

**Machine Job Scheduling**

**Software Requirement Specification Document**

**Project Timeline: 2/11/2022 to 9/11/2022**

Team Members

M. Kavyasri

Mamatha

V.Varshitha

N.Ramya

Sk. Neha Sulthana

**Table of Contents**

**1. Introduction 3**

1.1 Purpose

1.2 Intended Audience and Reading Suggestions

1.3 Scope of Application

**2. Overall Description 4**

2.1 Application Perspective

2.2 Application Features

2.3 Design and Implementation Constraints

2.4 Assumptions and Dependencies

**3. System Features 5-6**

3.1 Functional Requirements

**4. Technical Requirements 7**

**5. Non Functional Requirements 8**

**6. UML Designs 9- 10**

**7. Creating Makefile 11**

**8. CPP Files 12 - 19**

**9. Conclusion 20**

**Introduction**

**1.1 Purpose**

Machine job scheduling application is to be developed which will process jobs data and assign requested machines to the jobs. Schedule for each machine is generated. Jobs are of different types. Most of the information on jobs is common; some of the information is specific to each job type. This document provides an abstract overview of the Simulation of Scheduler and provides a general overview of the entire application.

**1.2 Intended Audience and Reading Suggestions**

This document explains our team architecture, our team’s initial understanding of the user needs. It will assist our team in understanding the system specifications and analyze the critical aspects of our project. This document will briefly discuss the stakeholders involved in the development, documents will show how our team was divided to handle the multiple stakeholders, the sources of the requirements, provide an informal preliminary requirements description, and address any issues encountered while transforming the requirements.

**1.3 Scope of Application**

Scheduling without interference and making all jobs done for given machines without leaving any job given in list of job file. Limitations will be some interference if two jobs at a time need same machine it need to be wait state. Required to involve techniques such as concurrency, inheritance to make the project done completely.

**2. Overall Description**

**2.1 Application Perspective**

CPP ATL enables to code the job specifications ,FileIO operations helps to read file from Command line arguments and write schedule into different text files .Valgrind captures the data of memory leak.

**2.2 Application Features**

Scheduling without interference and making all jobs done for given machines without leaving any job given in list of job file. Limitations will be some interference if two jobs at a time need same machine it need to be wait state.

**2.3 Design and Implementation Constraints**

The files that are sent through command line argument are checked for exception and catches exception if it occurs and at the conditions where the source code may terminate gets checked for exceptions by placing that code in try block if any exceptions it cached by catch block.

**2.4 Assumptions and Dependencies**

Schedule for different machines is done by assuming the job given in the job file is of correct format. Modified application will remove the jobs that are not in correct format All assumptions, functional overview and design parameters are documented without evaluation which are to be implemented without missing. It is expected to give a correct schedule for each machine, for each job given and accuracy of code under different circumstances occur while assigning different jobs

**3. System Features**

**3.1 Functional Requirements**

Machine job scheduling application is to be developed which will process jobs data and assign requested machines to the jobs. Schedule for each machine is generated. Jobs are of different types. Most of the information on jobs is common, some of the information is specific to each job type.

* Information on jobs is available in multiple files, these filenames will be passed to application as command line arguments.
* Format of Job description files -

Job No:Description:Machine No:Duration(in minutes):Client nam:Job Type:special attributes (separated by comma, as specified in MJ03)

Example job entries -

J001:Pipe cutting:M1:45:Expert Accessories:Cut:10,100

* 3 machines are resources to be shared among jobs. All machines start at time T1. Whenever a machine is allocated to a job its time will advanced by “Duration” and an entry will be written in “Schedule".
* Schedule entries format -
* Job No:Start time:End time
* All jobs files should be processed as specified in requirement
* At the end, schedule of each machines to be displayed and stored in schedule1.txt, schedule2.txt and schedule3.txt files.

