ECE 223 Programming Assignment 3
Due Date in Canvas

Disk Simulation Program

You will create two new ADTs for this program: a FIFO queue ADT named disk_queue.c based on your existing linked list ADT (which must be brought up to spec if it is not already) and a priority queue ADT named event_queue.c. You will provide two different queues. The first is a priority queue. This will be built in a module called event_queue.c with the following interface:

```
typedef struct event_queue_s *event_queue_t;

event_queue_t event_queue_init(int size);
int event_queue_insert(event_queue_t eq, event_t ev);
event_t event_queue_remove(event_queue_t eq);
int event_queue_empty(event_queue_t eq);
int event_queue_full(event_queue_t eq);
void event_queue_finalize(event_queue_t eq);
```

You will implement this priority queue as a binary tree stored in an array of fixed size as discussed in class. The second queue is a FIFO queue built in a module called disk_queue.c with the following interface:

```
typedef struct disk_queue_s *disk_queue_t;
typedef struct request_s *request_t;

disk_queue_t disk_queue_init(void);
int disk_queue_insert(disk_queue_t eq, request_t req);
request_t disk_queue_peek(disk_queue_t dq);
request_t disk_queue_remove(disk_queue_t dq);
int disk_queue_empty(disk_queue_t dq);
void disk_queue_finalize(disk_queue_t dq);
```

This will be implemented with your linked list module from Assignment 2.

Simulation

You will then write a program that simulates I/O requests submitted via the operating system for a disk drive. You will use an event driven simulation. You will create events (a struct) which includes an event type (an int), an event time (a float), and event data (a request_t). These will be placed on the event queue. A while loop will remove events from the event queue, and then use a switch statement to decide what to do based on the event type. Some events will cause new events to be created an added to the event queue. Some will cause old events to be re-added to the queue, and still others will free old events. If the event queue becomes full, it is an error and the simulation must be re-run with a larger event queue. When there are no more events in the event queue, the simulation is over.

Events

```
typedef struct event_s
{
     int event_type;
     double event_time;
     request_t request;
} event_t;
```

Each time you remove an event from the event queue set global time t to that event's event_time. Each time you schedule an event, set it's even time to current time t plus some value as given below.

REQUEST_SUBMIT

When this event happens:

- First check if 1,000,000 requests have been processed,
    - if not compute when the next request should arrive using using randsim_exp(), schedule a new REQUEST_SUBMIT event,
- Create a new request,
    - use a randsim_gauss() to determine the track the request is on, then put the request in the queue for the DQ,
    - Set the arrival_time of the request,
- If there were no requests in the DQ before this request
    - Schedule a DISK_READY event at time t + 0.

DISK_READY

When this event happens:

- Peek at the head of the DQ, compute how long the request has been in the queue and update the start_time of the request.
- Use the current track variable and the track field of the request to determine how long it will take to process that IO,
    - update the value finish_time for the request,
    - update the current track,
- Schedule a new REQUEST_DONE event.

REQUEST_DONE

When this event happens:

- Remove the request at the head of the DQ,
- See of there are any more requests in the DQ,
    - if so schedule another DISK_READY event at time t + 0,
- Update global statistics,

- Free the request struct

The random number generation codes will be provided, and each will have some parameters that can be set to create different results. Run the simulation for several different parameter settings to see how they affect results.

Requests

```
typedef struct request_s
{
     int track;
     double arrival_time;
     double start_time;
     double finish_time;
} request_t;
```

Each request is for one or more blocks on a track. Most of the time is taken up moving the disk head from the current track to a new track before we read or write. We use a random number generator to pick the track for each request. Then we compare the requested track to the current track (where the head is now). You will need a variable that keeps up with where the last request left the head. You will be provided a function to compute seek time from these two track numbers. Very nearby tracks are pretty quick to change. Distant tracks take a time depending on the distance travelled. This is used to select the time between the Disk Ready and the Request Done events.

When a request is created during the Request Submit event, record the current time in the arrival_time slot. When the IO starts at the Disk Ready event record the time in the start_time slot, and when IO is done at the Request Done event record the time in the finish_time slot. Queue time is the difference between the arrival and start times, IO time is the difference between the start and finish times, and total time is the difference between the arrival and finish times. You want to find the min, max, and average of those three times over all of the requests. You cannot compute the average incrementally, so we record the dum of the times over the simulation and then divide by the number of requests. Report all of these times at the end of the simulation.