Goals for this lab: ****

1. **Learn about the different CPU scheduling algorithms**
2. **Make observations about the pros/cons of each one**
3. **Compare the algorithms using different metrics**
4. **Implement some algorithms yourself**

First, we need to trace out some CPU scheduling examples on paper, including the Gannt chart, wait time, turnaround time, and response time for each process from sample data.

I've placed scans of my papers where I did this in the same directory as this PDF in my repo.

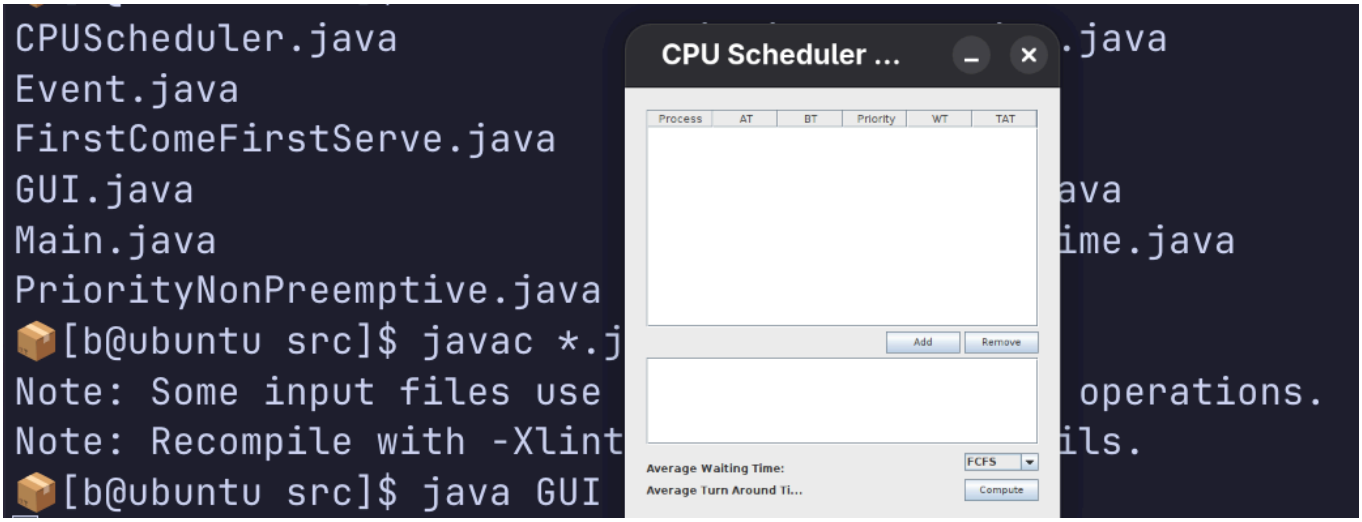
This lab requires we check our work with a java application to compare different CPU scheduling algorithms. I've setup java in an Ubuntu container.

