**Lab 9**

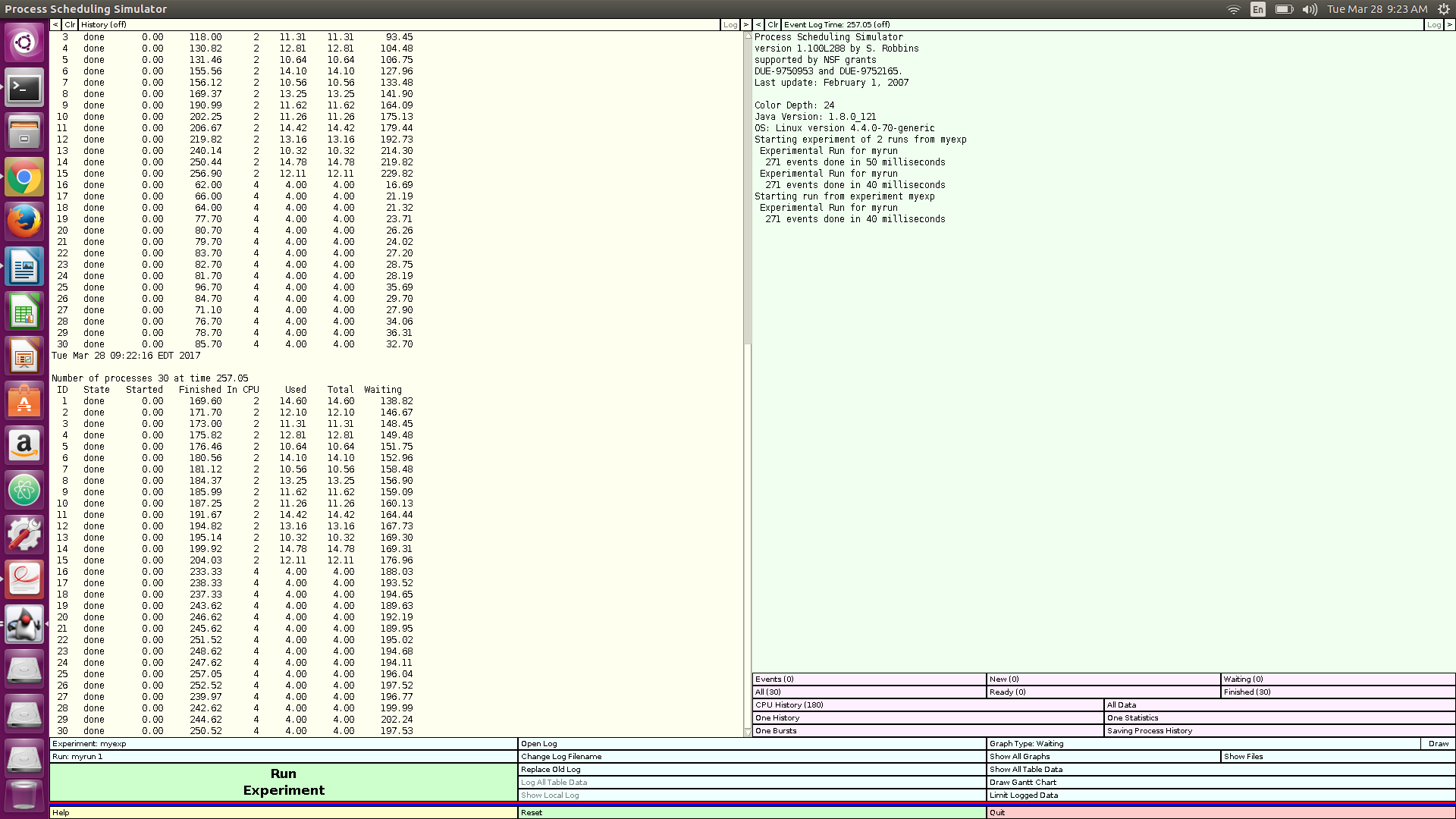
**Task1**

4a) Immediately after the experiment completes the read out confirms that it ran **271 events per run**.

