```cpp
/*

 Khaled Sharafaddin
 Kristina Reddy
 Qi Zhang
 Program Name: Project OS3

 Purpose: The purpose of this project is to simulate the workflow of the CPU scheduling based on the Round Robin scheduling algorithm.

 Input:
    1) Input from keyboard:
        - Input filename
        - The total simulation time (in integer seconds)
        - The quantum size (in integer milliseconds; usually between 10 and 100)
        - The number of processes allowed in the system (degree of multiprogramming - how many jobs are in the system)
    2) An input text file contains incomingQueue jobs. The first line is an integer that represents the total number of lines (jobs) in the file. Each subsequent line has four integers: start time of the job, PID, the probability of I/O requests, and the job length.

 Output:
    1) Output on console:
        - Prompt to enter input filename, simulation time, quantum size and the degree of multiprogramming
        - Error messages while file not found
        - Throughput (number of jobs completed during the simulation)
        - Number of jobs still in system
        - Number of jobs skipped
        - Average job length excluding I/O time
        - Average turnaround time
        - Average waiting time per process
        - Percentage of time CPU is busy (CPU utilization)

 - We have abided by the Wheaton College honor code in this work.
*/

#include<iostream>
#include<fstream>
#include<stdlib.h>
#include<time.h>
#include<queue>
#include<math.h>
using namespace std;


const int PENALTYOUTCPU = 4;  // Constant value indicating the penalty of incomplete job being swapped out of the CPU

// Define a class to hold all the relevant values for each job
class jobs
{
public:
    int jobStartTime;
    int jobPID;
    int jobProbIORequest;
    int jobLength;
    int jobLengthOriginal;
    bool firstEnter; // Initial false in main()
    int ioLength;  // Initial 0 in main()

};

//Define a class to hold the three queues;
```

```cpp
class queues
{
public:
    queue<jobs> incomingQueue;
    queue<jobs> readyQueue;
    queue<jobs> ioQueue;

};

int randomNumber(int start, int endtime)
{
    /* This function generates the random number within the limits of start and endtime
     - Pre-condition: two integers start and endtime are given
     - Post-condition: random number is generated between the limits
     - Return: random number
     */

    // A variable to hold the random number
    int randomNumber;

    // Generate the random number within the limits of start and endtime
    srand(time(NULL));
    randomNumber = rand() % endtime + start;

    return randomNumber;
}

void io(queues &simulationqueues, time_t &ioTimeStart, int &ioJobLength){
    /* This function simulates the IO
     - Pre-condition: Two queues (readyQueue, ioQueue) of type jobs are defined and stored in simulationqueues
                      ioTimeStart and ioJobLength are declared
     - Post-condition: Two queues, ioTimeStart, and ioJoblength are modified
     - Return: None
     */

    jobs inIO;  // Declare a variable of type jobs to hold the job that is in the IO
    bool enterIO = true;  // A boolean to hold if the first job of the IOqueue is allowed to enter the IO

    // Check if the previous IO job has finished or not
    if(ioTimeStart != 0 ){
        // If the IO is currently busy, then set the boolean enterIO to false to prevent the job entering IO; otherwise, next the first job from the ioQueue can enter the IO
        if ((ioJobLength < (time(NULL) * 1000 - ioTimeStart))) {
            enterIO = false;
        } else {
            enterIO = true;
        }
    }

    // If the ioQueue is not empty, and the previous job has left the IO, then get the job from the front of the ioQueue
    if(!simulationqueues.ioQueue.empty() && enterIO){

        // Get a job from the front of the ioQueue
        inIO = simulationqueues.ioQueue.front();

        // Start the clock for current job in IO
        ioTimeStart = time(NULL) * 1000; // Change seconds to milliseconds

        // Pop the job after finished the IO
        simulationqueues.ioQueue.pop();

        // Check to see if the jobLength is greater than 0, then push the job into the
```

```
    readyQueue
120         if (inIO.jobLength > 0) {
121             // Generate initial random value for new process and put it at the end of the
    readyQueue queue
122             ioJobLength = randomNumber(5,25);
123             inIO.ioLength += ioJobLength;
124             simulationqueues.readyQueue.push(inIO);
125         }
126     }
127 }
128
129 int cpu(int quantum, int &throughput, int &jobsInSystem, int systemTimeAt, int
    simulationTime, queues &simulationqueues, int &totalJobLength, int &totalWaitTime, int
    &totalTurnaround, int &cpubusy, bool &jobLeftInCPU)
130 {
131     /* This function simulates the CPU
132      - Pre-condition: quantum is defined as an integer from user input
133                       throughput is an integer indicating the amount of job finished
134                       jobsInSystem is an integer indicating how many jobs are in the system
135                       systemTimeAt is an integer indicating the current system time
136                       simulationTime is an integer indicating the total simulation time
