**University of Michigan – Dearborn**

**CIS 200 – Computer Science II**

**Team Project**

**Professor Robert Mann**

**Team 14:**

Demetrius Johnson

meech@umich.edu

Jingzhuo Fang

fangz@umich.edu

December 13, 2020

Source code:

//CPU.h

//CPU CLASS CPP FILE

//D Johnson

//J Fang

//12-6-2020

#ifndef CPU\_H

#define CPU\_H

#define nullchar '\0' ///define nullchar character for easy use in program

#include "JobType\_ABC.h"

#include <vector>

using namespace std;

class CPU {

protected: //set to protected so that only the CPU\_Controller class can access these data members

vector<JobType\_ABC> CPUjobQueue\_vector{}; //use this jobQ\_vector to store jobs while the processor is in use currently processing a job //initialize vector to size 0

JobType\_ABC currentJob = { nullchar, 0, 0 }; //use this to store the data of the current job that the CPU is working on, initialize it to an empty job

int currentJobQueue\_size = 0; //use this to store/edit/easily recall the current size of the jobQ of the CPU unit (object)

int clock\_CPU\_IDLE = 0; //track idle time units of CPU using this clock timer --> need to keep track of this for the log file

int clock\_CPU\_RUNTIME = 0; //track the run time units of CPU using this clock timer --> need to keep track of this for the log file

bool CPU\_status\_isEmpty = true; //initialzie this variable to true since by default CPU status will start with no jobs; thus it is in the ready state

bool CPU\_jobQ\_status\_isEmpty = true; //track the status of job q

int jobsCompleted = 0; //will be used to keep track of the number of jobs completed by the CPU

int totalRunTime = 0; //keep track of total processing time of the CPU

int totalIdleTime = 0; //keep track of total idle time of CPU

int totalTime\_JobsInQ = 0; //track total time jobs are in the job q

int totalJobs\_held\_InQueue = 0; //track total number of jobs that got sent and held in the job queue --> need this metric to calculate average time jobs were in queue

int JobQ\_maxsize = 0; //keep track of the largest Job Q on the CPU

int totalJobsCompleted\_A = 0;

int totalJobsCompleted\_B = 0; //keep track of A B and C jobs completed

int totalJobsCompleted\_C = 0;

public:

void process\_Job(void);

bool isCPU\_status\_Idle(void); //use this function to check if CPU is idle/empty and clear (from private CPU\_status data member) and ready for another job --> returns 1 (true) if it is empty, 0 (false) if it is not empty

bool isCPU\_jobQ\_status\_Empty(void);

int get\_RUNTIME(void);

int get\_IDLETIME(void);

int get\_JobsCompleted(void);

int get\_CPU\_JobQueue\_size(void);

int get\_totalRUNTIME(void);

int get\_totalIDLETIME(void);

int get\_totalTime\_JobsInQ(void);

int get\_totalJobs\_HeldInQ(void);

int get\_avgTime\_JobsInQ(void); // == total job queue time / total number of jobs that went to the queue --> this will give you average time jobs were in queue (also known as wait time, or queue time)

int get\_JobQ\_maxsize(void);

int get\_totalJobsCompleted\_A(void);

int get\_totalJobsCompleted\_B(void);

int get\_totalJobsCompleted\_C(void);

JobType\_ABC get\_CurrJob\_Processing(void);

void add\_JobToQueue(JobType\_ABC JobToAdd); //this function will add jobs to the job queue --> if processor is empty and the job queue is empty, then the job will automatically be placed on the processor

//void set\_IDLETIME(int time);

//void set\_JobsCompleted(int setJobsCompleted);

//void set\_CPU\_JobQueue\_size(int setQ\_size);

};

#endif

//CPU.cpp

//CPU CLASS HEADER FILE

#include "CPU.h"

void CPU::process\_Job(void) { //special note: remember to use the private data members -- don't use set and get methods within the class since it is easier for the program to access the class members rather than call another function

if (CPU\_status\_isEmpty == true && CPU\_jobQ\_status\_isEmpty == true && currentJob.jobType == nullchar) { // CPU is idle; so increment idle clock and total idle time

clock\_CPU\_IDLE++;

totalIdleTime++;

return; //exit function --> no need to check below statements and waste execution time since there is nothing on the processor or in the job queue

}

if (CPU\_status\_isEmpty == true && CPU\_jobQ\_status\_isEmpty == false && currentJob.jobType == nullchar) { //enter this statement if the CPU is currently processing nothing --> get a new job and place it on processor from job q

currentJob = CPUjobQueue\_vector.at(CPUjobQueue\_vector.size() - 1); //get the last element from the vector since we set the add\_to\_jobq function to add the newest arrivals to the front of the vector //-1 since size of vector included 0

CPUjobQueue\_vector.pop\_back(); //delete job from the queue

currentJobQueue\_size--; //decrement job q size since we've added one to the CPU for processing

if (currentJobQueue\_size == 0) { CPU\_jobQ\_status\_isEmpty = true; } //check to see if after taking the job from job Q if the job Q is now empty --> if it is then set the status to empty= true

CPU\_status\_isEmpty = false; //job will be placed on the processor; thus, cpu is not empty/free

}

if (currentJob.processingTime != 0) { //use this loop as long as there is a job on the processcor --> processingTime != 0

currentJob.processingTime--; //decrement the process time of the current job that the CPU is processing

clock\_CPU\_RUNTIME++; //increment the runtime since the another job's processing unit has been processed by 1 time unit

totalRunTime++; //increment total runtime

clock\_CPU\_IDLE = 0; //ensure that since a process has been completed on the processcor that idle time is at 0

}

if (currentJob.processingTime == 0 && currentJob.jobType != nullchar) { //execute this loop when processing time of a valid job reaches 0 --> now the job is finished processing

if (currentJob.jobType == 'A') { totalJobsCompleted\_A++; }

if (currentJob.jobType == 'B') { totalJobsCompleted\_B++; } //use these 3 statments to track job types completed

if (currentJob.jobType == 'C') { totalJobsCompleted\_C++; }

CPU\_status\_isEmpty = true; //change status of CPU to empty state if the current job reaches 0 for processing time (thus it is done processing)

currentJob = { nullchar, 0, 0 }; //use this to reset the job to empty Job since the current job has now been fully processed

jobsCompleted++; //increment number of jobs completed by this CPU object

clock\_CPU\_RUNTIME = 0; //reset runtime clock timer

}

if (CPU\_jobQ\_status\_isEmpty == false) { //use this to check and track the time jobs are in job Q --> do this after every time a job is processed

totalTime\_JobsInQ++;

if (currentJobQueue\_size > JobQ\_maxsize) { JobQ\_maxsize = currentJobQueue\_size; } //use this to keep track of largest job Q recorded during the running program --> if size of q is 0, no need to execute this loop which is why we placed it inside of the outer if loop

}

}

bool CPU::isCPU\_status\_Idle(void) {

if (CPU\_status\_isEmpty == true) { return true; }

else { return false; }

}

bool CPU::isCPU\_jobQ\_status\_Empty(void) {

if (CPU\_jobQ\_status\_isEmpty == true) { return true; }

else { return false; }

}

int CPU::get\_RUNTIME(void) { return clock\_CPU\_RUNTIME; }

int CPU::get\_IDLETIME(void) { return clock\_CPU\_IDLE; }

int CPU::get\_JobsCompleted(void) { return jobsCompleted; }

int CPU::get\_CPU\_JobQueue\_size(void) { return currentJobQueue\_size; }

int CPU::get\_totalRUNTIME(void) { return totalRunTime; }

int CPU::get\_totalIDLETIME(void) { return totalIdleTime; }

int CPU::get\_totalTime\_JobsInQ(void) { return totalTime\_JobsInQ; }

int CPU::get\_totalJobs\_HeldInQ(void) { return totalJobs\_held\_InQueue; }

int CPU::get\_avgTime\_JobsInQ(void) {

if (totalJobs\_held\_InQueue == 0) { return 0; } //catch divide by 0 case

return (totalTime\_JobsInQ / totalJobs\_held\_InQueue);

}

int CPU::get\_JobQ\_maxsize(void) { return JobQ\_maxsize; }

int CPU::get\_totalJobsCompleted\_A(void) { return totalJobsCompleted\_A; }

int CPU::get\_totalJobsCompleted\_B(void) { return totalJobsCompleted\_B; }

int CPU::get\_totalJobsCompleted\_C(void) { return totalJobsCompleted\_C; }

JobType\_ABC CPU::get\_CurrJob\_Processing(void) { return currentJob; }

void CPU::add\_JobToQueue(JobType\_ABC JobToAdd) {

CPUjobQueue\_vector.insert(CPUjobQueue\_vector.begin(), JobToAdd); //add the new job at the beginning of the job queue (at the front of the vector) //in this way the oldest (first to enter queue) jobs added to the vector will be processed first

//by doing this we can also take advantage of the pop\_back function of the vector class since jobs at the end of the vector are taken out first when sent to CPU

if (CPU\_status\_isEmpty == true && CPU\_jobQ\_status\_isEmpty == true && currentJob.jobType == nullchar) { //enter this statement if the CPU is currently processing nothing

currentJob = CPUjobQueue\_vector.at(CPUjobQueue\_vector.size() - 1); //get the last element from the vector since we set the add\_to\_jobq function to add the newest arrivals to the front of the vector //-1 since size of vector includes 0

CPUjobQueue\_vector.pop\_back(); //delete job from the queue

CPU\_status\_isEmpty = false; //job will be placed on the processor; thus, cpu is not empty/free

return; //exit funcion --> nothing else to do if this if-statment executes

}

totalJobs\_held\_InQueue++; //number of jobs held in queue is incremented --> need this metric to help calculate average queue time == totaltime jobs in que / total jobs held in queue

currentJobQueue\_size++; //increment current job queue size

totalTime\_JobsInQ++; //increment the total job q time since not all jobs made it out of the queue and into the processor if you get past the above if-statement in this function

CPU\_jobQ\_status\_isEmpty = false;

}

//CPU\_Controller.h

#ifndef CPU\_CONTROLLER\_H

#define CPU\_CONTROLLER\_H

#include "CPU.h"

class CPU\_Controller :

private CPU //use this derived class as a CPU controller in order to perform functions on the CPU objects --> this way we can keep our MAIN nice and clean and easy to follow/read and leave the messy functions in this class

//use the private filter since we only want this class to have access to the protected and public members of its base class --> that means you cannot declare an object of this class and access any of the derived classes attributes