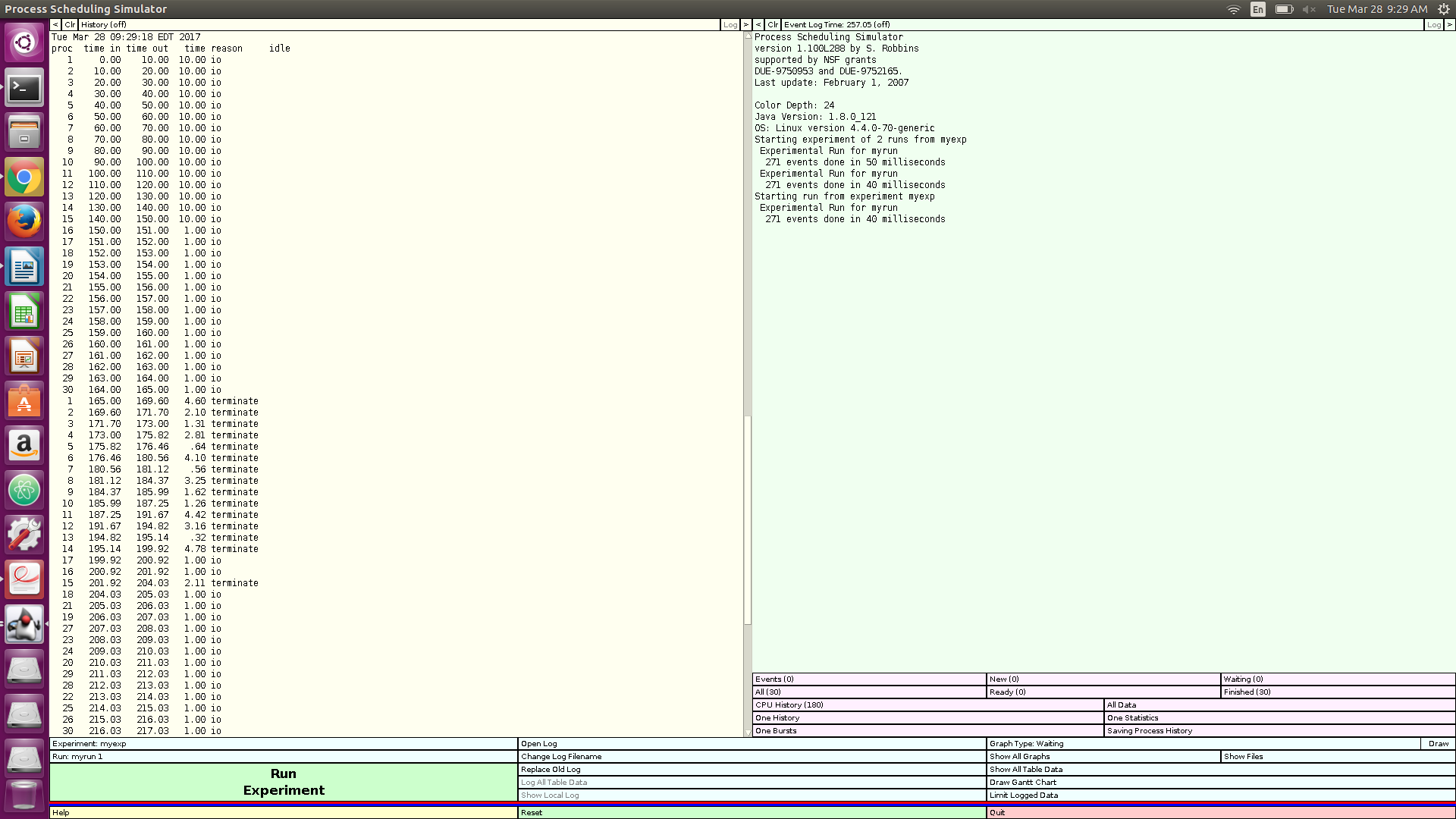
4b) Each experiment took a different amount of time. The first took **50ms** the second took **40ms**.

4c) By clicking “show all table data” I am able to see that **in each run 30 processes were created**.

5a) I examined the data for the All(#) selector attached below is a screen shot.