137                       Three queues (incomingQueue, readyQueue, ioQueue) of type jobs are
    defined
138                       Total job length holding the total length of jobs finished
139                       Total wait time for all the completed jobs
140                       Total turnaround time for all the completed jobs
141                       CPU busy time indicating the amount of time all jobs spent in the CPU
142                       A boolean indicating whether the CPU is currently busy or not
143      - Post-condition: throughput and jobsInSystem are passed by reference
144                        three queues are modified
145                        totalJobLength, totalWaitTime, totalTurnaround, cpubusy, jobLeftInCPU
    are passed by reference
146      - Return: Current system time
147      */
148
149     // Check to see if the readyQueue is empty or not. If not empty, then put a job from the
    readyQueue into the CPU
150     if(!simulationqueues.readyQueue.empty()){
151         // For Debugging
152         // cout << "Sys Time Before: " << systemTimeAt << endl;
153
154         // Declare a variable of type jobs to hold the job that is in the CPU, and get the
    job from the front of the readyQueue
155         jobs inCPU;
156         inCPU = simulationqueues.readyQueue.front();
157         simulationqueues.readyQueue.pop();
158
159         // Check to see if the job has already been into the CPU or not. If so, set the
    current system time to the start time of this job and change the boolean value to true
160         if (inCPU.firstEnter == false && (inCPU.jobStartTime >= systemTimeAt)){
161             systemTimeAt = inCPU.jobStartTime;
162             inCPU.firstEnter = true;
163         }
164
165         int rrandomNumber = randomNumber(1,100);  //sets a new variable to hold the random
    number to determine if it goes into IO
166         int iorandomNumber = randomNumber(0,quantum);  // Sets a new variable to hold when
    the job goes into the ioQueue
167
168         // If the current job length would not exceed the simulation time, then
169         if((systemTimeAt + quantum) <= simulationTime )
170         {
171             //if the probability of the joblength is greater than or equal to that of the
    calculated random number then the job goes into the ioQueue
172             if(inCPU.jobProbIORequest >= rrandomNumber)
```

```
173             {
174                 // Push this job into the ioQueue
175                 simulationqueues.ioQueue.push(inCPU);
176 
177                 // Decrement the current job length, and increment CPU busy time and current system time
178                 inCPU.jobLength -= iorandomNumber;
179                 cpubusy += iorandomNumber;
180                 systemTimeAt += iorandomNumber;
181 
182                 // For Debugging
183                 // cout << "CPU -> IO 1" << endl;
184             }
185             else{
186                 if (inCPU.jobLength > quantum){
187                     // Decrement the current job length, and increment CPU busy time and current system time
188                     inCPU.jobLength = inCPU.jobLength - quantum;
189                     cpubusy += quantum;
190                     systemTimeAt = systemTimeAt + quantum + PENALTYOUTCPU;
191 
192                     // Push this job into the readyQueue
193                     simulationqueues.readyQueue.push(inCPU);
194 
195                     // For Debugging
196                     // cout << "readyQueue -> CPU"  << endl;
197                 }
198 
199                 else{
200                     // As this job is finished, increment the throughput, CPU busy time, and current system time
201                     throughput++;
202                     cpubusy += inCPU.jobLength;
203                     systemTimeAt = systemTimeAt + inCPU.jobLength + PENALTYOUTCPU;
204 
205                     // Increment the totalJobLength, totalWaitTime, and totalTurnaround
206                     totalJobLength += inCPU.jobLengthOriginal;
207                     totalWaitTime += systemTimeAt - inCPU.jobLengthOriginal;
208                     totalTurnaround += systemTimeAt - inCPU.jobStartTime + inCPU.ioLength;
209 
210                     // If the incomingQueue is not empty, then push the first job to the readyQueue, and remove it from the incomingQueue
211                     if(!simulationqueues.incomingQueue.empty())
212                     {
213 
simulationqueues.readyQueue.push(simulationqueues.incomingQueue.front());
214                         simulationqueues.incomingQueue.pop();
215 
216                         // For Debugging
217                         // cout << "incomingQueue -> readyQueue"  << endl;
218                     }
219                 }
220             }
221         }
222         // If the current job length would exceed the simulation time, then
223         else
224         {
225             //if the probability of the joblength is greater than or equal to that of the calculated random number, and the time entering the IO would not exceed the total simulation time, then the job goes into the ioQueue
226             if(inCPU.jobProbIORequest >= rrandomNumber && iorandomNumber < (simulationTime - systemTimeAt))
227             {
228                 // Push this job into the ioQueue