{

private:

int lastRecorded\_smallestJobQ\_value = 0; //store the last smallest value recorded from the findSmallest function --> that way the function will finish execution faster if we can keep track of the most recent smallest value

int CPUsize = 0; //use this varibale to store the size of the CPU vector so that we dont have to call the vector.size function everytime we need the size of it to use it in this class's functions

public:

int findSmallest\_CPUjobQueue(vector<CPU>& CPU\_Vec); //use this function as a search algorithm to find the CPU with the smallest jobQueue so that when jobs are added to the CPU they can be added most efficiently //will use binary search algorithm

void all\_CPUs\_process\_Job(vector<CPU>& CPU\_Vec); //this CPU\_Controller function will cause all processcors in the CPU\_vec from main to run a process on their job --> each processor will decrement jobs if they have one

void addJobFrom\_arrivalQ\_to\_CPU(vector<JobType\_ABC>& ArrivalQ, vector<CPU>& CPU\_Vec, int& CPU\_ElementNumber); //use this function to add jobs from arrival queue vector to the sepecified CPU

void set\_CPUsize(int size);

int get\_CPUsize(void);

int getSum\_allJobsInQueue\_CPUs(vector<CPU>& CPU\_Vec); //use this function to get (current queue size) the summation of the number of jobs on all the job Queues of all CPU units in the CPU vector

int getSum\_allJobsCompleted\_CPUs(vector<CPU>& CPU\_Vec); //use this function to get the summation of the number of jobs completed of all CPU units in the CPU vector

int getSum\_allJobsCompleted\_CPUs\_A(vector<CPU>& CPU\_Vec); //use this function to get the summation of the number of jobs completed of all CPU units in the CPU vector for job type A

int getSum\_allJobsCompleted\_CPUs\_B(vector<CPU>& CPU\_Vec); //use this function to get the summation of the number of jobs completed of all CPU units in the CPU vector for job type B

int getSum\_allJobsCompleted\_CPUs\_C(vector<CPU>& CPU\_Vec); //use this function to get the summation of the number of jobs completed of all CPU units in the CPU vector for job type C

int getSum\_totalTimeJobsInQueue\_CPUs(vector<CPU>& CPU\_Vec); //use this function to get the summation of total time of jobs in job queue of all CPU units in the CPU vector

int getSum\_totalAmount\_JobsHeldInQueue\_CPUs(vector<CPU>& CPU\_Vec); //use this function to get the summation of the total number of jobs that was held in the job queue for all CPUs in the CPU vector

int getSum\_totalTimeProcessing\_CPUs(vector<CPU>& CPU\_Vec); //use this function to get the summation of total time jobs were being processed of all CPU units in the CPU vector

int getSum\_totalIdleTime\_CPUs(vector<CPU>& CPU\_Vec); //use this function to get the summation of total time idle time of all CPU units in the CPU vector

int getSum\_total\_Run\_Idle\_time(vector<CPU>& CPU\_Vec); //use this function to get the summation of both all idle and all processing time of all CPUs

int getTotalAverage\_TimeJobsInQueue\_CPUs(vector<CPU>& CPU\_Vec); //get the total average time of jobs in queue for all CPUs

int getCurrAverage\_CPUs\_queueSize(vector<CPU>& CPU\_Vec); //use this function to find the current average size of the job queues of all processors

int get\_All\_CPUs\_inUse(vector<CPU>& CPU\_Vec); //use this to return the total number of CPUs currently processing a job

int get\_MaxJobs\_inQueue(vector<CPU>& CPU\_Vec); //returns the largest CPU queue size recorded during duration of the program --> this means that this function will find the CPU that has the largest max\_queue value

int getSum\_MaxJobs\_inQueue(vector<CPU>& CPU\_Vec); // use this to return the sum of all max jobs from all CPUs

};

#endif //

//CPU\_Controller.cpp

//CPU\_Controller class CPP file

#include "CPU\_Controller.h"

int CPU\_Controller::findSmallest\_CPUjobQueue(vector<CPU>& CPU\_Vec) { //will use a 1-time binary split search to find smallest value //if program runs too slow we can easily add more binary splits

//use this function to return the element (CPU number) with the smallest job Queue, while also setting the last\_recordedSmallest Job queue data member equal to the smallest value to keep track and use in other functions

int lowerElements\_searchRange = 0; //use this to search the lower half of the vector during binary search

int upperElements\_searchRange = 0; //use this int to search the upper half of the vector during binary search

//in the below 2 stamenets we will do a binary split and then proceed to search for the lowest value in each half of the CPU vector seprately

lowerElements\_searchRange = CPUsize / 2; //lower range is the 1st half of the CPUs in the CPU vector

upperElements\_searchRange = CPUsize - lowerElements\_searchRange; //upper range is the last half of the CPUs in the CPU vector --> CPUsize - lowerRange == start of 2nd half of the CPU vector

int i = 0;

int j = upperElements\_searchRange; //use these 2 variables for the for-loops

while (true) { //execute this while loop until a smallest value is found --> upon finding smallest value, one of the for-loops inside of this loop will return and thus exit this funciton

for (i = 0; i <= lowerElements\_searchRange; i++) { //search the lower elements (lower CPU elements in the CPU vector) for the smallest job queue value //has to be less than or equal to to include that final lower half value

if (CPU\_Vec.at(i).get\_CPU\_JobQueue\_size() < lastRecorded\_smallestJobQ\_value) {

lastRecorded\_smallestJobQ\_value = CPU\_Vec.at(i).get\_CPU\_JobQueue\_size(); //set last recorded so that when this function is called again we have a better change of starting closer to the smallest value

return i; //return the element (CPU number) where the smallest job q value was found

}

}

for (j = upperElements\_searchRange; j < CPUsize; j++) { //search the lower elements (lower CPU elements in the CPU vector) for the smallest job queue value

if (CPU\_Vec.at(j).get\_CPU\_JobQueue\_size() < lastRecorded\_smallestJobQ\_value) {

lastRecorded\_smallestJobQ\_value = CPU\_Vec.at(j).get\_CPU\_JobQueue\_size(); //set last recorded so that when this function is called again we have a better change of starting closer to the smallest value

return j; //return the element (CPU number) where the smallest job q value was found

}

}

lastRecorded\_smallestJobQ\_value++; //this value will increment if neither of the for-loops can find the smallest value; now the last recorded smallest value will be increased to be compared again in the next iteration

}

}

void CPU\_Controller::all\_CPUs\_process\_Job(vector<CPU>& CPU\_Vec) {

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

CPU\_Vec.at(i).process\_Job(); //run all processors in the CPU vector from main by using this function through this loop

}

}

void CPU\_Controller::addJobFrom\_arrivalQ\_to\_CPU(vector<JobType\_ABC>& ArrivalQ, vector<CPU>& CPU\_Vec, int& CPU\_ElementNumber) {

//Special note: jobs added to the arrival q from MAIN were added as: oldest jobs at loewr elements --> newest job at back of vector (upper elements) --> thus jobs added to thequeue when we perform a while loop

CPU\_Vec.at(CPU\_ElementNumber).add\_JobToQueue(ArrivalQ.at(ArrivalQ.size() - 1)); //add the job at the end of the ArrivalQ vector to the CPU jobQ --> arrival queue added elements to its queue in --> 0 element == newest job arrival

//do the -1 since size of vector includes element 0; ex: size = 5 has elements 0-4; 4 is last element

ArrivalQ.pop\_back(); //delete the job at the end of the queue which was just inserted into a CPU

}

void CPU\_Controller::set\_CPUsize(int size) { CPUsize = size; }

int CPU\_Controller::get\_CPUsize(void) { return CPUsize; }

int CPU\_Controller::getSum\_allJobsInQueue\_CPUs(vector<CPU>& CPU\_Vec) {

int sum\_AllJobs\_inQueue = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

sum\_AllJobs\_inQueue += CPU\_Vec.at(i).get\_CPU\_JobQueue\_size(); //go to each CPU during this loop and add its jobQ size to the sum of allJobs variable to find total number of jobs in the job q for all Processors

}

return sum\_AllJobs\_inQueue;

}

int CPU\_Controller::getSum\_allJobsCompleted\_CPUs(vector<CPU>& CPU\_Vec) {

int sum\_AllJobsCompleted = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

sum\_AllJobsCompleted += CPU\_Vec.at(i).get\_JobsCompleted(); //go to each CPU during this loop and add its jobs\_completed value to the sum of allJobs variable to find total number of jobs compeleted for all Processors

}

return sum\_AllJobsCompleted;

}

int CPU\_Controller::getSum\_allJobsCompleted\_CPUs\_A(vector<CPU>& CPU\_Vec) {

int completed = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

completed += CPU\_Vec.at(i).get\_totalJobsCompleted\_A();

}

return completed; //return total jobs completed

}

int CPU\_Controller::getSum\_allJobsCompleted\_CPUs\_B(vector<CPU>& CPU\_Vec) {

int completed = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

completed += CPU\_Vec.at(i).get\_totalJobsCompleted\_B();

}

return completed; //return total jobs completed

}

int CPU\_Controller::getSum\_allJobsCompleted\_CPUs\_C(vector<CPU>& CPU\_Vec) {

int completed = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

completed += CPU\_Vec.at(i).get\_totalJobsCompleted\_C();

}

return completed; //return total jobs completed

}

int CPU\_Controller::getSum\_totalTimeJobsInQueue\_CPUs(vector<CPU>& CPU\_Vec) {

int totalTime = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

totalTime += CPU\_Vec.at(i).get\_totalTime\_JobsInQ();

}

return totalTime; //return total time job queue was occupied

}

int CPU\_Controller::getSum\_totalAmount\_JobsHeldInQueue\_CPUs(vector<CPU>& CPU\_Vec) {

int JobsHeld = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

JobsHeld += CPU\_Vec.at(i).get\_totalJobs\_HeldInQ();

}

return JobsHeld; //return total jobs held in Q

}

int CPU\_Controller::getSum\_totalTimeProcessing\_CPUs(vector<CPU>& CPU\_Vec) {

int totProcTime = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

totProcTime += CPU\_Vec.at(i).get\_totalRUNTIME();

}

return totProcTime; //return total processing time

}

int CPU\_Controller::getSum\_totalIdleTime\_CPUs(vector<CPU>& CPU\_Vec) {

int totIdleTime = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

totIdleTime += CPU\_Vec.at(i).get\_totalIDLETIME();

}

return totIdleTime; //return total idle time

}

int CPU\_Controller::getSum\_total\_Run\_Idle\_time(vector<CPU>& CPU\_Vec) {

return getSum\_totalIdleTime\_CPUs(CPU\_Vec) + getSum\_totalTimeProcessing\_CPUs(CPU\_Vec);

}

int CPU\_Controller::getTotalAverage\_TimeJobsInQueue\_CPUs(vector<CPU>& CPU\_Vec) {

int totAvg = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

totAvg += CPU\_Vec.at(i).get\_avgTime\_JobsInQ();

}

return totAvg; //return total avg wait time

}

int CPU\_Controller::getCurrAverage\_CPUs\_queueSize(vector<CPU>& CPU\_Vec) {

return (getSum\_allJobsInQueue\_CPUs(CPU\_Vec) / CPUsize); //return average

}

int CPU\_Controller::get\_All\_CPUs\_inUse(vector<CPU>& CPU\_Vec) {

int sum\_CPUs\_inUse = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

if (CPU\_Vec.at(i).isCPU\_status\_Idle() == false) {//if status is not idle (idle status ==false) then we will enter this if-statement and increment the sum

sum\_CPUs\_inUse++;

}

}

return sum\_CPUs\_inUse; //return sum of CPUs currently in use

}

int CPU\_Controller::get\_MaxJobs\_inQueue(vector<CPU>& CPU\_Vec) {

int Max\_ofAll = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

if (CPU\_Vec.at(i).get\_JobQ\_maxsize() > Max\_ofAll) {//compare current max to max of the CPU element of the loop iteration number

Max\_ofAll = CPU\_Vec.at(i).get\_JobQ\_maxsize();

}

}

return Max\_ofAll; //return max

}

int CPU\_Controller::getSum\_MaxJobs\_inQueue(vector<CPU>& CPU\_Vec) {

int sumMax = 0;

for (int i = 0; i < CPUsize; i++) { //must make i < size since the CPU vector includes the 0 element in the size

sumMax += CPU\_Vec.at(i).get\_JobQ\_maxsize();

}

return sumMax; //return summed max

}

//JobType\_ABC.h

//JOB TYPE CLASS HEADER FILE

//this struct is meant to be bear bones to simply store some data

//CPP file likely not needed/not expected to be needed

#ifndef JOBTYPE\_ABC\_H

#define JOBTYPE\_ABC\_H

struct JobType\_ABC

{

char jobType;

int arrivalTime;

int processingTime;