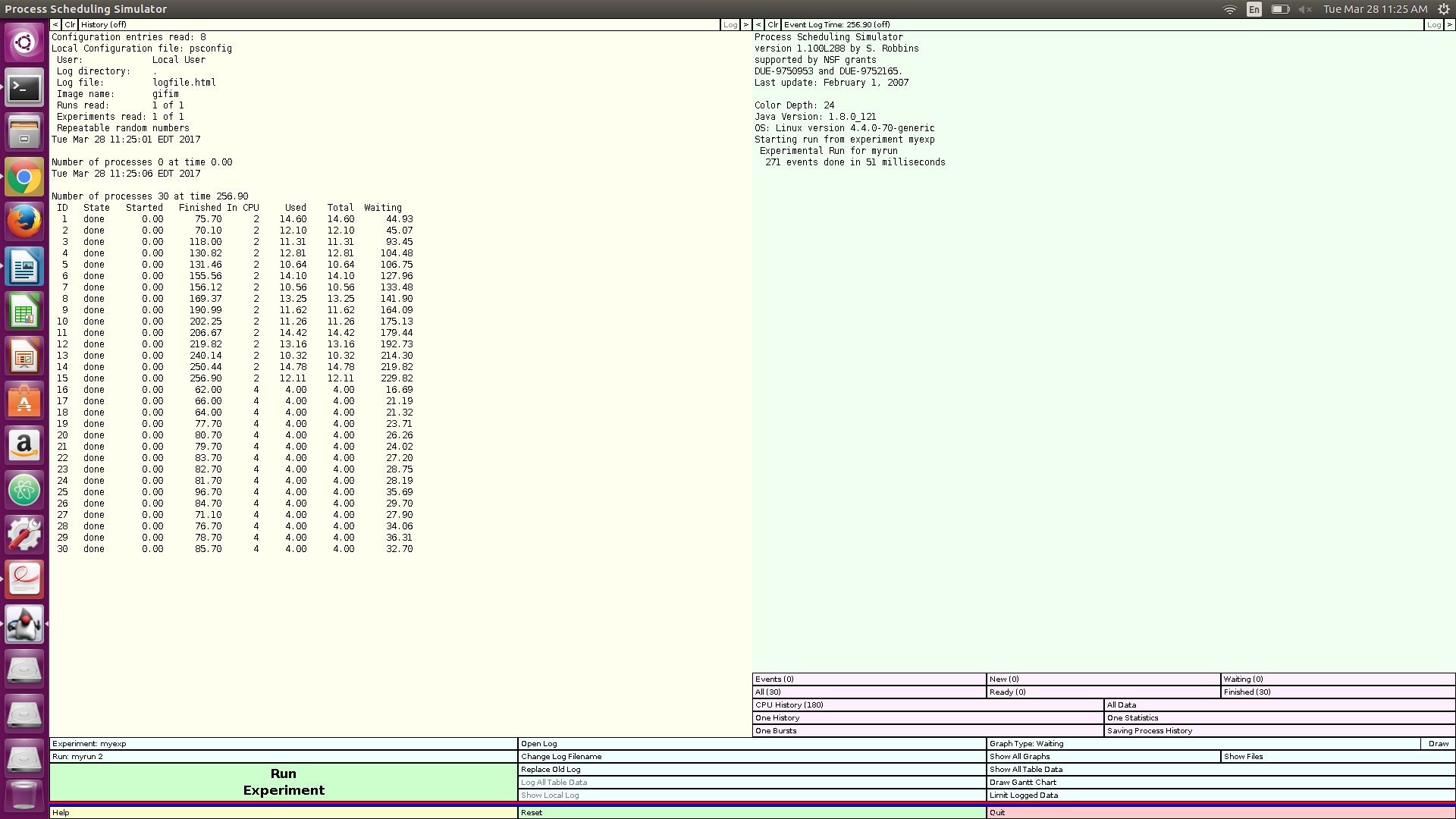


5b) I examined the CPU History (#) selector attached below is a screen shot.

