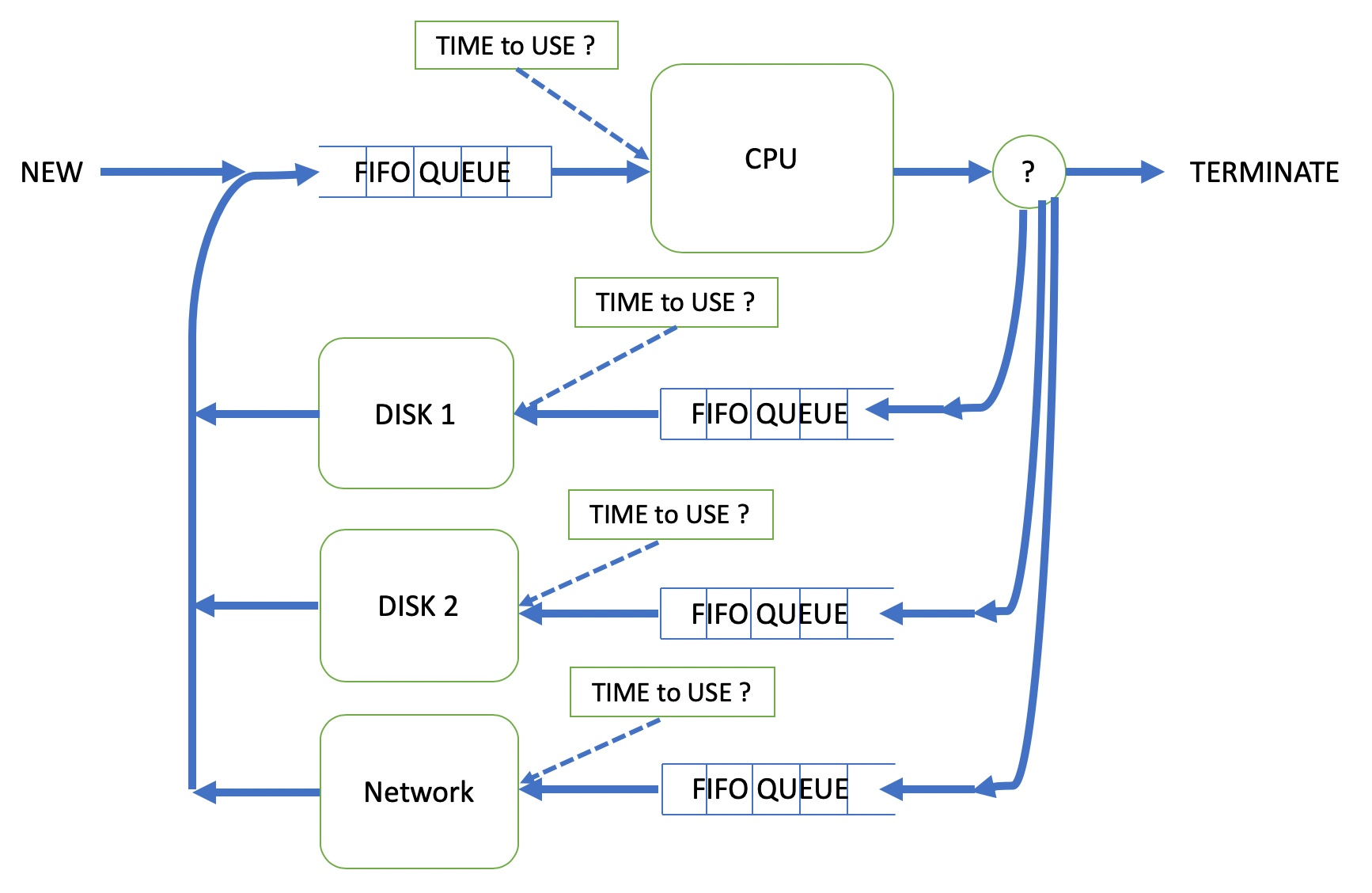
-------------------------------Read Me---------------------------------

This project is based on simulating the performance of computer's CPU and its interactions with various operating system resources, which in this case are 2 disks, and a network. ­ Processes demand and are given access to OS resources based on their time of arrival. However, there much more priorities than time in the real world. Basically, the program somewhat mimics how processes move in a computer system.

Data structures used in this program are a linked listed priority queue and A link-listed regular queue (first in first out)

Here’s a diagram of what the program should look like



To Model this program, I represented the different resources as states as shown in the diagram. So, there are 4 different FIFI queues for the CPU, Disk 1, disk 2 and the network. These queues store processes. Also, there is a priority queue which determines which process is worked on based on its arrival time. I implemented my queues through linked lists

For the actual coding:

I made a struct for the events going into the priority queue which carried and event time, job ID, and event type.

I represented the various event types ad integers. Alongside Booleans to represent the occupancy of a resource(queue).

I added the first event to the priority queue before anything, so I have an event to pop when I run my main while loop (referenced below)

I ran the entire code block based on a while loop if the priority queue was not empty and time was less than the finish time.

I then randomly generated time for that event to arrive the CPU(state) because the CPU is idle.

Every state led to the generation of another event or an enqueue of an process to the FIFO queues.These are my events:

int job\_arrival =1;

int job\_arrive\_cpu = 2;

int job\_exit\_cpu = 3;

int job\_terminates = 4;

int job\_arrive\_disk1 = 5;

int job\_exit\_disk1 = 6;

int job\_arrive\_disk2 = 7;

int job\_exit\_disk2 = 8;

int job\_arrive\_network = 9;

int job\_exit\_network = 10;

int jobs\_done = 11;

If a process was dequeued for use, the new appropriate event associated with that process was generated.

Processes leaving the system were dequeued when the CPU finishes with a process.

The program kept going running until I hit the hardcoded finish time mark because events were always being created or processes enqueued and associated with an event.

I wrote the description of the events, the Job Ids and progressive times to the LOG.txt file to keep a history of everything that transpired when I popped and event off the priority queue.

­­­­