};

#endif

//JobType\_ABC.cpp

#include "JobType\_ABC.h"

//ReadFromFile.h

//Read from file class HEADER FILE

//D Johnson

//J Fang

#ifndef READFROMFILE\_H

#define READFROMFILE\_H

#include "JobType\_ABC.h"

#include "CPU\_Controller.h"

#include <vector>

#include <iostream>

#include <fstream>

#include <stdlib.h> //use this library for functions such as rand() number function

#include <iomanip> //use this for table formatting

#define JobType\_ABC\_ByteSize sizeof(JobType\_ABC) //define the byte size of JobType\_ABC since it will not change for the duration of this program

using namespace std;

class ReadFromFile

{

private:

int totalJobsArrived\_A = 0;

int totalJobsArrived\_B = 0;

int totalJobsArrived\_C = 0;

int totalJobsArrived = 0;

public:

void openingMenu(void); //use this for opening menu of program

void re\_initFile\_newDataSet(void); //use this to initialize the data file that this program uses

void userDecide\_re\_initFile(void); //use this to give user decision about program initialization concerning the data file that the program uses

void displaySnippet\_dataFile(void); //use this to display a snippet size specified by the user to see what is in the currently loaded data file for the program and ensure it is formatted properly and as expected

int jobsReadFromFile(void); //use this to display how many jobs are currently stored in the data file that the program uses

void AddTo\_Queue\_FromFile(vector<JobType\_ABC>& arrvivalQueue\_vector, int& counter\_JobsReadFromFile\_seekInFile); //use this function to read jobs from the data file into the arrivalQueue vector that is in MAIN

int peekFile\_nextJob\_arrivalTime(int& counter\_JobsReadFromFile\_seekInFile); //use this function to check and return the arrival time of the next job in the data file so that we can approriately decide wether to add it to the arrivalQueue that is in MAIN

int userDecide\_setNumberOf\_CPUs(vector<CPU>& CPU\_vector); //use this function to establish the number of CPUs that the program will utilize --> will size the CPU\_vector that is in MAIN based on user decision

int get\_TotalArrivals\_A(void);

int get\_TotalArrivals\_B(void); //use these functions to return the total arrivals of each job type

int get\_TotalArrivals\_C(void);

int get\_totalArrivals(void); //get the total job arrivals

};

#endif

//ReadFromFile.cpp

//READ FROM FILE class CPP FILE

#include "ReadFromFile.h"

#include <vector>

using namespace std;

//

//Job Type Arrival Time Processing Time

// A 5 + / -1 3 + / -2

// B 10 + / -1 8 + / -2

// C 25 + / -1 11 + / -2

//Note:

//5 +/- 1 is 4, 5, 6 --> 4 + rand() % 3

//Arrival Time is time since the last job of the same Type

void ReadFromFile::openingMenu(void) {

cout << "---------------------------------------WELCOME---------------------------------------------\n\n";

cout << "The initialization of this program will read and write data to a JobDatabase binary file: 'JobDatabase.dat'.\n";

cout << "Below is a snippet of the current database on file for 4150 Jobs (2500 type A, 1200 type B, 450 type C) : \n";

}

void ReadFromFile::re\_initFile\_newDataSet(void) {

int curr\_SmallestClockVal = 0; //use this to find the smallest clock value to be written to the file for any specified job type that matches this value

int jobCounter\_A = 2500; //use this to write jobs to the file until total required number of A jobs have been written

int jobCounter\_B = 1200; //use this to write jobs to the file until total required number of B jobs have been written

int jobCounter\_C = 450; //use this to write jobs to the file until total required number of C jobs have been written

int clockCounter\_JobA = 0; //use this to store the next arrival time for jobA

int clockCounter\_JobB = 0; //use this to store the next arrival time for jobB

int clockCounter\_JobC = 0; //use this to store the next arrival time for jobC

int assignProcessTime = 0; //use this to assign a processing time to each job written to the file, based on the range of random process times given for the job type

//open the file and seek to a position that will skip the 'o-position' relative to the sizeof JobType\_ABC

ofstream JobDatabase\_write("JobDatabase.dat", ios::out | ios::binary); //NOTE:if no file exists, you may not be able to open in binary or using any other addtional modes

//must open the uncreated file first using only the default constructor (no additional paramters other than ios::out and the file name),

// then close it and reopen (in binary for example) the newly created file. (idk if this is because of a c++ error but it worked as long as i didnt set the int mode (the last parameter)

if (!JobDatabase\_write.is\_open()) {

cout << "\nunable to open 'JobDatabase.dat' file.\n";

}

else { cout << "\n~Random Access File 'JobDatabase.dat' opened successfully~\n"; }

JobDatabase\_write.seekp(1 \* JobType\_ABC\_ByteSize, ios::beg); //skip the 0-position and start writing jobs in the file at position 1

JobType\_ABC Job = { nullchar, 0, 0 }; //use this data structure object to write data to the file and temporarily store the current values to be written to teh file

//use below three statements to establish the first set of random values for each job

clockCounter\_JobA += 4 + rand() % 3; //rand() % number function gives a psuedo random number in the range 0-2 //so this line of code provived a number 5+/-1 == 4 + (0-2) --> 4, 5 or 6

clockCounter\_JobB += 9 + rand() % 3; //rand() % number function gives a psuedo random number in the range 0-2 //so this line of code provived a number 10+/-1 == 9 + (0-2) --> 9, 10, or 11

clockCounter\_JobC += 24 + rand() % 3; //rand() % number function gives a psuedo random number in the range 0-2 //so this line of code provived a number 25+/-1 == 24 + (0-2) --> 24, 25, or 26

while (jobCounter\_A > 0 || jobCounter\_B > 0 || jobCounter\_C > 0) {//use this to initialize and write all 4150 jobs to the data file //loop will continue until all jobCount value == 0

curr\_SmallestClockVal = clockCounter\_JobA;

if (curr\_SmallestClockVal > clockCounter\_JobB) { curr\_SmallestClockVal = clockCounter\_JobB; }

if (curr\_SmallestClockVal > clockCounter\_JobC) { curr\_SmallestClockVal = clockCounter\_JobC; }

//use the above 3 lines of code to determine the current largest value to assign jobs with and write to the file

if (clockCounter\_JobA == curr\_SmallestClockVal && jobCounter\_A > 0) {

//use this if-statment when clock counter for a specified job matches smallest current clock value and as long as number of jobs written for the specified job type is not at 0 --> if 0, then total required jobs for the type is already written to file

assignProcessTime = 1 + rand() % 5; //use rand() function to assign value in range 1 + 0-4 --> to produce a value in range 1-5 (Job A Process time == 3 +/- 2)

Job = { 'A', clockCounter\_JobA, assignProcessTime };

JobDatabase\_write.write(reinterpret\_cast<const char\*>(&Job), JobType\_ABC\_ByteSize); //reinterpret JobType\_ABC Job data structure address as a Char address object in order to write data to the file

clockCounter\_JobA += 4 + rand() % 3; //increment counter so that it can later be compared again to curr\_smallest clock counter value with its new value which will be used for the next A-job arrival

jobCounter\_A--; //decrement the jobA counter since an A-job arrival has been written to the file

}

if (clockCounter\_JobB == curr\_SmallestClockVal && jobCounter\_B > 0) {

//use this if-statment when clock counter for a specified job matches smallest current clock value and as long as number of jobs written for the specified job type is not at 0 --> if 0, then total required jobs for the type is already written to file

assignProcessTime = 6 + rand() % 5; //use rand() function to assign value in range 6 + 0-4 --> to produce a value in range 6-10 (Job B Process time == 8 +/- 2)

Job = { 'B', clockCounter\_JobB, assignProcessTime };

JobDatabase\_write.write(reinterpret\_cast<const char\*>(&Job), JobType\_ABC\_ByteSize); //reinterpret JobType\_ABC Job data structure address as a Char address object in order to write data to the file

clockCounter\_JobB += 9 + rand() % 3; //increment counter so that it can later be compared again to curr\_smallest clock counter value with its new value which will be used for the next B-job arrival

jobCounter\_B--; //decrement the jobB counter since a B-job arrival has been written to the file

}

if (clockCounter\_JobC == curr\_SmallestClockVal && jobCounter\_C > 0) {

//use this if-statment when clock counter for a specified job matches smallest current clock value and as long as number of jobs written for the specified job type is not at 0 --> if 0, then total required jobs for the type is already written to file

assignProcessTime = 9 + rand() % 5; //use rand() function to assign value in range 9 + 0-4 --> to produce a value in range 9-13 (Job C Process time == 11 +/- 2)

Job = { 'C', clockCounter\_JobC, assignProcessTime };

JobDatabase\_write.write(reinterpret\_cast<const char\*>(&Job), JobType\_ABC\_ByteSize); //reinterpret JobType\_ABC Job data structure address as a Char address object in order to write data to the file

clockCounter\_JobC += 24 + rand() % 3; //increment counter so that it can now be compared again to curr\_smallest clock counter value with its new value which will be used for the next C-job arrival

jobCounter\_C--; //decrement the jobC counter since a C-job arrival has been written to the file

}

if (jobCounter\_A == 0) { clockCounter\_JobA = INT\_MAX; }

if (jobCounter\_B == 0) { clockCounter\_JobB = INT\_MAX; }

if (jobCounter\_C == 0) { clockCounter\_JobC = INT\_MAX; }

//use the above 3 lines of code to set the clock counter values to very large values so that they are exlcuded from comparison when the while loop starts and is searching for the smallest value

}

JobDatabase\_write.clear(); //clear stream buffer

JobDatabase\_write.close(); //close the file

}

void ReadFromFile::userDecide\_re\_initFile(void) {

string userChoice;

cout << "\n~Program Initialization~\n";

cout << "Enter 'yes' to initiate JobDatabase.dat file to a new set of data and launch the program.";

cout << "\nEnter 'no' to leave JobDatabase.dat file as is and launch the program.\n" << "User option ('yes' or 'no'): ";

cin >> userChoice;

if (userChoice == "yes") {

cout << "\n...User has entered 'yes'....intiating JobDatabase.dat binary file to new data set.....reinitializing file\n\n......launching program...----(abort program to cancel)..\n\n";

system("pause");

cout << "\n...JobDatabase.dat file reinitialized successfully....\n\n";

re\_initFile\_newDataSet();

}

else {

cout << "\n...User has entered 'no', or user input undetermined...JobDatabase.dat binary file will NOT be reinitialized\n\n......launching program......\n\n";

system("pause");

fstream testStream("JobDatabase.dat", ios::in | ios::binary | ios::\_Nocreate);

//use fstream variable if user decides no or undetermined input, that way if file is not created it will still be created

if (testStream.is\_open()) { cout << "\n~testing file....'JobDatabase.dat' binary file opened successfully...\n\n"; }

if (!testStream.is\_open()) {

cout << "\n...Launch failed...???unable to open 'JobDatabase.dat' file....file does not exist???\n";

system("pause");

cout << "\n~Would you like the program to create the file 'JobDatabase.dat'? If file is already present but corrupt/undetected, it will be overwritten...~\n";

cout << "\nEnter the word 'proceed' to proceed with creating the file and launching the program;\notherwise, program will exit and NO FILE will be created or overwritten...\n";

cout << "User Choice: ";

userChoice.clear();

cin >> userChoice;

if (userChoice == "proceed") {

cout << "\n\n...File created or corrupt file over written....Opening 'JobDatabase.dat' file and launching program....\n\n";

system("pause");

re\_initFile\_newDataSet();

}

else { cout << "\n\n....Program failed to launch...Undefined Program Behavior expected....!!Please Abort Program!!..."; }

