Programming Languages
Homework 8 & 9

141OS is a very simple operating system that allows multiple users to save and print files.

NOTE: Read this entire assignment – AT LEAST TWICE – before you even think about writing any code. It is worth 200 points (or two homework assignments). Also see this important note about submission for both HW8 and HW9 LINK.

The 141OS manages multiple disks, multiple printers, and multiple users – all of which can be considered concurrent processes. The goal of the system is to exploit possible parallelism to keep the devices (disks and printers) as busy as possible.

This will be a two week assignment where the following is needed:

- The first part is to define the Java classes for the **backend** portion.
- The second part is to write the **GUI portion** using JavaFX or Swing.

We will run all your code through MOSS which will detect any sharing of code (either between classmates or by copying any code found on the Internet). **Anyone whose code shows up as a copy will (at least) get a 0 for this entire homework assignment.**

For this homework, we would like everyone to have the following project structure before you zip:

```
src/       # Has all your .java files
inputs/    # Has your USER* input files for the program
resources/ # Has your GUI resources (e.g. Disk.png, etc.)
Makefile   # Has scripts that will compile builds and runs
```

For the Makefile, make sure it has something akin to this for building:

```
build: <all your .java files>
     @echo ----- creating a jar file with java -----
     jar cf 141OS.jar <all your .java files>
```

This creates a jar file, which will only be created if your files compile successfully. From here, we can run your program like this:

```
$ java -jar 141OS.jar <args>
```

Please conform to this so the autograder can work as intended. Be sure to handle the command line arguments described below LINK.

The autograder will be using openjdk version 8, so keep that in mind while building your project.

Homework 8

Due Wednesday, March 4th at 2 AM

[100] DESIGN SKETCH: PART ONE

You will be defining the classes that work as the backend of the 141OS. For this assignment, here are some important requirements regarding the operating system:

- Files can be stored on any disk and printed on any printer by any user.
- Only one line of a file may be stored in a sector, so a file will require one sector per line.
- Although Printers and Disks can be thought of as parallel processes, I suggest you not make them Threads. If you do, you must make them *Daemon threads* by calling setDaemon or your program will never terminate.
- The classes with the name "Thread" in them, e.g., UserThread & PrintJobThread, must be running on their own Thread.
- Use the StringBuffer, which is implemented as an array of char. StringBuffer is the standard unit stored on disks, sent to printers *one line at a time*, and handled by users *one line at a time*. We are using StringBuffers because Strings are not modifiable. Note also that StringBuffer.equals must do a deep equality if you plan to do comparisons.
- You can only have one user writing to a disk at a time or the files will be jumbled. You can have any number of Threads reading at a time – even when someone is writing to the disk. This is not normally the case in a real OS; I had to simplify the problem here.

NOTE: If you use a BufferedWriter, be sure you flush the buffer after each write.

Your program **must** be runnable from command line and take command line arguments to define the number of users, disks, and printers. You should expect to follow this format:

```
$ java -jar 141OS.jar -#ofUsers <Users> -#ofDisks -#ofPrinters
```

An example input would be

```
$ java -jar 141OS.jar -3 USER1 USER2 USER3 -2 -3
```

This specifies that 3 users, 2 Disks, and 3 Printers are defined, with User1, User2, and User3 being files that mimic user input. Store instances of the appropriate objects in three separate arrays: Users, Printers, and Disks. Refer to this for how to pass arguments to your program.

A sample of the input (USER*) and output (PRINTER*) is in the hw8 directory located in my Homework 8 directory here. The PRINTER* files will be generated by your program, but the exact files that get sent to each printer may vary.

Below are the classes that you should define for the 141OS:

Disk

Each disk has a capacity specified to the constructor. The constructor must allocate all the StringBuffers (one per sector) when the Disk is created and must not allocate any after that. You can only read or write one line at a time, so to store a file, you must issue a write for each input line. You may implement it as an array of StringBuffers. Reads and writes each take **200 milliseconds**, so your read/write functions must sleep for that amount of time before copying the data to/from any disk sector.

```
class Disk {
     static final int NUM_SECTORS = 1024;
     StringBuffer sectors[] = new StringBuffer[NUM_SECTORS];
     void write(int sector, StringBuffer data);  // call sleep
     void read(int sector, StringBuffer data);   // call sleep
}
```

Printer

Each printer will write data to a file named PRINTERi where i is the index of this printer (starting at 1). A printer can only handle one line of text at a time. It will take **2750 milliseconds** to print one line; the thread needs to sleep to simulate this delay before the printing job finishes.

```
class Printer {
     void print(StringBuffer data);  // call sleep
}
```

UserThread

Each user will read from a file named USERi where i is the index of this user. Each USERi file will contain a series of the following three commands:

```
.save X
.end
.print X
```

All lines between a `.save` and a `.end` is part of the data in file `X`. Each line in the file takes up a sector in the disk. UserThread will handle saving files to disk, but it will create a new PrintJobThread to handle each print request. Each UserThread may have only **one local StringBuffer** to hold the current line of text. If you read a line from the input file, you must immediately copy that String into this single StringBuffer owned by the UserThread. The

UserThread interprets the command in the StringBuffer or saves the StringBuffer to a disk as appropriate for the line of input.