**4. Technical Requirements**

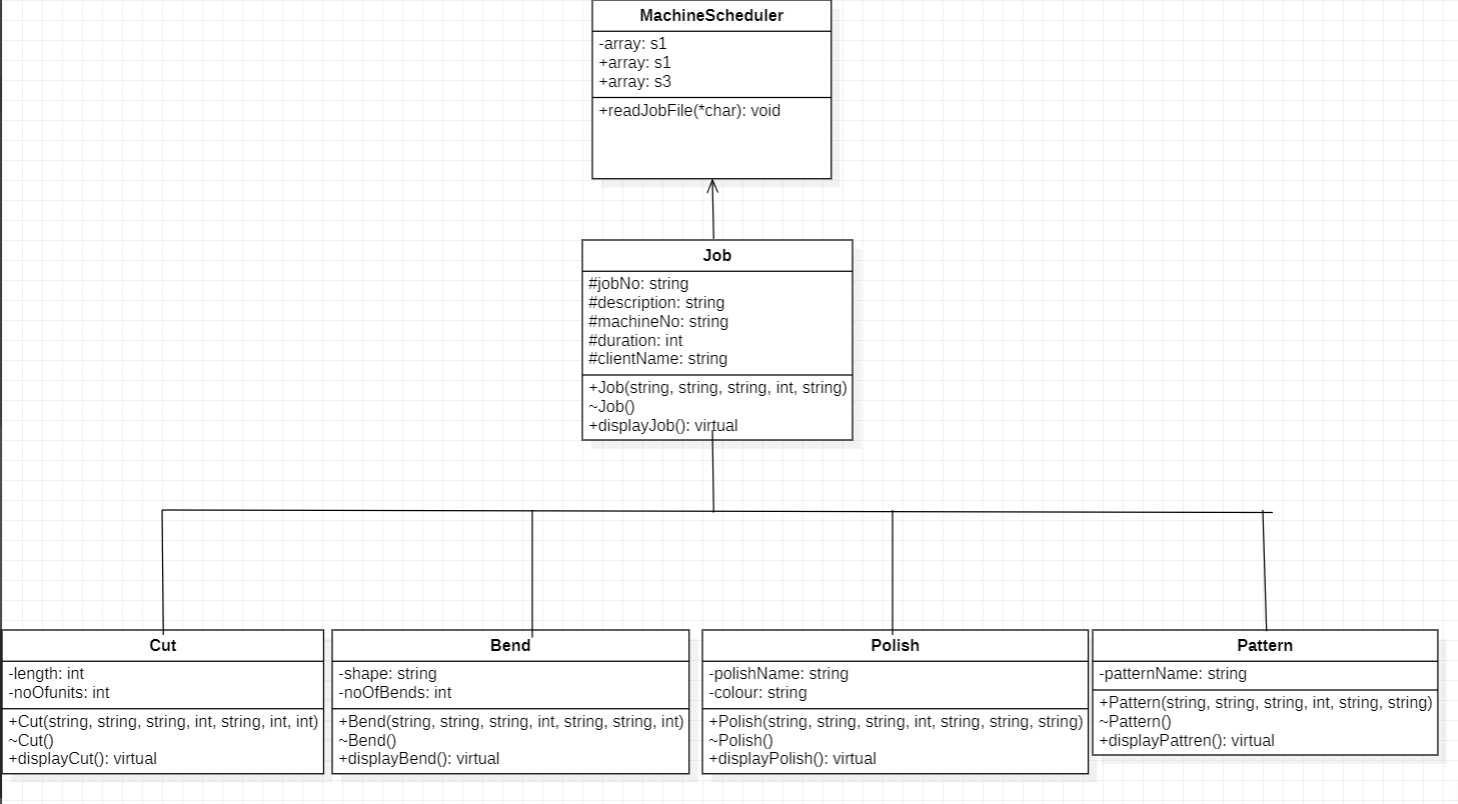
|  |
| --- |
| * Design the system using OOPs. Use C++ programming language to implement. |
| * Use multi-threading. Use POSIX-thread library. Access to shared resources to be synchronized. Use 3 mutexes for 3 machines. Create threads to process multiple input files. |
| * Use file input/output operations to read jobs data and write schedules. |
| * Use suitable STL to read jobs data and to store schedule entries. |
| * Use exception handling to validate data and handle exceptions. |
| * Use inheritance to store jobs and run-time polymorphism using virtual functions. |
| * Use dynamic memory allocation. |