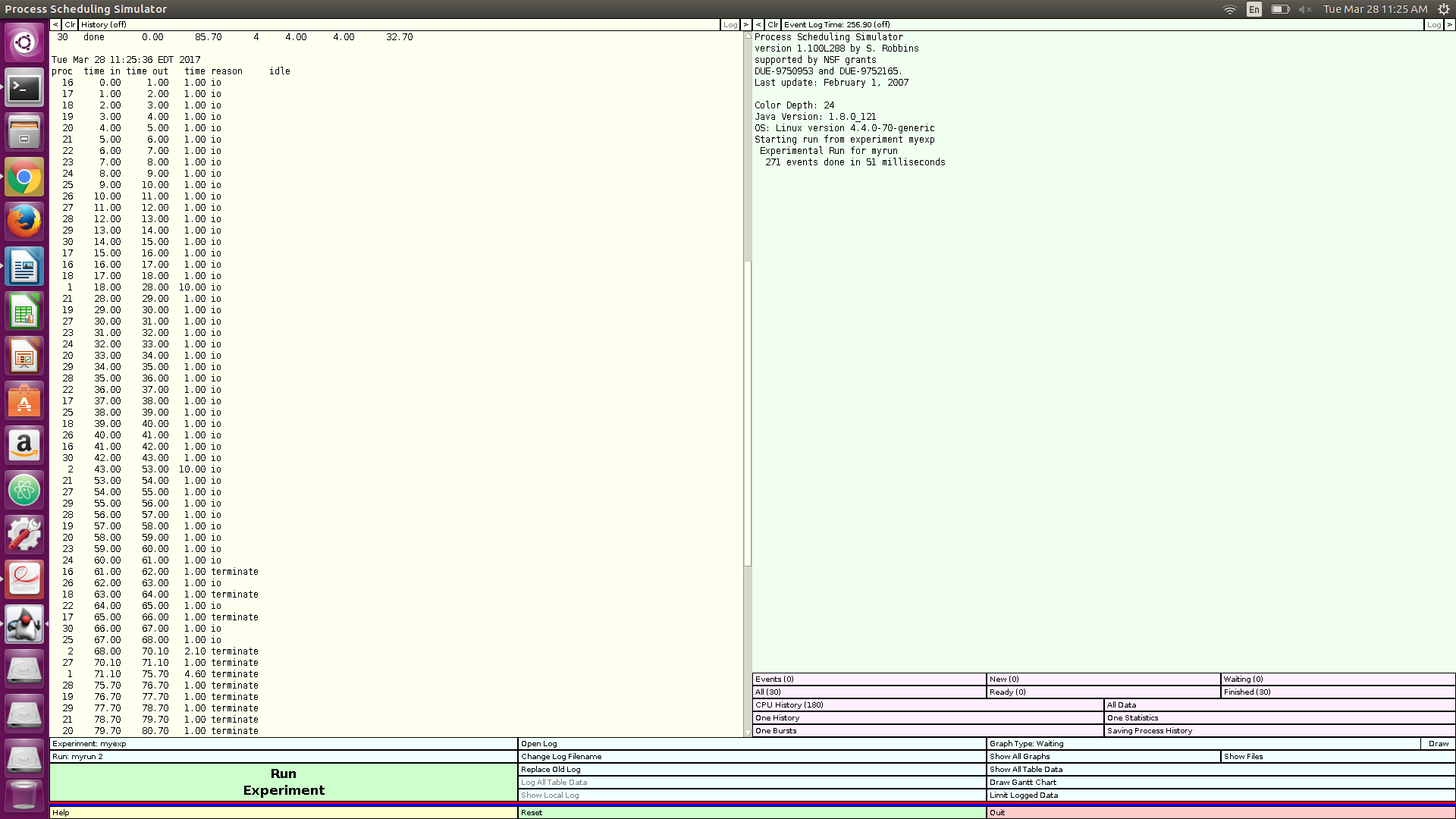


5c) By using the FCFS or first come first serve algorithm, each job is scheduled sequentially one after the other in the order that they come into the queue but process 30 functions a bit weirder than the rest because it is one of the last 5 terminate and comes into and out of the CPU 4 times, 3 of those 4 being for IO handling, it takes this long because of the FCFS. It cannot perform it’s IO until that resource gets opened up because the other processes that came before it are not using it. This order was in-sync until the second round of IO with processes 1-13 all terminating one after the other and 15 and 16 after some iterations of IO which changed the ordering up quite a bit. So even though 30 was traditionally last it was able to complete before processes 26,25 and 22.