}

testStream.clear();

testStream.close();

}

}

void ReadFromFile::displaySnippet\_dataFile(void) {

int display\_SnippetSize = 0;

int jobNumber = 0; //use this to display on the table the corresponding job # (note job# is not apart of / a data member in the JobType class, it is merely used here for the display table

cout << "\nInput a number (in the range from 0-4150) to select the snippet size of the table to be displayed: ";

cin >> display\_SnippetSize;

display\_SnippetSize++; //increment whatever the user inputs since the table starts output at the 0 position which has no data in the file

if (cin.peek() == '\n') { cin.ignore(); } //clear the cin stream for future use

int outputHeader\_menuCounter = 1; //use this counter in order to determine when to output the display table's header so that it is easy for a user to read/identify labels when they scroll down the list

JobType\_ABC readJob{ nullchar, 0, 0 };

ifstream JobDatabase\_read("JobDatabase.dat", ios::in | ios::binary); //No\_create flag already set since this is ifstream which is used by default for reading a file (for ofstream, default does not set the No\_create flag)

if (!JobDatabase\_read.is\_open()) {

cout << "\nunable to open 'JobDatabase.dat' file.\n";

}

else { cout << "\n~Random Access File 'JobDatabase.dat' opened successfully~\n"; }

cout << "-Job\_Type-" << setw(25) << "-Arrival\_Time-" << setw(30) << "-Processing\_Time-" << setw(25) << "-Job#-" << endl << endl; //initiate top of table labeling

JobDatabase\_read.seekg(0 \* JobType\_ABC\_ByteSize, ios::beg); //use this to start at beggning of file and read the empty 0 position for output so user can see 0 position is not used for data

for (int i = 0; i < display\_SnippetSize; i++) {

if (JobDatabase\_read.eof() || JobDatabase\_read.peek() == EOF) { //use this to break out of loop if EOF bit is set

cout << "\n\n...NO MORE DATA TO READ...EOF BIT SET...end of file reached....\n\n";

break;

}

JobDatabase\_read.read(reinterpret\_cast<char\*>(&readJob), JobType\_ABC\_ByteSize); //read data of given byte\_size into readJob struct using a char\* pointer

//The function has parameters to point to the data with a char\* pointer, which is why readJob reference address has to be cast as a char\* pointer

cout << readJob.jobType << setw(30);

cout << readJob.arrivalTime << setw(30);

cout << readJob.processingTime << setw(20);

cout << jobNumber << endl;

cout << "---" << setw(30) << "------------------" << setw(30) << "----------------" << setw(25) << "------------" << endl;

jobNumber++;

outputHeader\_menuCounter++;

if (outputHeader\_menuCounter % 15 == 0) {//output table header labeling again every 15 lines

cout << endl << "-Job\_Type-" << setw(25) << "-Arrival\_Time-" << setw(30) << "-Processing\_Time-" << setw(20) << "-Job#-" << endl << endl; //initiate top of table labeling

}

}

cout << "\n..~Total number of jobs read from JobDatabase.dat file is: " << jobsReadFromFile() << "~..\n";

JobDatabase\_read.clear();

JobDatabase\_read.close(); //clear and close the file stream

}

int ReadFromFile::jobsReadFromFile(void) {

int counter\_JobsInFile = 0;

ifstream JobDatabase\_read("JobDatabase.dat", ios::in | ios::binary); //No\_create flag already set since this is ifstream which is used by default for reading a file (for ofstream, default does not set the No\_create flag)

if (!JobDatabase\_read.is\_open()) {

cout << "\nunable to open 'JobDatabase.dat' file.\n";

}

//else { cout << "\n~Random Access File 'JobDatabase.dat' opened successfully~\n"; }

JobDatabase\_read.seekg(JobType\_ABC\_ByteSize, ios::beg); //skip position 0 in file -->seek to beginning of file at the location of the first valid job (position 1) so that we can count total number of jobs written to file in the next while-loop statement

while (JobDatabase\_read.peek() != EOF) {

JobDatabase\_read.seekg(JobType\_ABC\_ByteSize, ios::cur);

//cout << JobDatabase\_read.tellg(); //used for debugging and testing

counter\_JobsInFile++;

}

JobDatabase\_read.clear();

JobDatabase\_read.close(); //clear and close the file stream

return counter\_JobsInFile;

}

void ReadFromFile::AddTo\_Queue\_FromFile(vector<JobType\_ABC>& arrvivalQueue\_vector, int& counter\_JobsReadFromFile\_seekInFile) {

counter\_JobsReadFromFile\_seekInFile++; //increment this int variable since upon entering this function we are now executing the reading of the next value in the file into the arrivalQueue vector

JobType\_ABC newJobFromFile{ nullchar, 0, 0 }; //create an object of type Job in order to store the next job read from the file into this object so that we can add it to the arrival job vector

ifstream JobDatabase\_read("JobDatabase.dat", ios::in | ios::binary); //No\_create flag already set since this is ifstream which is used by default for reading a file (for ofstream, default does not set the No\_create flag)

if (!JobDatabase\_read.is\_open()) {

cout << "\nunable to open 'JobDatabase.dat' file.\n";

} //open the file

//else { cout << "\n~Random Access File 'JobDatabase.dat' opened successfully~\n"; } //check to make sure file is open

JobDatabase\_read.seekg(counter\_JobsReadFromFile\_seekInFile \* JobType\_ABC\_ByteSize, ios::beg); //start at beginning of file and then move the read pointer of the ifstream variable to the location of the job to be read using the counter

JobDatabase\_read.read(reinterpret\_cast<char\*>(&newJobFromFile), JobType\_ABC\_ByteSize); //read data of given byte\_size into readJob struct using a char\* pointer

arrvivalQueue\_vector.insert(arrvivalQueue\_vector.begin(), newJobFromFile); //insert new jobs into the queue at the front of queue since processcor will end up taking the job at the end of it's queue (highest element) and process it first

totalJobsArrived++; //increment jobs arrived

if (newJobFromFile.jobType == 'A') { totalJobsArrived\_A++; }

if (newJobFromFile.jobType == 'B') { totalJobsArrived\_B++; } //use these three statements to update jobType trackers for the job that arrived

if (newJobFromFile.jobType == 'C') { totalJobsArrived\_C++; }

//cout << "\n~..1 job has been read from the data file into the arrivalQueue\_vector..~...~New size of arrivalQueue is: " << arrvivalQueue\_vector.size() << " ~...\n";

JobDatabase\_read.clear();

JobDatabase\_read.close(); //clear and close the file stream

}

int ReadFromFile::peekFile\_nextJob\_arrivalTime(int& counter\_JobsReadFromFile\_seekInFile) { //pass in the JobCounter from Main--> for this function we only want to use that value to check the next job in the file, and nothing else.

int arrivalTime = 0;

JobType\_ABC Job\_PeekedFromFile{ nullchar, 0, 0 }; //create an object of type Job in order to store the next job read from the file into this object so that we can add it to the arrival job vector

ifstream JobDatabase\_read("JobDatabase.dat", ios::in | ios::binary); //No\_create flag already set since this is ifstream which is used by default for reading a file (for ofstream, default does not set the No\_create flag)

if (!JobDatabase\_read.is\_open()) {

cout << "\nunable to open 'JobDatabase.dat' file.\n";

} //open the file

//else { cout << "\n~Random Access File 'JobDatabase.dat' opened successfully~\n"; } //check to make sure file is open

JobDatabase\_read.seekg((counter\_JobsReadFromFile\_seekInFile + 1) \* JobType\_ABC\_ByteSize, ios::beg); //start at beginning of file and then move the read pointer of the ifstream variable to the location of the job to be read using the counter

//use +1 since we wish to see the NEXT arrival time in the file, not the current one

JobDatabase\_read.read(reinterpret\_cast<char\*>(&Job\_PeekedFromFile), JobType\_ABC\_ByteSize); //read data of given byte\_size into readJob struct using a char\* pointer

arrivalTime = Job\_PeekedFromFile.arrivalTime; //set the arrival time of the peeked job equal to arrivalTime integer variable so it can be returned by the function

JobDatabase\_read.clear();

JobDatabase\_read.close(); //clear and close the file stream

return arrivalTime;

}

int ReadFromFile::userDecide\_setNumberOf\_CPUs(vector<CPU>& CPU\_vector) {

int userDecision\_numCPU = 1; //defualt is 1 CPU; otherwise the program will not run

cout << "\nPlease select the number of CPUs that you would like to use for this program (default is 1): ";

cin >> userDecision\_numCPU;

if (cin.peek() == '\n') { cin.ignore(); } //clear cin istream buffer

cout << "\nYou have selected to use " << userDecision\_numCPU << " CPUs for this program.........~Finishing Program Launch~.........\n";

CPU\_vector.resize(userDecision\_numCPU); //resize the CPU vector to the size speicified by user so that program will utilize the desired number of CPUs

if (CPU\_vector.size() == userDecision\_numCPU) {

cout << "\n\n~CPU\_vector resized successfully~\n\n\t...Program Launched Successfully...\n\n";

system("pause");

}

else { cout << "~..CPU\_vector resize failed..~\n"; }

return userDecision\_numCPU;

}

int ReadFromFile::get\_TotalArrivals\_A(void) { return totalJobsArrived\_A; }

int ReadFromFile::get\_TotalArrivals\_B(void) { return totalJobsArrived\_B; }

int ReadFromFile::get\_TotalArrivals\_C(void) { return totalJobsArrived\_C; }

int ReadFromFile::get\_totalArrivals(void) { return totalJobsArrived; }

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu

// Debug program: F5 or Debug > Start Debugging menu

//Main.cpp

// CIS\_200\_TEAM14\_PRJ.cpp : This file contains the 'main' function. Program execution begins and ends there.

//Demetrius Johnson

//Jingzhuo Fang

//special note: We will use vectors for this project; even though a linked list is better to use when you have to delete elements from the container at the front of the list (element 0);

//however, our vector will remain relatively small and so it will have an affect on the Big 0 time complexity (number of executions) of our project, but not enough in our judgement to use a linked list; we will use the ease of std vector class

//In the future, we could easily revamp this program and replace any vectors and all vector functionality with a more efficient linked list

//Another note: if any jobs in arrivalQueue match the clock they will move to a CPU to be processed --> if no CPU is available they will move into the jobQueue

/\*

Primary Logic (Clock Manager):

1) Check data file for any arrival times that match the clock and add it to the ArrivalQ vector

2) Flush the arrivalQueue vector --> send all jobs from arrival Queue to the CPU Vector to be processed (or to the CPU(s) with the smallest JobQ)

3) Run all processors

4) incremenet clock manager

5) repeat 1-4

\*/

#include "ReadFromFile.h"

using namespace std;

int main()

{

cout << "\nProgram by: Demetrius Johnson && Jingzhou Fang --UM DEARBRON F2020 SEMESTER --PROF ROBERT MAN CIS 200\n";

int counter\_logFile = 0;

int CPUtotalTime\_Idle\_Running = 0;

int nextArrivalTime\_inFile = 0; //use this variable to store what is peeked as the next arrival time of the next job in the file

int CPUvec\_smallestJobQ = 0; //use this variable for storing the smallest jobq in the CPU vector

int counter\_CurrJobsReadFromFile = 0; //use this variable in order to keep track of the number of jobs that has been read from the data file --> this value will also be used to manipulate the pointer in a file stream variable reading the data file

int counter\_ClockManager\_TimeUnit = 0; //use this as the clock management variable in order to track and perform clock cycles

int counter\_JobsCompleted\_Total = 0; //use this to keep track of the total jobs completed by all processors in order that the clockManager Loop has a basis for which to exit and stop iterating through the while-loop

int totalJobs\_inCurrLoadedDatFile = 0; //use this variable to store the total number of jobs read from the data file used for the program --> we will use this variable to determine when we will exit the clockManager loop

int size\_of\_CPU\_vec = 0; //use this to store the size of CPU vector in order to use this variable by reference in some of the functions that will be used in this program in order to boost program performance

vector<JobType\_ABC> arrivalQueue\_vector{}; //reading from file into this vector the next jobs that are wating for the matching clock value in order to enter the jobQueue\_vector

vector<CPU> CPU\_vector; //use this vector to set the number of CPU's the program will use based on user input at initialization of program; size of this vector will not change for the duration of this program

