from the Disk and push it to the Scheduler

\*If event type is Finish at Disk, we determine which disk was responsible for processing the job. Now the job is ready to go back to CPU for further processing (switch from blocked state to ready), we create a new event with type CPU Arrival and send it to the Scheduler, the time of that job was the current time.

**File**

**main.c**

-Initialized all event and priority queue to constant

-Create 3 priority queues for CPU, disk1, and disk 2

-A while loop to process simulation,

\*Pop job from priority queue

\*See event is job arrive at a CPU

\*Find the time job enter the queue to the time job leave the CPU

\*Compare CPU max with current CPU size

\* Find average size

\* Event is job finish at CPU

\* Event is job arrive at Disk 1 & 2

\*Event is job finish at disk

\* Event is simulation end

After all process finished, print the time job arrival at CPU, disk 1, and disk 2. Also, print the average size of disk, statistic max size.

**fifoqueue.c**

-create struct for node include: int time; int eventID; int jobID, ffnode \*next;

- create another struct for queue include: int size; ffNode \*head; ffNode \*tail; int state;

- make new node and set time, eventID, jobID

-initialize jobQueue

-void Enqueue to enqueue

-Dequeue method to dequeue

-print the queue include jobID, eventID, and time.

-ffNode\* top: a function to get the top element of queue

**fifoqueue.h**

-two struct for node and queue

-include all function header that use in fifoqueue.c

ffNode\* createffNode (int time, int eventID , int jobID);

Queue\* initializeJobQueue();

void Enqueue(Queue \*q , int time , int eventID , int jobID);

ffNode\* Deque(Queue \*q);

void print\_ff( Queue \*q);

ffNode\* top(Queue \*q);

**priorityqueue.c**

-struct Node : int time; int eventID; int jobID;

-struct node \* next

-set new node to its time, eventID, and jobID

Method:

-Node\* peek(Node\*\* head): return the value at head

-Node\* pop(Node\*\* head): Removes the element with the highest time form the list

-void push(Node\*\* head, int time, int eventID , int jobID): to enqueue

-int isEmpty(Node\* head): check whether list is empty

-void print\_pq( Node\* head ): print priority\_queue

**priorityqueue.h**

**-** struct Node : int time; int eventID; int jobID;

- include all function header that use in priorityqueue.c

Node\* newNode(int time,int eventID, int jobID);

Node peek(Node\*\* head);

Node\* pop(Node\*\* head);

void push(Node\*\* head, int time, int eventID, int jobID);

int isEmpty(Node\* head);

void print\_pq( Node\* head);

**otherFuction.c**

-method:

int searchValue(char \*fname, char \*str): read parameter value from log file

int randomInt(int a, int b): generate random number in a range

int quit(int QUIT\_PROB): determine whether job will quit system or arrive a disk

int create\_job(): create new job and return it in CPU, disk 1, disk 2

void appendToFile(int time, int eventID, int jobID, char \*filename): write events of arrival time, finish time, simulate to log file

**otherFuction.h**

**-** include all function header that use in otherFuction.c

int searchValue(char \*fname, char \*str);

int randomInt(int a, int b);

int quit(int QUIT\_PROB);

int create\_job();

void appendToFile(int time, int eventID, int jobID, char \*filename);