**5. Non Functional Requirements**

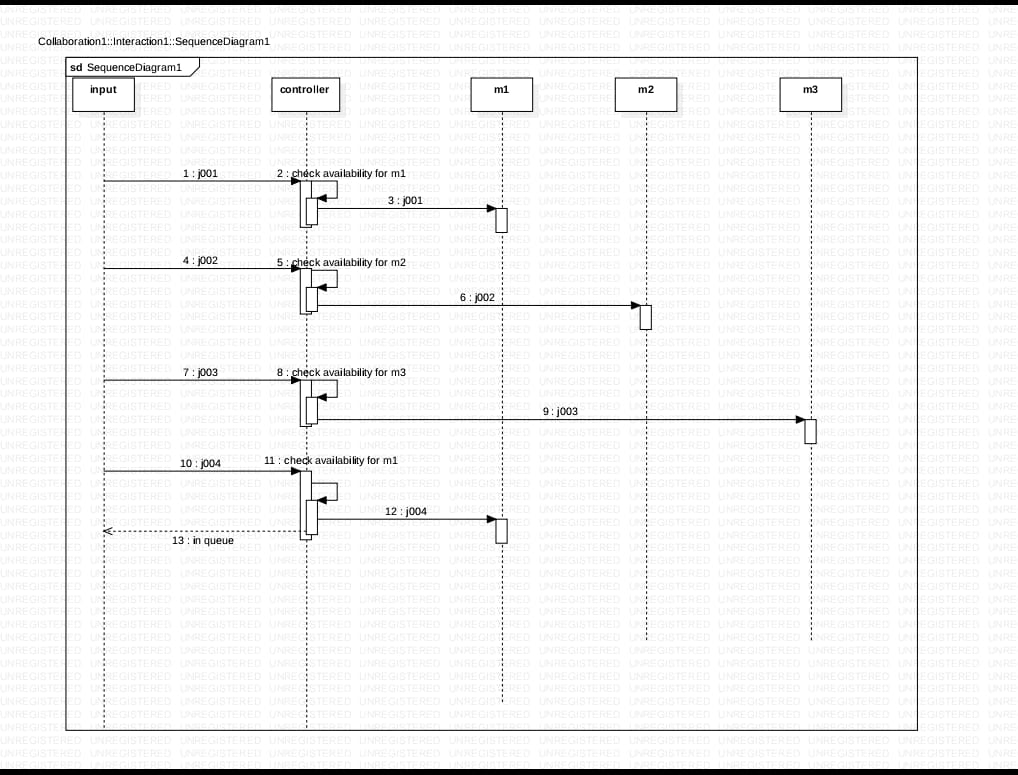
|  |
| --- |
| * Multi-file multi-directory solution is expected. Modular and maintainable code (comments) and all coding standards should be followed. |
| * makefile to build application. Two-step compilation process - .o and then executable should be generated. .o files to be stored under “obj” sub-directory of project directory. |
| * Use valgrind tool on application executable to detect memory leak. Final valgrind report to be submitted in “tools/valgrind” directory. |
| * Class diagram, sequence diagram, usecase diagrams. |
| * Use CPPunit to automate unit testing. At least 1 or 2 testcases using CPPunit. Other testcases can be tested manually. CPPUnit code to be submitted under “tools/CPPUnit” directory. |
| * HLD, LLD of the system |
| * SRS(in docx or pdf format), RTM, Plan, Presentation |
| * Unit test cases (at least 10 test cases per module) and Integration test cases in UT\_IT document. Both types of test cases i.e. sunny and rainy should be present in this document |