229                 simulationqueues.ioQueue.push(inCPU);
```

```
230
231                 // Decrement the current job length, and increment CPU busy time and current system time
232                 inCPU.jobLength -= iorandomNumber;
233                 cpubusy += iorandomNumber;
234                 systemTimeAt += iorandomNumber;
235
236                 // For Debugging
237                 // cout << "CPU -> IO 2" << endl;
238             }
239             else{
240                 // If the job length left is smaller than or equal to the simulation time left, then this job is able to finish within the total simulation time
241                 if (inCPU.jobLength <= (simulationTime - systemTimeAt)) {
242                     // As this job is finished, increment the throughput, CPU busy time, and current system time
243                     throughput++;
244                     cpubusy += inCPU.jobLength;
245                     systemTimeAt = systemTimeAt + inCPU.jobLength + PENALTYOUTCPU;
246
247                     // Increment the totalJobLength, totalWaitTime, and totalTurnaround
248                     totalJobLength += inCPU.jobLengthOriginal;
249                     totalWaitTime += systemTimeAt - inCPU.jobLengthOriginal;
250                     totalTurnaround += systemTimeAt - inCPU.jobStartTime + inCPU.ioLength;
251                 }
252                 // The job length left is greater than the simulation time, then this job will not be able to finish within the simulation time, and will remain in the CPU
253                 else {
254                     // Set the systemTimeAt to the simulation time and set the boolean jobLeftInCPU to true to indicate current job has not finished yet
255                     systemTimeAt = simulationTime;
256                     jobLeftInCPU = true;
257
258                     // Increment CPU busy time
259                     cpubusy += (simulationTime - systemTimeAt);
260
261                     // For Debugging
262                     // cout << "Simulation done" << endl;
263                 }
264             }
265         }
266
267         // For Debugging
268         // cout << "-------- cpu -------\n"
269         //      << inCPU.jobStartTime << "\t"
270         //      << inCPU.jobPID << "\t"
271         //      << inCPU.jobProbIORequest << "\t"
272         //      << inCPU.jobLength << "\n"
273         //      << "-------- cpu -------\n";
274         // cout << "Sys Time After: " << systemTimeAt << "\n\n\n\n" << endl;
275     }
276
277     return systemTimeAt;
278 };
279
280 int nbjobsstillinsystem(queues simulationqueues, bool &jobLeftInCPU)
281 {
282     /* This function calculates the total number of jobs still in the system
283      - Pre-condition: Two queues of type jobs called readyQueue and ioQueue that are stored in simulationqueues
284                       A boolean indicating whether the CPU is currently busy or not
285      - Post-condition: Both queues will be empty, and jobLeftInCPU is passed by reference
286      - Return: total number of jobs still in the system
287      */
288
```

```
289     int jobsInSystem = 0;  // the variable to determine the total number of jobs still in
    the system
290 
291     // Loop through the readyQueue to find how many jobs are still in it
292     while(!simulationqueues.readyQueue.empty())
293     {
294         jobsInSystem++;
295         simulationqueues.readyQueue.pop();
296     }
297 
298     // Loop through the ioQueue to find how many jobs are still in it
299     while(!simulationqueues.ioQueue.empty())
300     {
301         jobsInSystem++;
302         simulationqueues.ioQueue.pop();
303     }
304 
305     // Check if there is a job in the CPU, and if so increment the value
306     if (jobLeftInCPU){
307         jobsInSystem++;
308     }
309 
310     return jobsInSystem;
311 }
312 
313 int totaljobsSkipped(queues simulationqueues)
314 {
315     /* This function calculates the amount of job skipped
316      - Pre-condition: a queue of type jobs called incomingQueue that is stored in
    simulationqueues
317      - Post-condition: The incomingQueue will become empty
318      - Return: number of job skipped
319      */
320 
321     int jobsSkipped = 0 ;  // total number of job being skipped
322 
323     // Loop through the incomingQueue to see how many jobs are still left in the queue
324     while(!simulationqueues.incomingQueue.empty())
325     {
326         jobsSkipped++;
327         simulationqueues.incomingQueue.pop();
328     }
329     return jobsSkipped;
330 }
331 
332 void gousie() {
333     /* This function contains an ASCII art of a ghost
334      - Pre-condition: None
335      - Post-condition: Ghost in cout
336      - Return: None
337      */
338     cout << " .-." << endl
339          << "(o o) boo!" << endl
340          << "| O \\ " << endl
341          << " \\   \\ " << endl
342          << "  `~~~' " << endl
343          << "HAPPY HALLOWEEN!!!" << endl;
344 }
345 
346 int main()
347 {
348     ifstream file;  // variable to hold the input file
349     string filename;    // variable to hold the filename
350     double simulationTime;   // Variable to hold the simulation time from the user input
351     int quantumSize;    // Variable to hold the quantum size from the user input
```

```