ReadFromFile RFF; //delcare ReadFromFile variable for use of the duration of this program to handle reading the data file

CPU\_Controller CPU\_C; //use this object which is a derived object of the base class CPU --> need this class to perform algorithms on the CPU\_vector such as adding jobs, checking status, and using functions to distribute jobs to the CPUs jobQueue vector

RFF.userDecide\_re\_initFile();

RFF.openingMenu();

RFF.displaySnippet\_dataFile();

size\_of\_CPU\_vec = RFF.userDecide\_setNumberOf\_CPUs(CPU\_vector); //store CPU size in this variable

CPU\_C.set\_CPUsize(size\_of\_CPU\_vec); //set the CPUsize data member in CPU\_controller for use in the class functions

totalJobs\_inCurrLoadedDatFile = RFF.jobsReadFromFile(); //establish total number of jobs that are in the data file to be processed --> will use this value for the loop in order to exit the loop once all jobs have been processed

ofstream LogFileStream("Logfile.txt"); //open log file and use below statements to make sure it opens and creates a new log file successfully

if (!LogFileStream.is\_open()) {

cout << "\n~Logfile Failed to create and open~\n";

system("pause");

} //ensure logfile is open

else {

cout << "\n~New Logfile Opened Successfully~...\n";

system("pause");

} //if open then let the user know opened successfully

LogFileStream << "\nProgram by: Demetrius Johnson && Jingzhou Fang --UM DEARBRON F2020 SEMESTER --PROF ROBERT MAN CIS 200\n";

/\*\*/

while (counter\_JobsCompleted\_Total < totalJobs\_inCurrLoadedDatFile) { //use this clockManager loop to execeute 1 iteration of 1 clock cycle unit

////loop until all jobs in the file have been read and processed --> based on data file and user input

//CLOCK SEQUENCE 1: Check data file for any arrival times that match the clock and add it to the ArrivalQ vector

nextArrivalTime\_inFile = RFF.peekFile\_nextJob\_arrivalTime(counter\_CurrJobsReadFromFile);

while (nextArrivalTime\_inFile == counter\_ClockManager\_TimeUnit && counter\_ClockManager\_TimeUnit != 0) { //if the next job's arrival time is equal to the clockManager, then use this loop to input the job(s) into the arrivalQ\_vector

RFF.AddTo\_Queue\_FromFile(arrivalQueue\_vector, counter\_CurrJobsReadFromFile); //this function will add the job(s) to the ArrivalQ vector and increment the jobsRead from file counter accordingly

nextArrivalTime\_inFile = RFF.peekFile\_nextJob\_arrivalTime(counter\_CurrJobsReadFromFile); //set next arrival time to be compared to clock manager again

}

//CLOCK SEQUENCE 2: Flush the arrivalQueue vector --> send all jobs from arrival Queue to the CPU Vector to be processed (or to the CPU(s) with the smallest JobQ)

while (arrivalQueue\_vector.size() != 0) {

CPUvec\_smallestJobQ = CPU\_C.findSmallest\_CPUjobQueue(CPU\_vector); //algorithm in this function will find the smallest jobQ out of all the CPUs

CPU\_C.addJobFrom\_arrivalQ\_to\_CPU(arrivalQueue\_vector, CPU\_vector, CPUvec\_smallestJobQ);

}

//CLOCK SEQUENCE 3: Process all jobs on all CPUs --> all CPUs will decrement processes and add jobs from their jobQ accordingly and track their processing metrics

CPU\_C.all\_CPUs\_process\_Job(CPU\_vector);

counter\_JobsCompleted\_Total = CPU\_C.getSum\_allJobsCompleted\_CPUs(CPU\_vector);

//CLOCK SEQUENCE 4: Increment clock manager

counter\_ClockManager\_TimeUnit++; //increment the clockManager after this loop executes so we know which clock time unit is iterating during the next loop iteration

//use the below statment to pause program to see current metrics for 500 time units and for 9500 time units

CPUtotalTime\_Idle\_Running = CPU\_C.getSum\_total\_Run\_Idle\_time(CPU\_vector);

if (

(CPUtotalTime\_Idle\_Running > 495 && CPUtotalTime\_Idle\_Running < 505)

|| (CPUtotalTime\_Idle\_Running > 9995 && CPUtotalTime\_Idle\_Running < 10005) //use these ranges to get at or close to 500 time units and 10,000 time units

|| (counter\_JobsCompleted\_Total == totalJobs\_inCurrLoadedDatFile) //output final table

)

{

//OUTPUT CURRENT METRICS

cout << "\n------------------------------------------------------------------------------";

cout << "\nClockManagement--CYCLE NUMBER--: <" << counter\_ClockManager\_TimeUnit << ">";

cout << "\nNumber of processor(s) being used : <" << size\_of\_CPU\_vec << ">";

cout << "\nCurrent total queue size : <" << CPU\_C.getSum\_allJobsInQueue\_CPUs(CPU\_vector) << ">";

cout << "\nAverage CPU\_queue size : <" << CPU\_C.getCurrAverage\_CPUs\_queueSize(CPU\_vector) << ">";

cout << "\nMaximum jobs in one CPU\_queue : <" << CPU\_C.get\_MaxJobs\_inQueue(CPU\_vector) << ">";

cout << "\nAverage time in CPU\_queue : <" << CPU\_C.getTotalAverage\_TimeJobsInQueue\_CPUs(CPU\_vector) << ">" << " time\_units";

cout << "\nTotal time jobs in CPU\_queue : <" << CPU\_C.getSum\_totalTimeJobsInQueue\_CPUs(CPU\_vector) << ">" << " time\_units" << endl;

cout << "\nTotal number of jobs A arrived : <" << RFF.get\_TotalArrivals\_A() << ">";

cout << "\nTotal number of jobs B arrived : <" << RFF.get\_TotalArrivals\_B() << ">";

cout << "\nTotal number of jobs C arrived : <" << RFF.get\_TotalArrivals\_C() << ">";

cout << "\nTotal number of jobs arrived : <" << RFF.get\_totalArrivals() << ">" << endl;

cout << "\nTotal number of jobs A completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs\_A(CPU\_vector) << ">";

cout << "\nTotal number of jobs B completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs\_B(CPU\_vector) << ">";

cout << "\nTotal number of jobs C completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs\_C(CPU\_vector) << ">";

cout << "\nTotal jobs completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs(CPU\_vector) << ">" << endl;

cout << "\nTotal time CPU(s) were processing : <" << CPU\_C.getSum\_totalTimeProcessing\_CPUs(CPU\_vector) << ">" << " time\_units";

cout << "\nTotal time CPU(s) were idle : <" << CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector) << ">" << " time\_units";

cout << "\nTotal time CPU(s) time\_Idle + time\_Running = Total CPU time : <" << CPU\_C.getSum\_total\_Run\_Idle\_time(CPU\_vector) << ">" << " time\_units";

cout << "\nProcessing time to Idle time Ratio --> ProcessTime:Idle time --> PT/IT = " //use this information to determine the growth or decay of this ratio --> we can compare ratio size at certain job completetion levels for diff num of CPUs

<< CPU\_C.getSum\_totalTimeProcessing\_CPUs(CPU\_vector) << "/" << CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector) << " = ";

if (CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector) != 0) {

cout << (double)(CPU\_C.getSum\_totalTimeProcessing\_CPUs(CPU\_vector)) / (double)(CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector)); //use type casting since int/int will not give you remainder or any ratio less than 1

}

else { cout << "~undetermined~ --> divide by 0"; } //catch divide by 0 case

cout << "\n+++++++++++++++++++++---------------------------------++++++++++++++++++++++++";

cout << "\n------------------------------------------------------------------------------";

system("pause");

}

//LOG FILE OUTPUT INFORMATION

LogFileStream << "\nClockManagement Cycle Time " << counter\_ClockManager\_TimeUnit << ": ";

for (counter\_logFile = 0; counter\_logFile < size\_of\_CPU\_vec; counter\_logFile++) {

LogFileStream << endl;

LogFileStream << "CPU " << counter\_logFile << ":: Queue: " << CPU\_vector.at(counter\_logFile).get\_CPU\_JobQueue\_size()

<< "; Idle Time: " << CPU\_vector.at(counter\_logFile).get\_IDLETIME()

<< "; Processing Time: " << CPU\_vector.at(counter\_logFile).get\_RUNTIME()

<< "; Current Job >Type: " << CPU\_vector.at(counter\_logFile).get\_CurrJob\_Processing().jobType

<< "; >ArrivalTime: " << CPU\_vector.at(counter\_logFile).get\_CurrJob\_Processing().arrivalTime

<< "; >RemainingProcessTime: " << CPU\_vector.at(counter\_logFile).get\_CurrJob\_Processing().processingTime;

}

//LOG FILE OUTPUT METRICS TO LOG FILE

if (

(CPUtotalTime\_Idle\_Running > 495 && CPUtotalTime\_Idle\_Running < 505)

|| (CPUtotalTime\_Idle\_Running > 9995 && CPUtotalTime\_Idle\_Running < 10005) //use these ranges to get at or close to 500 time units and 10,000 time units

|| (counter\_JobsCompleted\_Total == totalJobs\_inCurrLoadedDatFile) //output final table

)

{

LogFileStream << "\n------------------------------------------------------------------------------";

LogFileStream << "\nClockManagement--CYCLE NUMBER--: <" << counter\_ClockManager\_TimeUnit << ">";

LogFileStream << "\nNumber of processor(s) being used : <" << size\_of\_CPU\_vec << ">";

LogFileStream << "\nCurrent total queue size : <" << CPU\_C.getSum\_allJobsInQueue\_CPUs(CPU\_vector) << ">";

LogFileStream << "\nAverage CPU\_queue size : <" << CPU\_C.getCurrAverage\_CPUs\_queueSize(CPU\_vector) << ">";

LogFileStream << "\nMaximum jobs in one CPU\_queue : <" << CPU\_C.get\_MaxJobs\_inQueue(CPU\_vector) << ">";

LogFileStream << "\nAverage time in CPU\_queue : <" << CPU\_C.getTotalAverage\_TimeJobsInQueue\_CPUs(CPU\_vector) << ">" << " time\_units";

LogFileStream << "\nTotal time jobs in CPU\_queue : <" << CPU\_C.getSum\_totalTimeJobsInQueue\_CPUs(CPU\_vector) << ">" << " time\_units" << endl;

LogFileStream << "\nTotal number of jobs A arrived : <" << RFF.get\_TotalArrivals\_A() << ">";

LogFileStream << "\nTotal number of jobs B arrived : <" << RFF.get\_TotalArrivals\_B() << ">";

LogFileStream << "\nTotal number of jobs C arrived : <" << RFF.get\_TotalArrivals\_C() << ">";

LogFileStream << "\nTotal number of jobs arrived : <" << RFF.get\_totalArrivals() << ">" << endl;

LogFileStream << "\nTotal number of jobs A completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs\_A(CPU\_vector) << ">";