6a) I examined the data for the All(#) selector attached below is a screen shot.



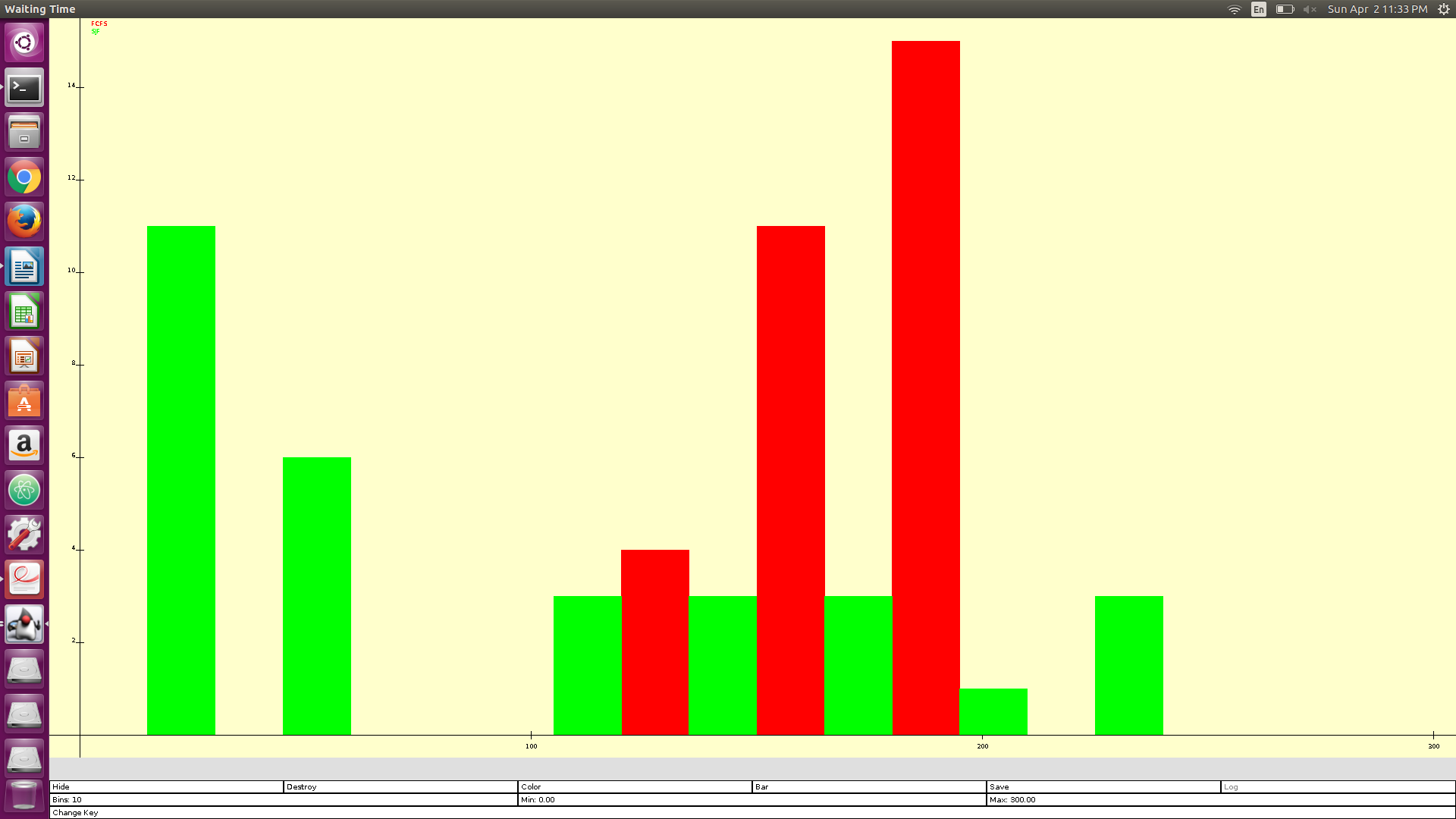
6b) I examined the CPU History (#) selector attached below is a screen shot.