352     int numProcesses;   // Variable to hold the number of processes (degree of
    multiprogramming) from the user input
353     int lines;  // variable to hold the total number of lines (jobs) from the input file
354 
355     // Prompt user for input filename and open file
356     cout << "Please enter the file name: ";
357     cin >> filename;
358     file.open(filename.c_str());
359 
360     // If the file is open, then
361     if(file.is_open())
362     {
363         // declare variables and initialize to 0
364         int throughput = 0;     // Number of finished jobs
365         int jobsInSystem = 0;   // Number of jobs in the system
366         int systemTimeAt = 0;   // The current system time
367         int jobsSkipped = 0;    // Number of jobs skipped
368         queues simulationqueues; //declare a variable of type queues
369 
370         // Prompt user for simulationTime, quantumSize, and numProcesses
371         cout << "What is the desired simulation time (in seconds)? ";
372         cin >> simulationTime;
373         simulationTime = simulationTime * 1000;     // Change seconds to milliseconds
374 
375         cout << "What is the desired quantum size (in milliseconds)? ";
376         cin >> quantumSize;
377 
378         cout << "What is the number of processes allowed in the system? ";
379         cin >> numProcesses;
380 
381         // Declare a variable of type jobs to hold the each job being put into the
    incomingQueue from the file
382         jobs nextjob;
383         file >> lines;
384 
385         // Put all the values from the input file into the incomingQueue
386         for(int i=0; i<lines; i++)
387         {
388             file >> nextjob.jobStartTime
389                  >> nextjob.jobPID
390                  >> nextjob.jobProbIORequest
391                  >> nextjob.jobLength;
392 
393             // Initialize the firstEnter, ioLength and jobLengthOriginal
394             nextjob.firstEnter = false;
395             nextjob.ioLength = 0;
396             nextjob.jobLengthOriginal = nextjob.jobLength;
397 
398             // Push the job into the incomingQueue
399             simulationqueues.incomingQueue.push(nextjob);
400         }
401 
402         // Push jobs from the incomingQueue into the readyQueue within the degree of
    multiprogramming, and pop them from the incomingQueue
403         for(int j=0; j<numProcesses; j++)
404         {
405             simulationqueues.readyQueue.push(simulationqueues.incomingQueue.front());
406             simulationqueues.incomingQueue.pop();
407         }
408 
409         // Declare variables for IO
410         time_t ioTimeStart = 0;     // Indicating the starting time of the job in IO
411         int ioJobLength = 0;        // A variable indicating the previous IO job length
412 
413         // Declare variables for CPU
```

```
414         int totalJobLength = 0;     // Total job length (for all the completed jobs)
415         int totalWaitTime = 0;      // Total waiting time (for all the completed jobs)
416         int totalTurnaround = 0;    // Total turnaround time (for all the completed jobs)
417         int cpubusy = 0;            // Total time when the CPU is busy
418 
419         bool jobLeftInCPU = false;  // A boolean to hold if there is a job in the CPU
420 
421         // While not exceed the simulation time, and either the readyQueue or the ioQueue is
    not empty, then
422         while(systemTimeAt < simulationTime && (!simulationqueues.readyQueue.empty() ||
    !simulationqueues.ioQueue.empty()))
423         {
424             // call the CPU and IO functions to simulate
425             systemTimeAt = cpu(quantumSize, throughput, jobsInSystem, systemTimeAt,
    simulationTime, simulationqueues, totalJobLength, totalWaitTime, totalTurnaround, cpubusy,
    jobLeftInCPU);
426             io(simulationqueues, ioTimeStart, ioJobLength);
427         }
428 
429         // call functions to calculate jobsInSystem and jobsSkipped
430         jobsInSystem = nbjobsstillinsystem(simulationqueues, jobLeftInCPU);
431         jobsSkipped = totaljobsSkipped(simulationqueues);
432 
433         // Declare a double to hold the CPU utilization, and calculate the value
434         double cpuUtilization = (double) cpubusy / simulationTime;
435 
436         // Output all the values
437         cout << "\nThroughput (number of jobs completed during the simulation): " <<
    throughput << endl
438              << "Number of jobs still in system: " << jobsInSystem << endl
439              << "Number of jobs skipped: " << jobsSkipped << endl;
440 
441         printf("Average job length excluding I/O time: %5.2f%s \n", (double)
    totalJobLength/throughput, " (ms)");
442         printf("Average turnaround time: %5.2f%s \n", (double) totalTurnaround / throughput,
    " (ms)");
443         printf("Average waiting time per process: %5.2f%s \n", (double) totalWaitTime /
    throughput, " (ms)");
444         printf("CPU utilization (percentage of time CPU is busy): %5.2f%c \n",
    cpuUtilization * 100, '%');
445 
446         cout << endl;
447 
448         // call this function for Halloween surprise
449         gousie();
450 
451     }
452     // If file cannot be opened, then show the error message
453     else
454         cout << "sorry not a valid file" << endl;
455 
456     return 0;
457 }
```