**6. UML Designs**

**Class Diagram:**

****

**Sequence Diagram**

****

**7. Creating Makefile**

|  |
| --- |
| SRCDIR= src |
|  | INCLUDEDIR = include |
|  | OBJDIR = obj |
|  | BINDIR = bin |
|  | VALGRIND = tools/valgrind.txt |
|  |  |
|  | INCLUDEFLAGS = -I${INCLUDEDIR} |
|  | CPPFLAGS = -ansi -c -g -Wall -DDEBUG ${INCLUDEFLAGS} |
|  |  |
|  | final: job.o cut.o bend.o polish.o pattern.o MachineScheduler.o main.o |
|  | g++ -o ${BINDIR}/final ${OBJDIR}/job.o ${OBJDIR}/cut.o ${OBJDIR}/bend.o ${OBJDIR}/polish.o ${OBJDIR}/pattern.o ${OBJDIR}/MachineScheduler.o ${OBJDIR}/main.o |
|  | main.o : ${SRCDIR}/main.cpp |
|  | g++ -o ${OBJDIR}/main.o ${CPPFLAGS} ${SRCDIR}/main.cpp |
|  |  |
|  | MachineScheduler.o : ${SRCDIR}/machinescheduler.cpp |
|  | g++ -o ${OBJDIR}/MachineScheduler.o ${CPPFLAGS} ${SRCDIR}/machinescheduler.cpp |
|  |  |
|  | pattern.o : ${SRCDIR}/pattern.cpp |
|  |  |
|  | g++ -o ${OBJDIR}/pattern.o ${CPPFLAGS} ${SRCDIR}/pattern.cpp |
|  | polish.o : ${SRCDIR}/polish.cpp |
|  | g++ -o ${OBJDIR}/polish.o ${CPPFLAGS} ${SRCDIR}/polish.cpp |
|  |  |
|  | bend.o : ${SRCDIR}/bend.cpp |
|  | g++ -o ${OBJDIR}/bend.o ${CPPFLAGS} ${SRCDIR}/bend.cpp |
|  |  |
|  | cut.o : ${SRCDIR}/cut.cpp |
|  | g++ -o ${OBJDIR}/cut.o ${CPPFLAGS} ${SRCDIR}/cut.cpp |
|  |  |
|  | job.o : ${SRCDIR}/job.cpp |
|  | g++ -o ${OBJDIR}/job.o ${CPPFLAGS} ${SRCDIR}/job.cpp |
|  |  |
|  | valgrind: $(BIN)/final |
|  | valgrind --leak-check=full --show-reachable=yes $(BIN)/final 2>$(VALGRIND)/valgrind.txt |
|  | clean : |
|  | echo "cleaning up .o and exes" |
|  | rm ${BINDIR}/final ${OBJDIR}/main.o ${OBJDIR}/job.o |

**CPP Files**

**1.job.cpp**

|  |
| --- |
|  |
|  | #include<iostream>  #include<vector> |
|  | #include<algorithm> |
|  | #include<job.h> |
|  |  |
|  | //vector declaration |
|  | void Job::show(vector<Job> jobs){ |
|  |  |
|  | vector<Job>::iterator it; |
|  | for(it=jobs.begin();it!=jobs.end();++it) |
|  | { |
|  | cout<<"JobId"<<"Machine No"<<"Duration"<<endl; |
|  | cout<<it->jobId<<" "<<it->machineNo<<" "<<it->duration<<endl; |
|  | } |
|  |  |
|  | } |
|  | //parameterized constructor |
|  | Job::Job(string jobId,string description,string machineNo,int duration1,string client){ |
|  | this->jobId=jobId; |
|  | this->description=description; |
|  | this->machineNo=machineNo; |
|  | this->duration=duration1; |
|  | this->client=client; |
|  | } |
|  |  |
|  | string Job::getJobId(){return jobId;} |
|  | string Job::getDescription(){return description;} |
|  | string Job::getMachineNo(){return machineNo;} |
|  | int Job::getDuration(){return duration;} |
|  | string Job::getClient(){return client;} |
|  |  |
|  | //to display job details |
|  | void Job::displayJob () const{ |
|  | cout<<jobId<<","<<description<<","<<machineNo<<","<<duration<<","<<client<<endl; |
|  | }; |
|  |  |
|  |  |

**2.cut.cpp**

|  |  |
| --- | --- |
|  | #include<iostream>  #include<cut.h> |
|  | using namespace std; |
|  |  |
|  | //parameterized constructor |
|  |  |
|  | Cut::Cut(string jobId,string description,string machineNo,int duration,string client,int len,int units){ |
|  | this->length=len; |
|  | this->noOfUnits=units; |
|  | } |
|  | int Cut::getLength(){return length;} |
|  | int Cut::getNoOfUnits(){return noOfUnits;} |
|  | //displays extended data including base data |
|  | void Cut:: displayJob() const{ |
|  | Job::displayJob(); |
|  | cout<<","<<length<<","<<noOfUnits<<endl; |
|  | } |