LogFileStream << "\nTotal number of jobs B completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs\_B(CPU\_vector) << ">";

LogFileStream << "\nTotal number of jobs C completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs\_C(CPU\_vector) << ">";

LogFileStream << "\nTotal jobs completed : <" << CPU\_C.getSum\_allJobsCompleted\_CPUs(CPU\_vector) << ">" << endl;

LogFileStream << "\nTotal time CPU(s) were processing : <" << CPU\_C.getSum\_totalTimeProcessing\_CPUs(CPU\_vector) << ">" << " time\_units";

LogFileStream << "\nTotal time CPU(s) were idle : <" << CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector) << ">" << " time\_units";

LogFileStream << "\nTotal time CPU(s) time\_Idle + time\_Running = Total CPU time : <" << CPU\_C.getSum\_total\_Run\_Idle\_time(CPU\_vector) << ">" << " time\_units";

LogFileStream << "\nProcessing time to Idle time Ratio --> ProcessTime:Idle time --> PT/IT = " //use this information to determine the growth or decay of this ratio --> we can compare ratio size at certain job completetion levels for diff num of CPUs

<< CPU\_C.getSum\_totalTimeProcessing\_CPUs(CPU\_vector) << "/" << CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector) << " = ";

if (CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector) != 0) {

LogFileStream << (double)(CPU\_C.getSum\_totalTimeProcessing\_CPUs(CPU\_vector)) / (double)(CPU\_C.getSum\_totalIdleTime\_CPUs(CPU\_vector)); //use type casting since int/int will not give you remainder or any ratio less than 1

}

else { LogFileStream << "~undetermined~ --> divide by 0"; } //catch divide by 0 case

LogFileStream << "\n+++++++++++++++++++++---------------------------------++++++++++++++++++++++++";

LogFileStream << "\n------------------------------------------------------------------------------";

}

}

LogFileStream.clear();

LogFileStream.close(); //clear and close the file stream when program finishes execution

cout << "\n\n~All jobs processed\_\_ExitingProgram\_\_Thank You...";

system("pause");

return 0;

}

/\* //this block is used for testing only\*\*\*

JobType\_ABC testJob;

ifstream testStream("JobDatabase.dat", ios::in | ios::binary);

if (!testStream.is\_open()) {

cout << "\nunable to open 'JobDatabase.dat' file.\n";

}

else { cout << "\n~Random Access File 'JobDatabase.dat' opened successfully~\n"; }

testStream.seekg(4149\*JobType\_ABC\_ByteSize, ios::beg);

testStream.read(reinterpret\_cast<char\*>(&testJob), JobType\_ABC\_ByteSize);

cout << endl << testJob.arrivalTime << endl;

cout << testStream.eof() << endl;

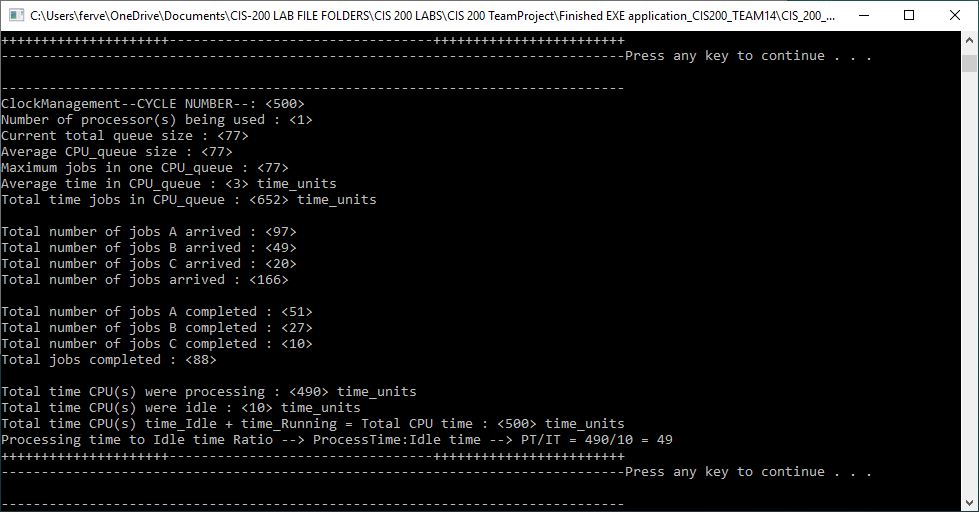
testStream.close();

\*/

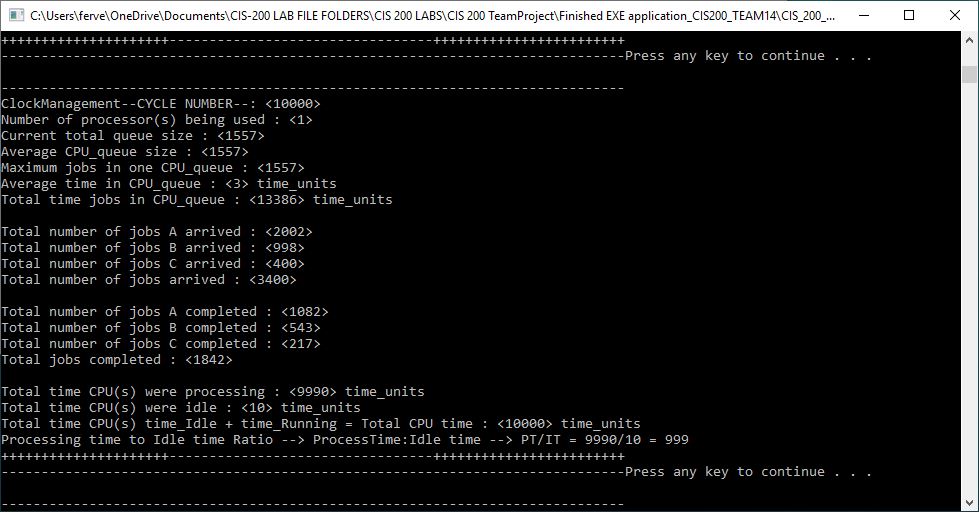
**Output snippets:**

**1 Processor:**

500 time units matrix:

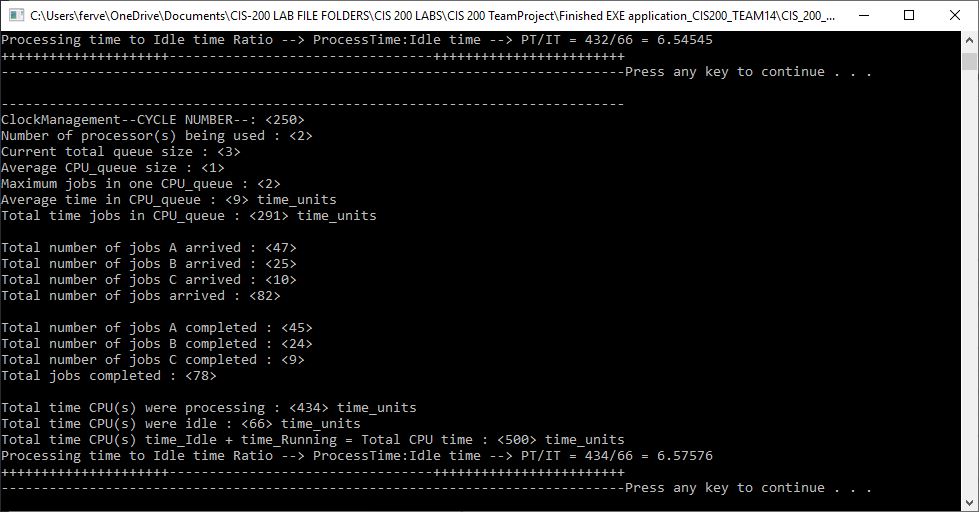


10000 time units matrix

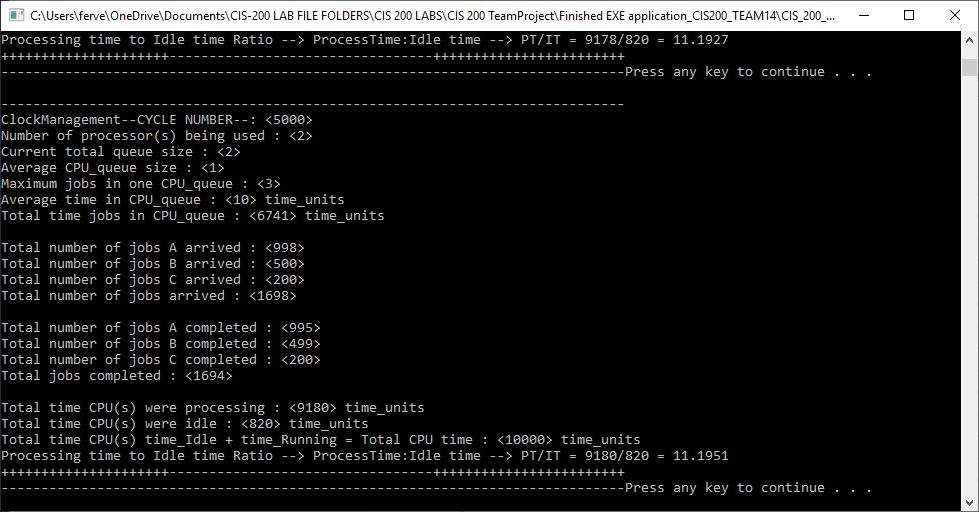


**2 Processors:**

500 time units matrix:

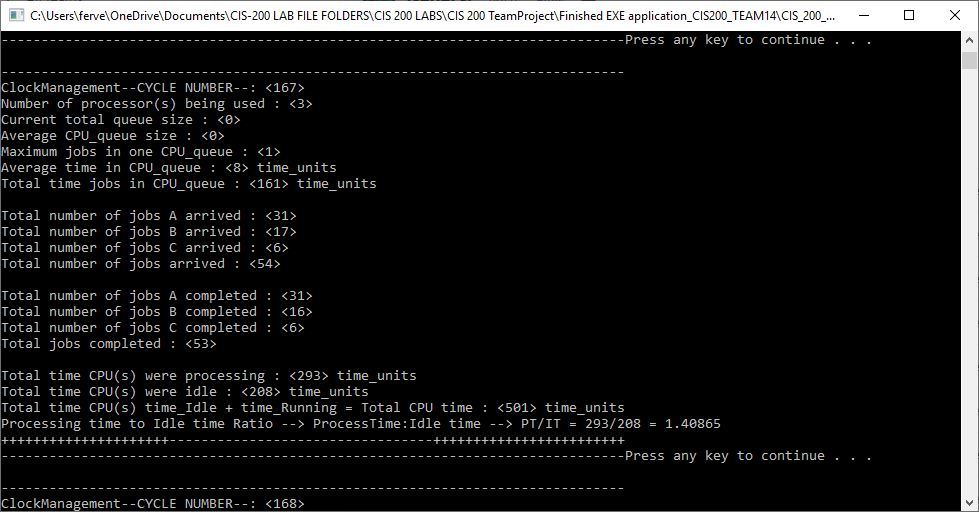


10000 time units matrix:

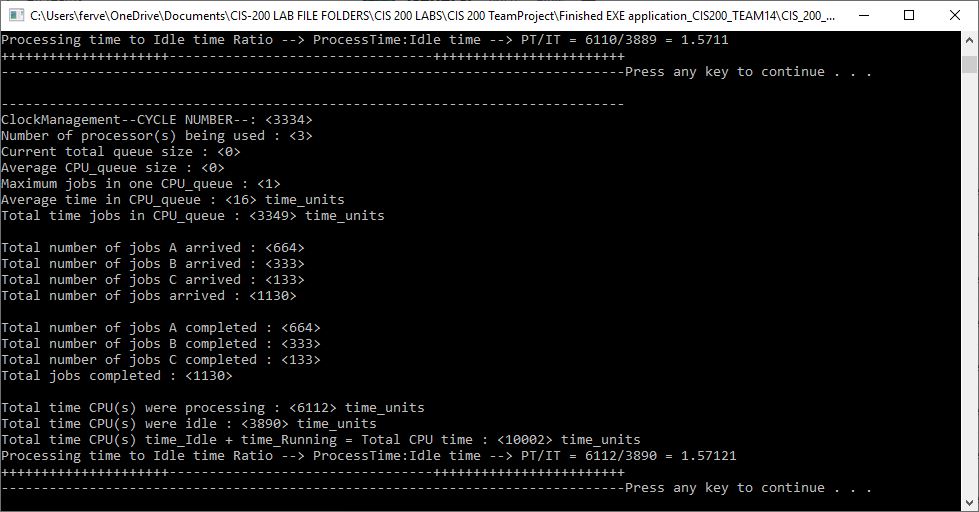


**3 Processors:**

500 time units matrix:

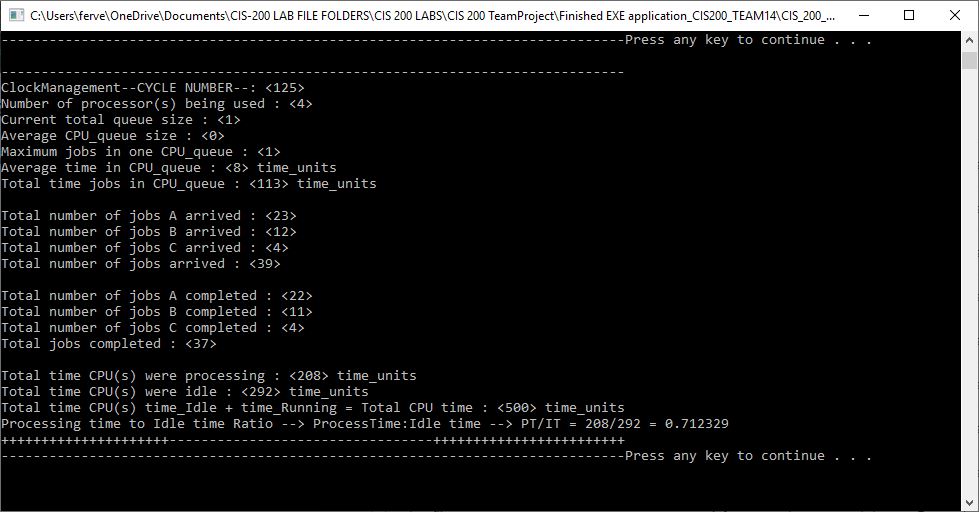


10000 time units matrix:

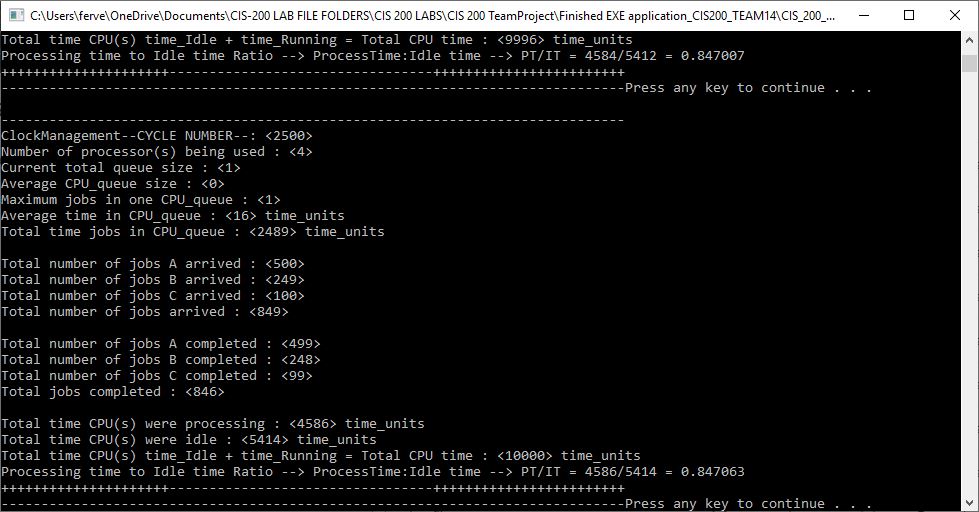


**4 Processors:**

500 time units matrix:

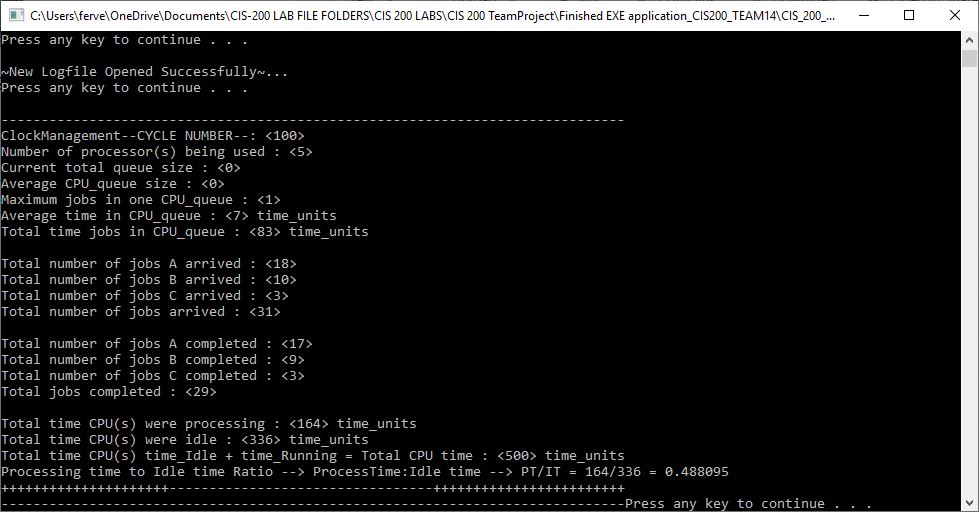


10000 time units matrix:

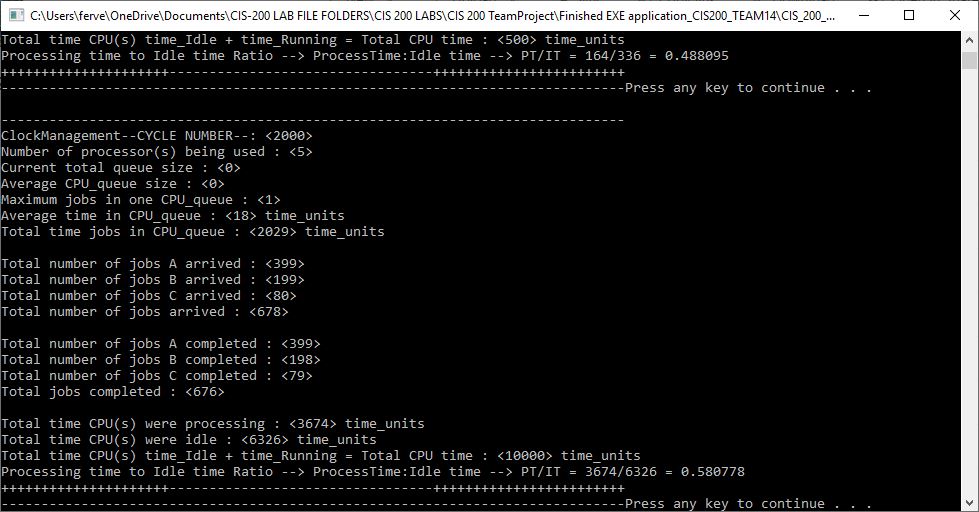


**5 Processors:**

500 time units matrix:

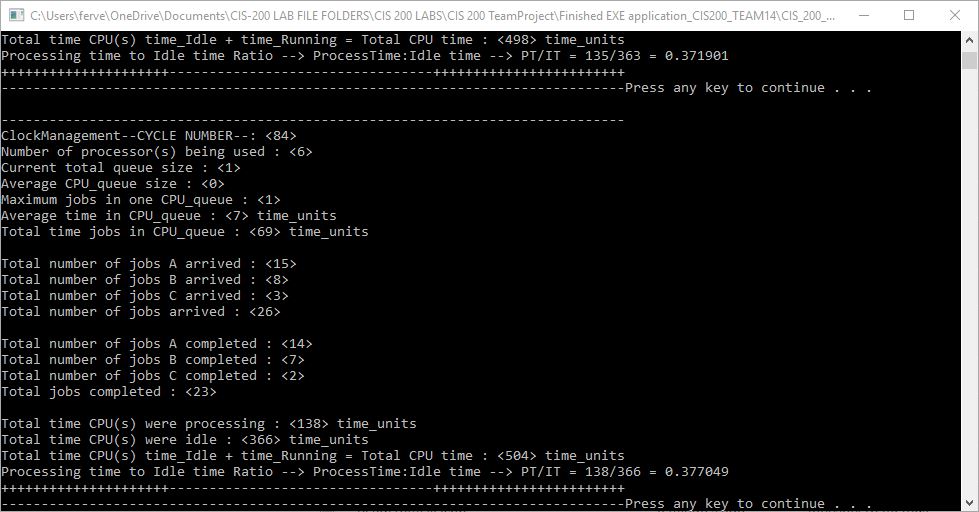


10000 time units matrix:

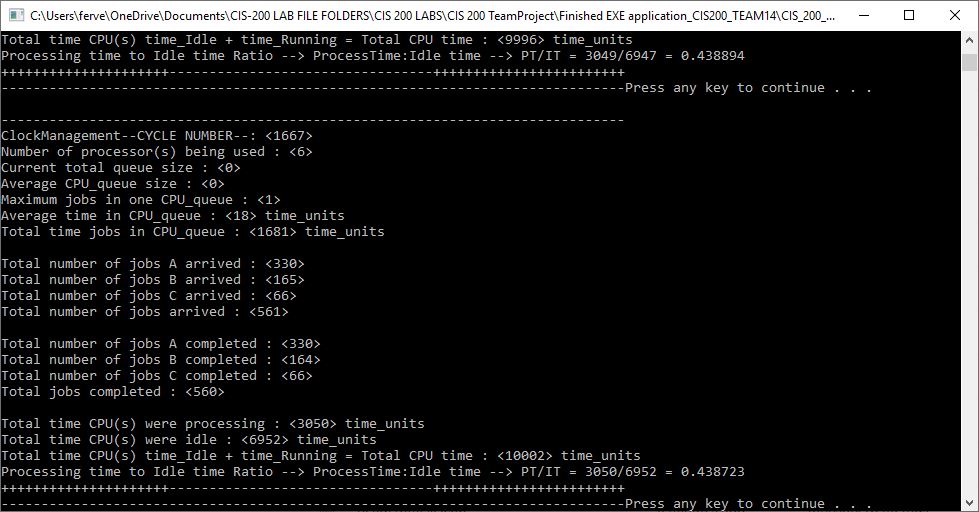


**6 Processors:**

500 time units matrix:

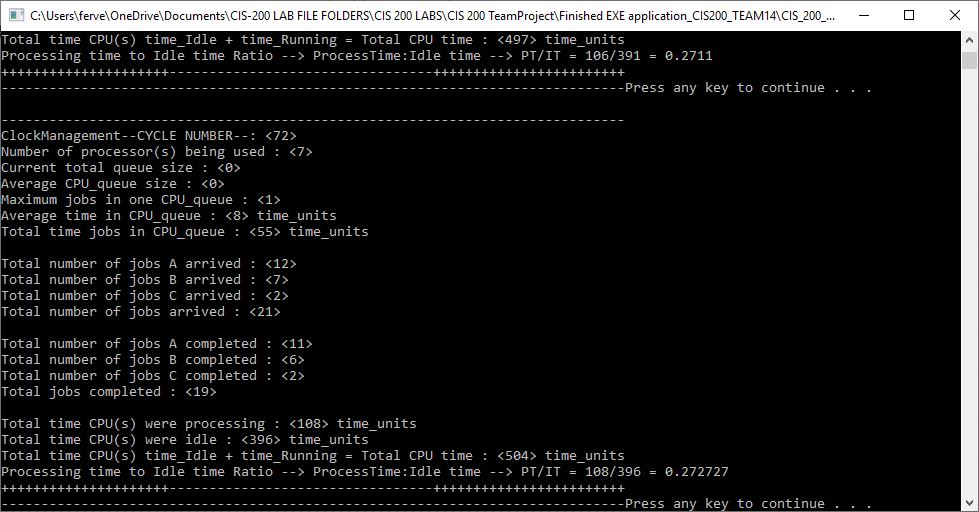


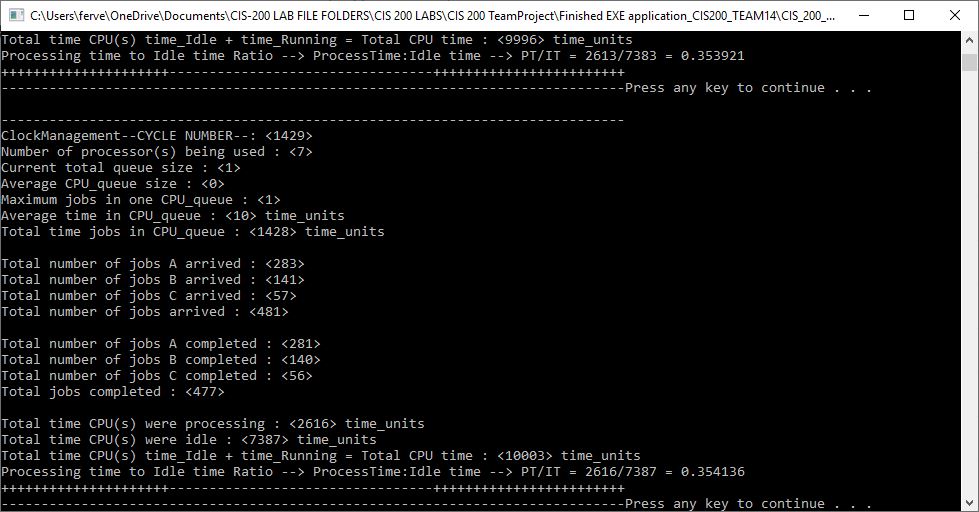
10000 time units matrix:



**7 Processors:**

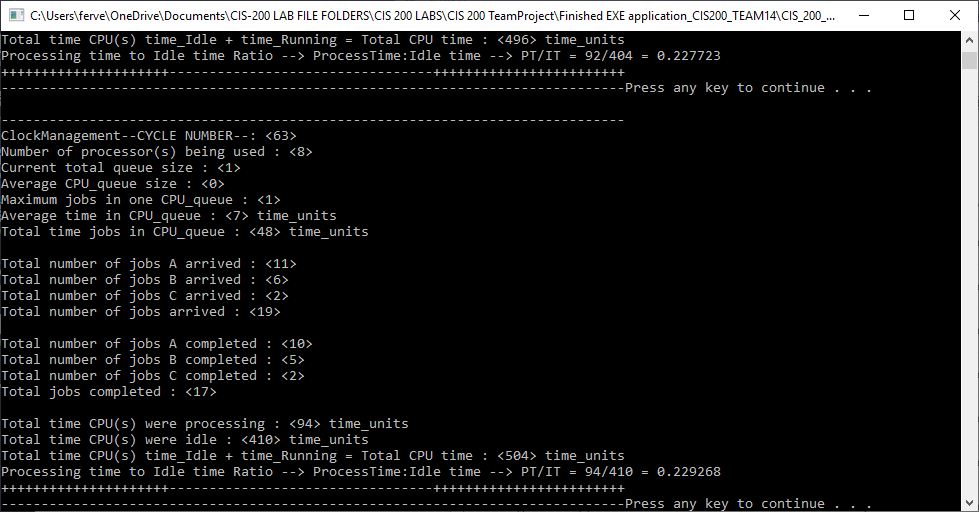
500 time units matrix:



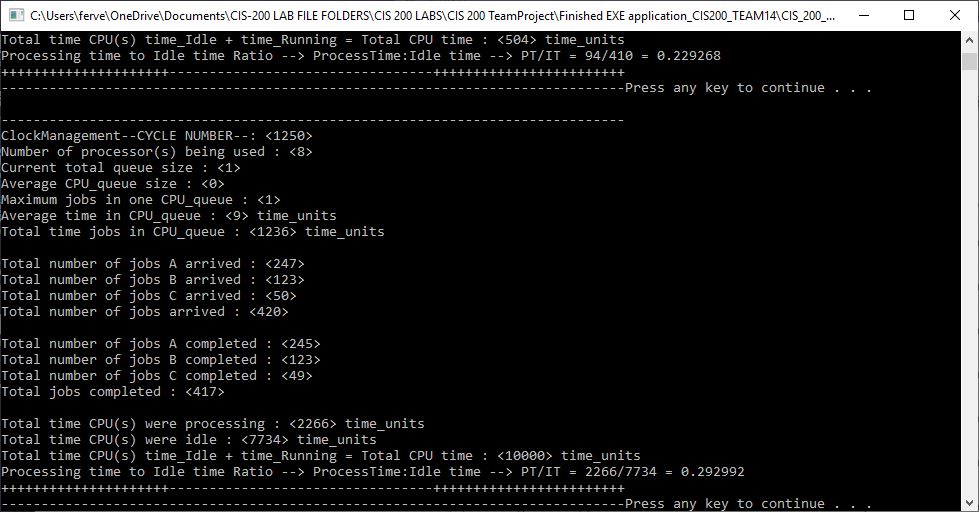
10000 time units matrix:

**8 Processors:**

500 time units matrix:



10000 time units matrix:



Log File (last 100 line for test run 4):

ClockManagement Cycle Time 12470:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 2; Current Job >Type: A; >ArrivalTime: 12468; >RemainingProcessTime: 2

CPU 1:: Queue: 0; Idle Time: 446; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1226; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12470; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12471:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 3; Current Job >Type: A; >ArrivalTime: 12468; >RemainingProcessTime: 1

CPU 1:: Queue: 0; Idle Time: 447; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1227; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12471; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12472:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 448; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1228; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12472; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12473:

CPU 0:: Queue: 0; Idle Time: 1; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 449; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1229; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12473; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12474:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 450; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1230; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12474; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12475:

CPU 0:: Queue: 0; Idle Time: 1; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 451; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1231; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12475; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12476:

CPU 0:: Queue: 0; Idle Time: 2; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 452; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1232; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12476; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12477:

CPU 0:: Queue: 0; Idle Time: 3; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 453; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1233; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12477; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12478:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 1; Current Job >Type: A; >ArrivalTime: 12477; >RemainingProcessTime: 1

CPU 1:: Queue: 0; Idle Time: 454; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1234; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12478; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12479:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 455; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1235; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12479; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12480:

CPU 0:: Queue: 0; Idle Time: 1; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 456; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1236; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12480; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12481:

CPU 0:: Queue: 0; Idle Time: 2; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 457; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1237; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12481; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12482:

CPU 0:: Queue: 0; Idle Time: 3; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 458; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1238; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12482; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12483:

CPU 0:: Queue: 0; Idle Time: 4; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 459; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1239; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12483; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12484:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 1; Current Job >Type: A; >ArrivalTime: 12483; >RemainingProcessTime: 3

CPU 1:: Queue: 0; Idle Time: 460; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1240; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12484; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12485:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 2; Current Job >Type: A; >ArrivalTime: 12483; >RemainingProcessTime: 2

CPU 1:: Queue: 0; Idle Time: 461; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1241; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12485; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12486:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 3; Current Job >Type: A; >ArrivalTime: 12483; >RemainingProcessTime: 1

CPU 1:: Queue: 0; Idle Time: 462; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1242; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12486; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12487:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 463; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1243; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12487; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12488:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 1; Current Job >Type: A; >ArrivalTime: 12487; >RemainingProcessTime: 3

CPU 1:: Queue: 0; Idle Time: 464; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1244; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12488; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12489:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 2; Current Job >Type: A; >ArrivalTime: 12487; >RemainingProcessTime: 2

CPU 1:: Queue: 0; Idle Time: 465; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1245; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12489; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12490:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 3; Current Job >Type: A; >ArrivalTime: 12487; >RemainingProcessTime: 1

CPU 1:: Queue: 0; Idle Time: 466; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1246; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12490; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12491:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 467; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1247; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12491; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12492:

CPU 0:: Queue: 0; Idle Time: 1; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 468; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1248; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12492; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12493:

CPU 0:: Queue: 0; Idle Time: 2; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 469; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1249; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12493; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12494:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 1; Current Job >Type: A; >ArrivalTime: 12493; >RemainingProcessTime: 1

CPU 1:: Queue: 0; Idle Time: 470; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1250; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12494; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

ClockManagement Cycle Time 12495:

CPU 0:: Queue: 0; Idle Time: 0; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 1:: Queue: 0; Idle Time: 471; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 2:: Queue: 0; Idle Time: 1251; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

CPU 3:: Queue: 0; Idle Time: 12495; Processing Time: 0; Current Job >Type: ; >ArrivalTime: 0; >RemainingProcessTime: 0

**Test runs and analysis:**

1 CPUs)

This run has the highest running time to idle time ratio, means the processor is the busiest

2 CPUs)

It’s more efficient to run compare to a single processor, which in result has a shorter time taken for it to finish processing the same amount of data

3 CPUs)

This time it’s even faster than 2 processors, however there’s more idle time for the CPU(s)

4 CPUs)

So again, faster processing speed, but more idle time, what’s worth mentioning is that the processing time/idle time ratio is less than 1 meaning there’s more idle time.

5 CPUs)

Really long idle time with a small increase on the processing speed

6 CPUs)

Really long idle time with a small increase on the processing speed

7 CPUs)

Really long idle time with a small increase on the processing speed

8 CPUs)

Really long idle time with a small increase on the processing speed

Conclusion:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| busiest |  |  | Most efficient overall |  |  |  | Fastest, more efficient in long run?? |

The more processors it has, the faster it will process, however, there’s more unutilized processing power, which will make it less efficient in a way.

Special note: the reason why 4 is most efficient is possibly attributed to a factor concerning processing time in the short run and the long run; in short term, 4CPUs at 500 units had about 2.17 times the amount of jobs completed as did the 8 CPUs; however as time went on, at 10,000 time units the gap began to shrink from 2.17 times to 2.02.

The question remains: To what extent did the data file affect performance, the number of small jobs (type A, then type B) with shorter processing times was significantly larger than the total amount of higher process time jobs (type C); thus it becomes inefficient to use so many processors.

In the event where a data file had more type C jobs, or shorter arrival time between shops, this will have likely had a great impact on performance versus CPU count; ie, 8CPUs would likely perform better especially in the long run if a data file contained lots of large jobs with high processing times or with a data file with very short arrival time between jobs (or even both).