FileInfo

A FileInfo object will hold the disk number, length of the file (in sectors), and the index of the starting sector (i.e. which sector the first line of the file is in). This FileInfo will be used in the DirectoryManager to hold meta information about a file during saving as well as to inform a PrintJobThread when printing.

```
class FileInfo {
     int diskNumber;
     int startingSector;
     int fileLength;
}
```

DirectoryManager

The DirectoryManager is a table that knows where files are stored on disk via mapping file names into disk sectors. You should use the pre-defined Hashtable to store this information (in the form of FileInfo), but it must be private to class DirectoryManager.

```
class DirectoryManager {
     private Hashtable<String, FileInfo> T =
         new Hashtable<String, FileInfo>();
     void enter(StringBuffer key, FileInfo file);
     FileInfo lookup(StringBuffer key);
}
```

DiskManager

The DiskManager is derived from ResourceManager (given below) and also keeps track of the next free sector on each disk, which is useful for saving files. The DiskManager should contain the DirectoryManager for finding file sectors on Disk.

PrinterManager

The PrinterManager is derived from ResourceManager.

PrintJobThread

PrintJobThread handles printing an entire print request. It must get a free printer (blocking if the are all busy) then read sectors from the disk and send to the printer - one at a time. A UserThread will create and start a new PrintJobThread for each print request. Note if each PrintJob is not a new thread you will get very little parallelism in your OS.

ResourceManager

A ResourceManager is responsible for allocating a resource like a Disk or a Printer to a specific Thread. Note you will have *two instances*: one for allocating Disks and another for allocating Printers, but I suggest you use inheritance to specialize for Disks and Printers. The implementation will be something like the following (you may use mine, but be sure you understand what it does and how it works; do you know why these methods must be synchronized?):

```
class ResourceManager {
     boolean isFree[];
     ResourceManager(int numberOfItems) {
          isFree = new boolean[numberOfItems];
          for (int i=0; i<isFree.length; ++i)
               isFree[i] = true;
     }
     synchronized int request() {
          while (true) {
               for (int i = 0; i < isFree.length; ++i)
                    if ( isFree[i] ) {
                         isFree[i] = false;
                         return i;
                    }
               this.wait(); // block until someone releases Resource
          }
     }
     synchronized void release( int index ) {
          isFree[index] = true;
          this.notify(); // let a blocked thread run
     }
}
```

I suggest you start on the GUI this week too (described below) – at least do the layout and any controls (e.g. start), and include some way to control the speed. You may choose to do PrintJobThread and ResourceManager in the next homework, but make sure the rest of the OS works as intended.

The autograder will test that you correctly create the supplied number of printers and that you print at least something in to each one of them. We will not test load balancing until the submission for homework 9.

Submit the following:

- **a zip of the electronic version of your OS on Gradescope**

Homework 9

Due Wednesday, March 11th at 2 AM, HARD DEADLINE

[100] DESIGN SKETCH: PART TWO

Build by hand-coding (i.e., don't use a program to generate the GUI program) a GUI that displays the status of the 141OS. I want to see the following whenever the state of your program changes:

- what each **User** is doing
- what each **Disk** is doing
- what each **Printer** is doing

The details of the GUI are left up to you. You may use Swing, AWT, or JavaFX to develop the GUI to your liking. Add speed control so the user can speed up or slow down the simulation!

If you do something impressive, I want to see a personal demo. We will give extra credit of zero, one, two, or three points (in course score scale) for impressive implementations of 141OS – depending on how impressive it is beyond the basic GUI. This is almost the value of one entire homework assignment. Extra credit demos can be in office hours 10th week or on Monday of final's week between noon and 5pm. No extra credit demos after that time. I will hold for demo appointments in DBH 3074.

IMPORTANT (for both HW8 and HW9): Handle the following command line arguments that allows operation of your program *without* the GUI. This will be done as a flag to your command line input via "-ng":

```
$ java -jar 141OS.jar -2 USER1 USER2 -3 -4 -ng
```

It is **very** important you remember this step for the autograder.

IF YOU ATTEMPT TO RUN GRAPHICS ON A HEADLESS MACHINE LIKE THE ONES THE AUTOGRADER USES, YOUR PROGRAM WILL CRASH.

For the visual GUI, you will be submitting a URL to a video on YouTube. Here are the following requirements for the video:

- Use the sample inputs given in the homework as the User inputs.
- You will need to make **2 Disks** and **3 Printers** (you can assume it will only be these values)
- You need to have your **Name** and your **UCINetID** visible in the GUI itself, either before the simulation or during the simulation.
- Take a screen recording of your GUI in action with these inputs in mind. This simulation will take around **3 minutes** approximately with the normal speed. Keep it at normal speed, but your speed control should be visible.
- Upload that screen recording onto **YouTube**, and submit the URL of that link onto the Canvas submission. You only need that link for the Canvas submission.

We will be looking for these things:

- Printers
- Disks
- Users
- Speed Control
- *Concurrency* – this is **important.**

Submit the following:

- **a zip of the electronic version of your entire program including the GUI on Gradescope**
- **a video of your GUI in action also on gradescope**

If you want to apply for extra credit, you must come to office hours to demonstrate your GUI to Dr. Klefstad. He will set time to examine more projects during Finals Week as well.