**3.pattern.cpp**

|  |
| --- |
|  |
|  |
|  | #include<iostream>  #include<pattern.h> |
|  | using namespace std; |
|  |  |
|  | Pattern::Pattern(string jobId,string description,string machineNo,int duration,string client,string patternName){ |
|  | this->patternName=patternName; |
|  |  |
|  | } |
|  | string Pattern::getPatternName(){return patternName;} |
|  |  |
|  | void Pattern::displayJob() const{ |
|  | Job::displayJob(); |
|  | cout<<","<<patternName<<endl; |
|  | } |
|  |  |

**4.Bend.cpp**

|  |  |
| --- | --- |
|  | #include<iostream>  #include <bend.h> |
|  | using namespace std; |
|  |  |
|  |  |
|  | Bend::Bend(string jobId,string description,string machineNo,int duration,string client,string shape,string noOfBends){ |
|  | this->shape=shape; |
|  | this->noOfBends=noOfBends; |
|  | } |
|  | string Bend::getShape(){return shape;} |
|  | string Bend::getNoOfBends(){return noOfBends;} |
|  |  |
|  | void Bend::displayJob() const{ |
|  | Job::displayJob(); |
|  | cout<<","<<shape<<","<<noOfBends<<endl; |
|  | } |

**5. polish.cpp**

|  |
| --- |
|  |
|  | #include<iostream>  #include<polish.h> |
|  | using namespace std; |
|  |  |
|  | Polish::Polish(string jobId,string description,string machineNo,int duration,string client,string polishName1,string colour1){ |
|  | this->polishName=polishName1; |
|  | this->colour=colour1; |
|  | } |
|  | string Polish::getPolishName(){return polishName;} |
|  | string Polish::getColour(){return colour;} |
|  |  |
|  | void Polish:: displayJob() const{ |
|  | Job::displayJob(); |
|  | cout<<","<<polishName<<","<<colour<<endl; |
|  | } |
|  |  |