**file.txt**

-test program with input include

SEED 8

INIT\_TIME 0

FIN\_TIME 10000

ARRIVE\_MIN 1

ARRIVE\_MAX 100

QUIT\_PROB 2

CPU\_MIN 1

CPU\_MAX 50

DISK1\_MIN 1

DISK1\_MAX 50

DISK2\_MIN 1

DISK2\_MAX 50

**out.txt**

-result after input in file.txt is run

At time 0 job 0 arrives

At time 57 job 1 arrives

At time 103 job 0 finished at CPU

At time 103 job 0 arrived at disk

At time 106 job 2 arrives

At time 109 job 0 finished at disk 1

At time 109 job 0 arrives

At time 139 job 3 arrives

At time 142 job 1 finished at CPU

At time 142 job 1 arrived at disk

At time 232 job 4 arrives

At time 282 job 5 arrives

At time 312 job 1 finished at disk 2

At time 312 job 1 arrives

At time 345 job 6 arrives

At time 350 job 2 finished at CPU

At time 350 job 2 exits

At time 387 job 7 arrives

At time 484 job 8 arrives

At time 521 job 9 arrives

At time 554 job 10 arrives

At time 584 job 0 finished at CPU

At time 584 job 0 arrived at disk

At time 681 job 11 arrives

At time 765 job 12 arrives

At time 783 job 13 arrives

At time 811 job 0 finished at disk 1

At time 811 job 0 arrives

At time 910 job 14 arrives

At time 937 job 3 finished at CPU

At time 937 job 3 arrived at disk

At time 1030 job 15 arrives

At time 1120 job 16 arrives

At time 1208 job 17 arrives

At time 1254 job 18 arrives

At time 1281 job 1 finished at disk 2

At time 1281 job 1 arrives

…

**RUNS.txt**

-use different input to run

-choose seed value is 9, not much different result from seed value is 8

SEED 9

INIT\_TIME 0

FIN\_TIME 10000

ARRIVE\_MIN 1

ARRIVE\_MAX 100

QUIT\_PROB 2

CPU\_MIN 1

CPU\_MAX 50

DISK1\_MIN 1

DISK1\_MAX 50

DISK2\_MIN 1

DISK2\_MAX 50

-In Out.txt

At time 0 job 0 arrives

At time 64 job 1 arrives

At time 109 job 0 finished at CPU

At time 109 job 0 arrived at disk

At time 185 job 2 arrives

At time 218 job 0 finished at disk 1

At time 218 job 0 arrives

At time 220 job 3 arrives

At time 267 job 1 finished at CPU

At time 267 job 1 arrived at disk

At time 288 job 4 arrives

At time 314 job 5 arrives

At time 334 job 1 finished at disk 2

At time 334 job 1 arrives

At time 432 job 6 arrives

At time 482 job 2 finished at CPU

At time 482 job 2 exits

At time 561 job 7 arrives

At time 645 job 8 arrives

At time 685 job 9 arrives

At time 747 job 10 arrives

At time 767 job 0 finished at CPU

At time 767 job 0 arrived at disk

At time 776 job 11 arrives

At time 863 job 12 arrives

At time 903 job 13 arrives

At time 941 job 0 finished at disk 1

At time 941 job 0 arrives

At time 953 job 14 arrives

At time 963 job 3 finished at CPU

At time 963 job 3 arrived at disk

At time 1055 job 15 arrives

At time 1123 job 16 arrives

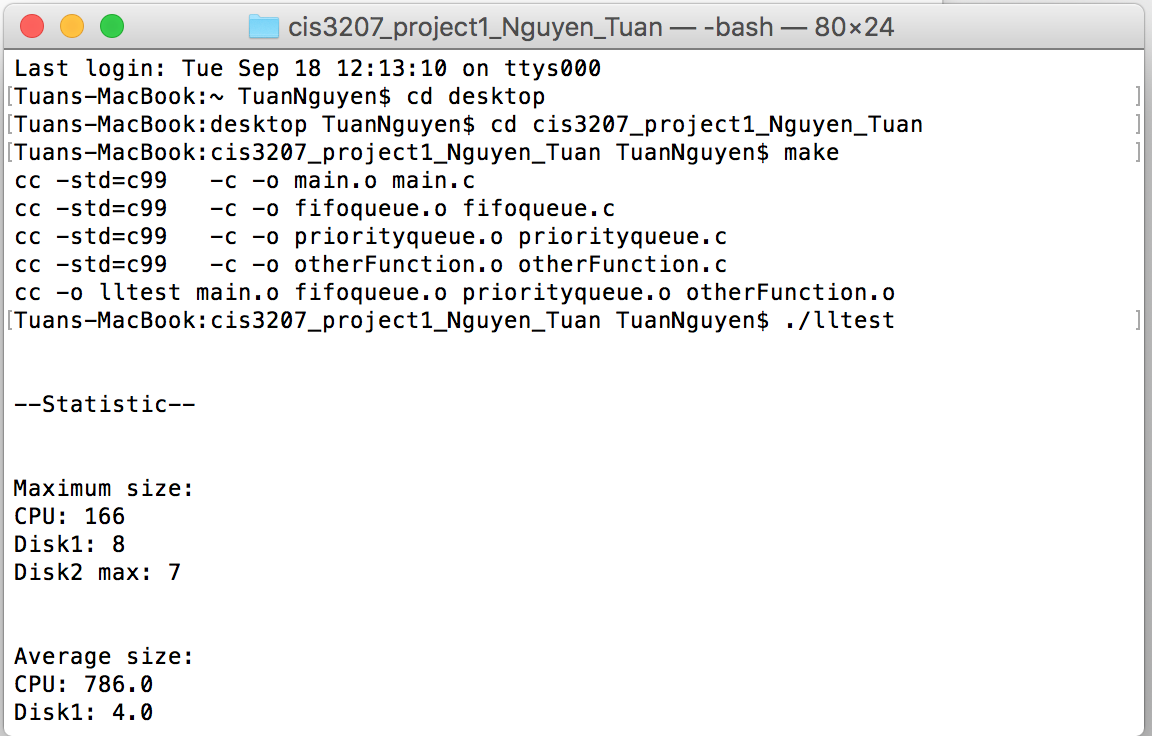
At time 1127 job 17 arrives

At time 1161 job 18 arrives

…

We still obtain over 1000 jobs with both input seed 8 and 9

To compile

-

-cd cis3207\_project1\_Nguyen\_Tuan

-make

-./lltest