First, I cloned the repo. Now, I have to compile all java files in `src` from the project.

```
📦[b@ubuntu src]$ ls
CPUScheduler.java           PriorityPreemptive.java
Event.java                  RoundRobin.java
FirstComeFirstServe.java    Row.java
GUI.java                    ShortestJobFirst.java
Main.java                   ShortestRemainingTime.java
PriorityNonPreemptive.java  Utility.java
📦[b@ubuntu src]$ javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

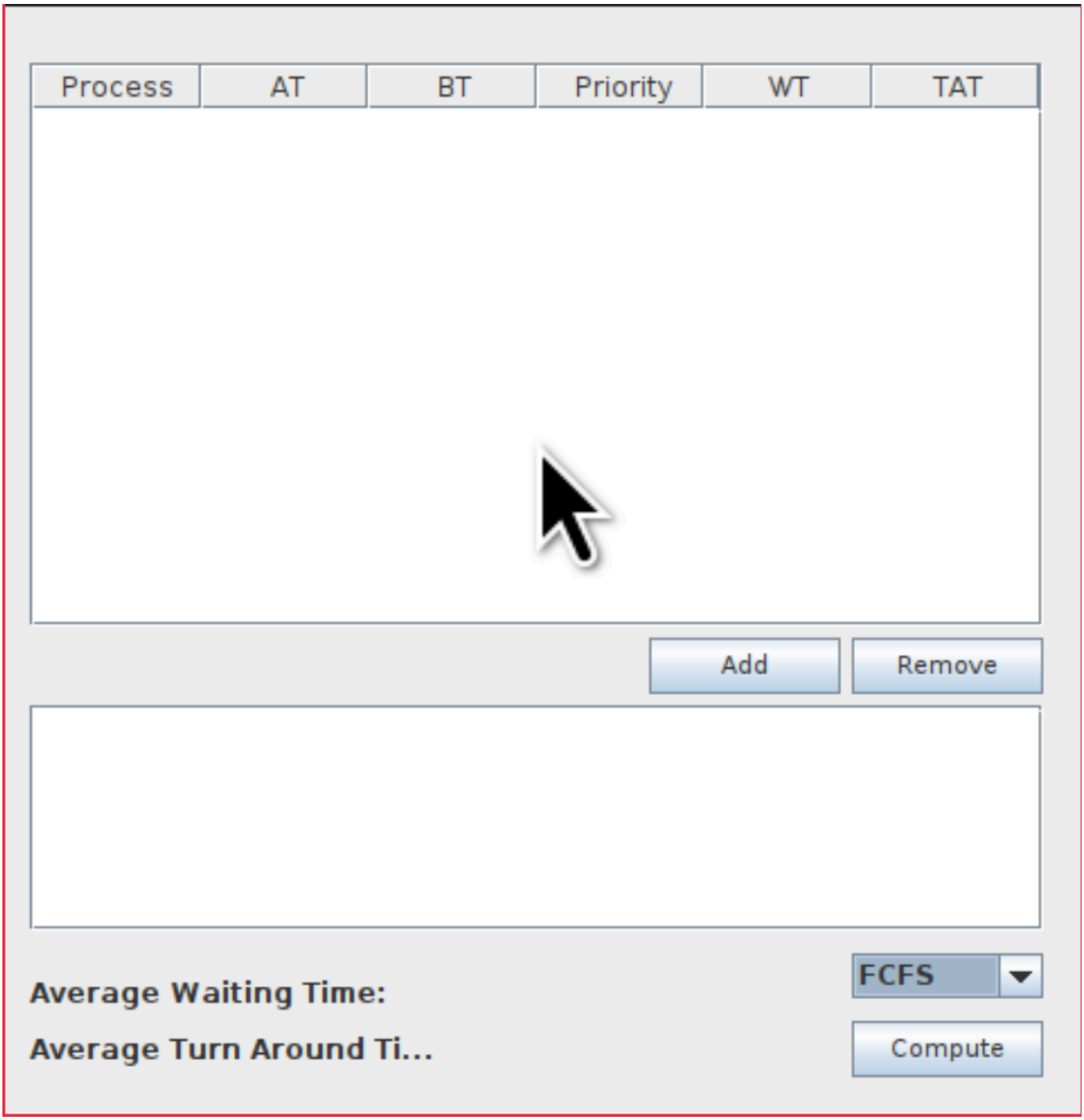
We get a warning about "unchecked or unsafe input files". This is just a compiler warning. I'm not concerned because this is just for the lab, not for anything too serious.

Next, we are tasked to run java GUI.



We can see it launched the GUI, just very very small.

I can't figure out how to resize the size of the window. I was able to scale the interface in the window, but that doesn't really help if the window is tiny. So, my solution is to use the magnifier from accessibility features.



It's a temporary solution but at least I can see what I'm doing.

Now, we have a nice area to experiment and check our work with different

algorithms.

I checked over my handwritten work and fixed any errors. The posted handwritten work is correct.

Observations:

Which algorithms are preemptive?

- SRT
- PSP

Which algorithms give the best/lowest average wait time?

- SJF
- SRT

Which algorithms give the best/lowest average turnaround time?

- SJF
- SRT

Which algorithms give the best/lowest average response time?

- Round Robin Quantum 2
- SJF
- SRT
- PSP

Which algorithm(s) give the most 'fair' access to each process to the CPU

- Round Robin

What is the response time for the non-preemptive algorithms?

- The same as the wait time.

What is the relationship between lowering wait times and turnaround times?

- When one is decreased, the other decreases.

Do you have any idea how to blend priority, fairness and turnaround times?

- Round Robin with an appropriate quantum time

Can you think of any other ways to do the scheduling?

- The opposite of SJF:
  - LJF: Longest Job First!

Implement

For this section, we attempt to alter the code of the Java GUI we were given to include new custom algorithms. Other students have reported many issues with this, so I don't fully expect

to create a working product.

Add your algorithm to work with the Java GUI:

Let's see if I can implement Longest Job First. We need to open up the source code for the Java GUI in VSCODE.

All I did was copy paste the code for ShortestJobFirst.java reverse the sorting, then in GUI.java I added LJF to the JComboBox :

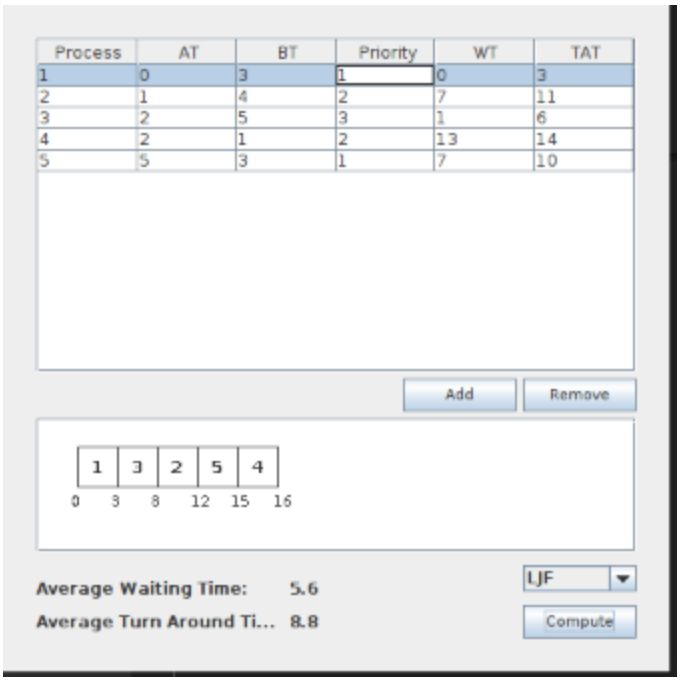
```
option = new JComboBox(new String[]{"FCFS", "SJF", "SRT", "PSN", "PSP",
"RR", "LJF"});
```

And to the switch cases case "LJF":