**6. machinescheduler.cpp**

|  |
| --- |
|  |
|  | #include<iostream> |
|  | #include<fstream> |
|  | #include<istream> |
|  | #include<ostream> |
|  | #include<vector> |
|  | #include<iterator> |
|  | #include <sstream> |
|  | #include <algorithm> |
|  | #include<array> |
|  | #include<string> |
|  | #include<mutex> |
|  | #include<machinescheduler.h> |
|  | #include<job.h> |
|  | #include<cut.h> |
|  | #include<bend.h> |
|  | #include<polish.h> |
|  | #include<pattern.h> |
|  | using namespace std; |
|  | //declaring mutex |
|  | mutex m1; |
|  | mutex m2; |
|  | mutex m3; |
|  |  |
|  |  |
|  | int MachineScheduler::readInputFile(const char\* inpFile) |
|  | { |
|  | int i=0; |
|  | ifstream input;//opening file to be readed |
|  | input.open(inpFile); |
|  | //try block to catch ecxeption |
|  | try{ |
|  | if (!input.is\_open()) |
|  | { |
|  | cout << "Unable to open input file " << inpFile << endl; |
|  | return 1; |
|  | } |
|  | }catch(...){ |
|  | cout << "Unable to open input file "; |
|  | } |
|  |  |
|  |  |
|  | string line; |
|  | while(getline(input,line)) |
|  | { |
|  | //declaring instance variables of all classes |
|  | string line; |
|  | stringstream ss; |
|  | string jobId; |
|  | string description; |
|  | int duration; |
|  | string machineNo; |
|  | string client; |
|  | int length; |
|  | int noOfUnits; |
|  | string shape; |
|  | string noOfBends; |
|  | string polishName; |
|  | string colour; |
|  | string patternName; |
|  | int c=0,c1=0,c2=0,c3=0,c4=0; |
|  |  |
|  | //parsing jobdata from inpFile |
|  | if(getline(ss,jobId,':')) |
|  | c++; |
|  | if(getline(ss,description,':')) |
|  | c++; |
|  | if(getline(ss,machineNo,':')) |
|  | c++; |
|  | ss>>duration; |
|  | duration=atof(duration); |
|  | if(getline(ss,duration,":")); |
|  | c++; |
|  | if(getline(ss,client,':')) |
|  | c++; |
|  | c1=c;c2=c;c3=c;c4=c; |
|  |  |
|  | ss>>len; |
|  | length=atof(len); |
|  | if(getline(ss,length,":")); |
|  | c1++; |
|  | ss>>units; |
|  | noOfUnits=atof(units); |
|  | if(getline(ss,noOfUnits,":")); |
|  | c1++; |
|  |  |
|  | if(getline(ss,shape,":")); |
|  | c2++; |
|  | if(getline(ss,noOfBends,":")); |
|  | c2++; |
|  |  |
|  | if(getline(ss,polishName,":")); |
|  | c3++; |
|  | if(getline(ss,colour,":")); |
|  | c3++; |
|  |  |
|  | if(getline(ss,patternName,":")); |
|  | c4++; |
|  |  |
|  |  |
|  | if(c1==7 && jobId!=" " && machineNo=="M1" && duration!=0){ |
|  |  |
|  | s1[i]=Schedule1(jobId:description:machineNo:duration:client,length,noOfUnits); |
|  | i++; |
|  | } |
|  | else if(c2==7 && jobId!=" " && machineNo=="M2" && duration!=0) |
|  | { |
|  | s2[i]=Schedule2(jobId:description:machineNo:duration:client:shape:noOfBends); |
|  | i++; |
|  |  |
|  | } |
|  | else if(c3==7 && jobId!=" " && machineNo=="M2" && duration!=0) |
|  | { |
|  | s2[i]=Schedule2(jobId:description:machineNo:duration:client:poliahName:colour); |
|  | i++; |
|  | } |
|  | else if(c4==6 && jobId!=" " && machineNo=="M3" && duration!=0) |
|  | { |
|  | s3[i]=Schedule3(jobId:description:machineNo:duration:client:patternName); |
|  | i++; |
|  | } |
|  | else |
|  | cout<<"Invalide job type"<<endl; |
|  | } |
|  |  |
|  | input.close(); |
|  |  |
|  | return 0; |
|  | } |
|  | //Thread class |
|  | void Thread::thread1(){ |
|  | //opening file to be writed |
|  | ofstream file1; |
|  | file1.open("schedule1.txt"); |
|  | try{ |
|  | if (!file1.is\_open()) |
|  | cout<<"invalid file"; |
|  | }catch(...){ |
|  | cout<<"invalid file"; |
|  | } |
|  | m1.lock(); |
|  | for(int i=0;i<=size(s1);i++) |
|  | { |
|  | cout<<s1<<endl; |
|  | file1<<jobId<<" "<<d1<<" "<<d1+duration; |
|  | } |
|  | m1.unlock(); |
|  | } |
|  |  |
|  | void Thread::thread2(){ |
|  | ofstream file2; |
|  | file2.open("schedule2.txt"); |
|  | try{ |
|  | if (!file2.is\_open()) |
|  | cout<<"invalid file"; |
|  | }catch(...){ |
|  | cout<<"invalid file"; |
|  | } |
|  | m2.lock(); |
|  | for(int i=0;i<=size(s2);i++) |
|  | { |
|  | cout<<s2<<endl; |
|  | file2<<jobId<<" "<<d2<<" "<<d2+duration; |
|  | } |
|  | m2.unlock(); |
|  | } |
|  | void Thread::thread3(){ |
|  | ofstream file3; |
|  | file3.open("schedule3.txt"); |
|  | try{ |
|  | if (!file3.is\_open()) |
|  | cout<<"invalid file"; |
|  | }catch(...){ |
|  | cout<<"invalid file"; |
|  | } |
|  |  |
|  | m3.lock(); |
|  | for(int i=0;i<=size(s3);i++) |
|  | { |
|  | cout<<s3<<endl; |
|  | file3<<jobId<<" "<<d3<<d3+duration; |
|  | } |
|  | m3.unlock(); |
|  | } |
|  | file1.close(); |
|  | file2.close(); |
|  | file3.close(); |
|  | return 0; |
|  | } |

**9.Conclusion**

We can implement schedule for more than 3 machines, by using maps, vectors, arrays and strings. The fact that constraint programming approaches are so effective is not only relevant for the dedicated purpose, but for industry as well.

In fact, the need for easy adaptability and cheap maintenance in the long run or easily met by such approaches