/\*

\* File Name : driver.cpp

\* Author(s) : Francesco Polizzi, Katie Schaffer, Jeremy Viner, Hein Htet Zaw

\* Date Created : 26 April 2016

\* Date Last Modified : 6 May 2016

\*

\* Description : Main routine of the program; reads the data files and runs

\* the simulation.

\*

\*/

//libraries to include

#include <iostream>

#include <fstream>

#include <iomanip>

#include "simulation\_header.h"

using namespace std;

// Declare tracking variables

int total\_jobs\_run; // Total jobs run

double total\_response\_time; // Total response time

double total\_productive\_time; // Total productive time

double total\_turnaround\_time; // Total turnaround time

double total\_switch\_time; // Total time spent context switching

double total\_ltq\_wait; // Total time spent waiting in longterm queue

double total\_stq\_wait; // Total time spent waiting in shortterm queue

double total\_ioq\_wait; // Total time spent waiting in the IO queue

int sys\_clock; // Current system time (in clock ticks)

/\* main

\* Author(s): Francesco Polizzi, Katie Schaffer, Jeremy Viner, Hein Htet Zaw

\* Date Created: 28 April 2016

\* Last revised: 10 May 2016

\*

\* Description: Primary simulation routine; initializes counters and variables, reads input file,

\* calls all functions to managed parts of the computer, and prepares output

\* data for printing

\*/

int main() {

//////////////////////////////////////////////////////////////////////////////////////////

/// STEP 1 - Initialize

//////////////////////////////////////////////////////////////////////////////////////////

// Initialize tracker variables to 0

total\_stq\_wait = 0;

total\_jobs\_run = 0;

total\_response\_time = 0;

total\_productive\_time = 0;

total\_turnaround\_time = 0;

total\_switch\_time = 0;

total\_stq\_wait = 0;

total\_ltq\_wait = 0;

total\_ioq\_wait = 0;

sys\_clock = 0;

// Declare counter variables

int jobs\_admitted = 0; // Counts number of jobs admitted so far

int job\_timer = 0; // Keeps track of the time between job arrivals

// Simulation devices

longQueue longterm\_queue; // Longterm queue

shortQueue shortterm\_queue; // Shortterm queue

ioQueue io\_queue; // IO queue

IOdevice io\_device; // IO device

CPU cpu; // CPU

// Initialize flags and flag container

FlagContainer flags;

flags.jobs\_in\_system = 0;

flags.incoming\_job = false;

flags.interrupt = false;

// Initialize IO device values

io\_device.available = true;

io\_device.complete = false;

io\_device.job\_finished = false;

io\_device.timer = 0;

// Initialize CPU values

cpu.ready = true;

cpu.timer = 0;

cpu.complete = false;

cpu.processing\_stopped = false;

cpu.suspended = false;

// Initialize our job and jobs list

job tempJob;

job\* current\_job;

job job\_list[150];

// Initialize data files

ifstream infile("SIM\_DATA.txt", ios::in); // Onput file

ofstream outfile("Output.txt", ios::out); // Output file

//initialize our reading flag and job count

bool reading = true;

int job\_count = 0;

int jobs\_entering\_system=0;

//////////////////////////////////////////////////////////////////////////////////////////

/// STEP 2 - Get data from input file

//////////////////////////////////////////////////////////////////////////////////////////

// Read and process data from our file

while (reading) {

// Create a new job

tempJob = \*new job();

// Read in job information

infile >> tempJob.num;

infile >> tempJob.length;

infile >> tempJob.inter\_arrival;

infile >> tempJob.io\_burst;

// Initialize other job variables

tempJob.burst\_num = 0;

tempJob.response = -1;

// Initialize burst list to all -1

for (int burst\_num = 0; burst\_num < cpu\_burst\_max; burst\_num++) {

tempJob.cpu\_burst[burst\_num] = -1;

}

// Next value to read could be burst or sentinel

int temp\_input;

infile >> temp\_input;

// Continue to read until sentinel

while (temp\_input > 0){

// Add CPU burst to temp job cpu burst array

tempJob.cpu\_burst[tempJob.burst\_count]=temp\_input;

tempJob.burst\_count++;

infile >> temp\_input;

}

// Add new job to job array

job\_list[job\_count] = tempJob;

job\_count++;

// Confirm we've reached the sentinel and finish reading

if (temp\_input == -1) {

reading = false;

}

}

//////////////////////////////////////////////////////////////////////////////////////////

/// STEP 3 - Get first job into the system

//////////////////////////////////////////////////////////////////////////////////////////

// Update job timer

job\_timer++;

// When a job enters the system

if (job\_list[jobs\_admitted].inter\_arrival == job\_timer) {

// Set job flag to true

flags.incoming\_job = true;

// Get reference to job

current\_job = &job\_list[total\_jobs\_run];

// Record time of arrival

current\_job->arrival = sys\_clock;

// Reset job\_timer to zero

job\_timer = 0;

// Update counter of jobs admitted

jobs\_admitted++;

// Increment number of jobs currently int the system

jobs\_entering\_system++;

flags.jobs\_in\_system++;

}

//////////////////////////////////////////////////////////////////////////////////////////

/// STEP 4 - Process incoming jobs until all are processed

//////////////////////////////////////////////////////////////////////////////////////////

// Process while there are jobs to process

while(total\_jobs\_run < job\_count) {

// Manage all parts of the computer

manage\_ltq(longterm\_queue, current\_job, flags);

manage\_stq(shortterm\_queue, longterm\_queue, &io\_device, flags);

manage\_cpu(&cpu, shortterm\_queue, flags);

manage\_ioq(io\_queue, &cpu);

manage\_iodevice(&io\_device, io\_queue, flags);

// Increment clock

sys\_clock++;

// Check for incoming processes.

// When a job enters the system...

if (job\_list[jobs\_admitted].inter\_arrival <= job\_timer && !longterm\_queue.isFull()) {

// Set job flag to true

flags.incoming\_job = true;

// Get reference to job

current\_job = &job\_list[jobs\_entering\_system];

// Record time of arrival

current\_job->arrival = sys\_clock;

// Reset job\_timer to zero

job\_timer = 0;

// Increment admitted job count

jobs\_admitted++;

// Increment more\_jobs

jobs\_entering\_system++;

flags.jobs\_in\_system++;

}

// Update job timer

job\_timer++;

}

//////////////////////////////////////////////////////////////////////////////////////////

/// STEP 5 - Compile results and print to output file

//////////////////////////////////////////////////////////////////////////////////////////

// Process accumulated data

double total\_time = total\_switch\_time + sys\_clock;

double avgLTQ = avg\_ltq(total\_jobs\_run, total\_ltq\_wait);

double avgSTQ = avg\_stq(total\_jobs\_run, total\_stq\_wait);

double avgIOQ = avg\_ioq(total\_jobs\_run, total\_ioq\_wait);

double avgResponse = avg\_response\_time(total\_jobs\_run, total\_response\_time);

double avgTurnaround = avg\_turnaround\_time(total\_jobs\_run, total\_turnaround\_time);

double cpuUtilization = cpu\_utilization(total\_productive\_time, sys\_clock);

double contextSwitchTime = total\_switch\_time;

double systemThroughput = ((double)total\_jobs\_run) / ((double)total\_time);

// Print header before printing anything

print\_header(outfile);

// Print "First in First Out" results

print\_output("First in First Out", total\_time, contextSwitchTime,

cpuUtilization, avgResponse, avgTurnaround, systemThroughput, avgLTQ,

avgSTQ, avgIOQ, outfile);

// Indicate end of output at the end

print\_footer(outfile);

return 0;

}