```
scheduler = new LongestJobFirst();

break;
```

option.setBounds(390, 420, 85, 20);`



Write a scheduling algorithm that works like PSN but decreases the priority of any *processes by one waiting in the queue for X+ (i.e. 3+) time. Why might we want to decrease priority of processes after a time period?

For this scheduling algorithm, I started by working off the PSN class.

I added a `WAIT` variable to state how long a process can sit in the queue before decreasing priority:

```
final int WAIT = 3;
```

Then I declared variables to track each process's wait time and priority level.

```
int wait1 = time -row1.getArrivalTime();

int wait2 = time -row2.getArrivalTime();



int priorityLevel1 = row1.getPriorityLevel();

int priorityLevel2 = row2.getPriorityLevel();
```

Next, I compared the waiting times with our `WAIT` variable and decreased if above.

```
if (wait1 >= WAIT){

     priorityLevel1++;

}

if (wait2 >= WAIT){

     prioritylevel2++;

}
```

Finally, we compare the priorities of the two rows and return value to sort accordingly:

```
if (priorityLevel1 ==priorityLevel2)

{

     return 0;

}

else if (priorityLevel1 < priorityLevel2)
```

```
{
     return -1;
}
else
{
     return 1;
}
```

I can imagine this sort of process scheduling could be useful in a situation like a RTOS for something like missile guidance, where a process that takes too long to run is less important to run, and ultimately could just be ignored entirely.

Write a scheduling algorithm that works like PSN but decreases the priority of any processes by one waiting in queue for X+ (i.e. 3+) times and increases the priority of a process leaving CPU during the context switch

All I did here was copy the class I just made into a file called `SwitchingPriorityNonPreemptive.java` and make 1 change:

```
for (Row row : rows){
     if(row.getPriorityLevel() > 1){
          row.setPriorityLevel(row.getPriorityLevel() - 1);
     }

}
```

This boosts the priority of any processes coming off of the CPU during the context switch.

This could be useful to be more "fair" to processes that are stuck waiting for too long, for example in a load balancing situation.