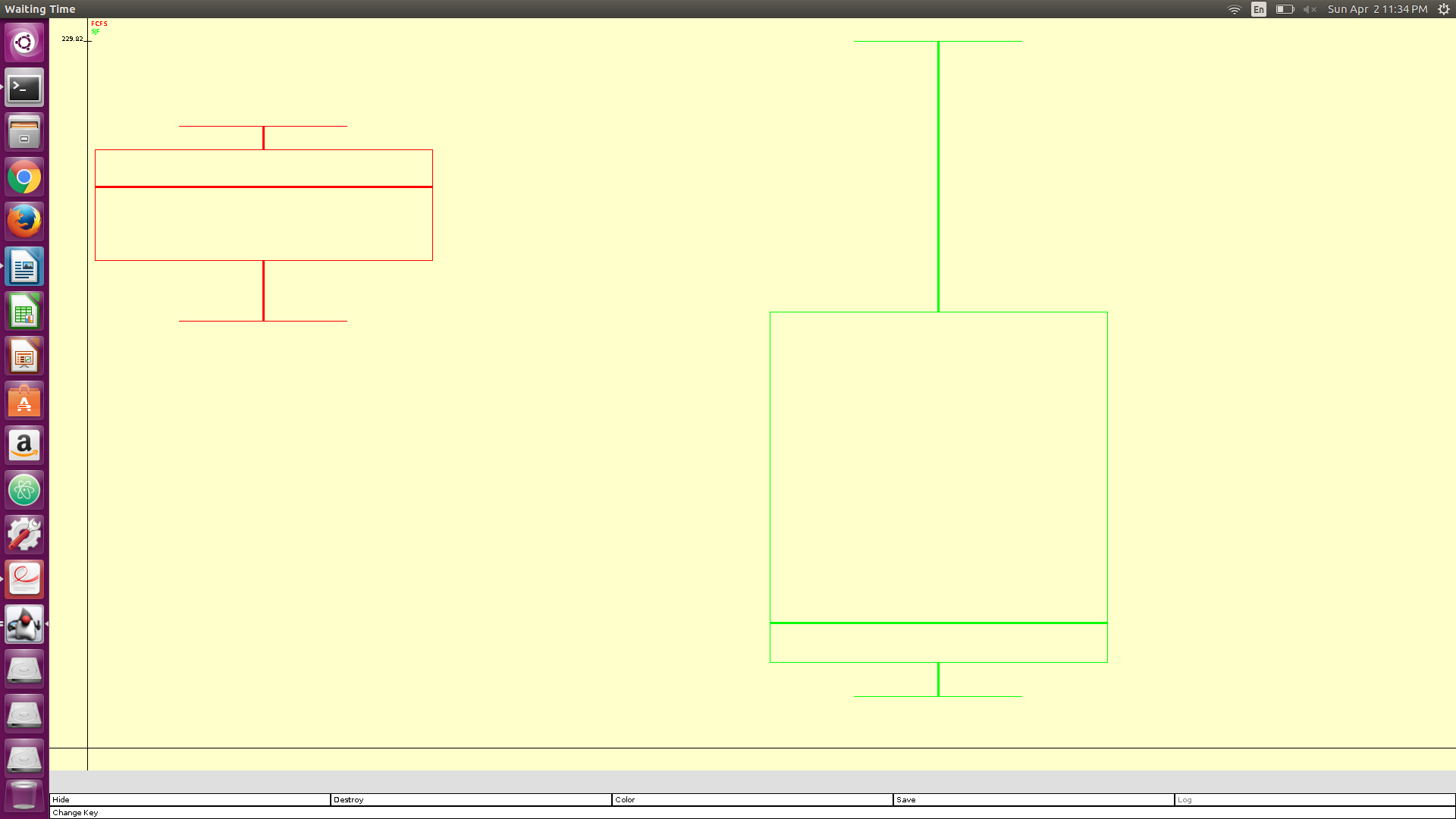
6c) Now we are using the SJF or shortest job first algorithm so process 30 gets scheduled after processes shorter than it but before processes longer than it.

6d) In comparison to the previous algorithm, process 30 terminates earlier than the last 5 as it is able to be completed faster by giving it a higher priority advantage due it’s size in relation to the other processes. It comes back for IO 3 times and terminates on the fourth like before but in this case SJF is more optimized for process 30 because it finishes sooner.

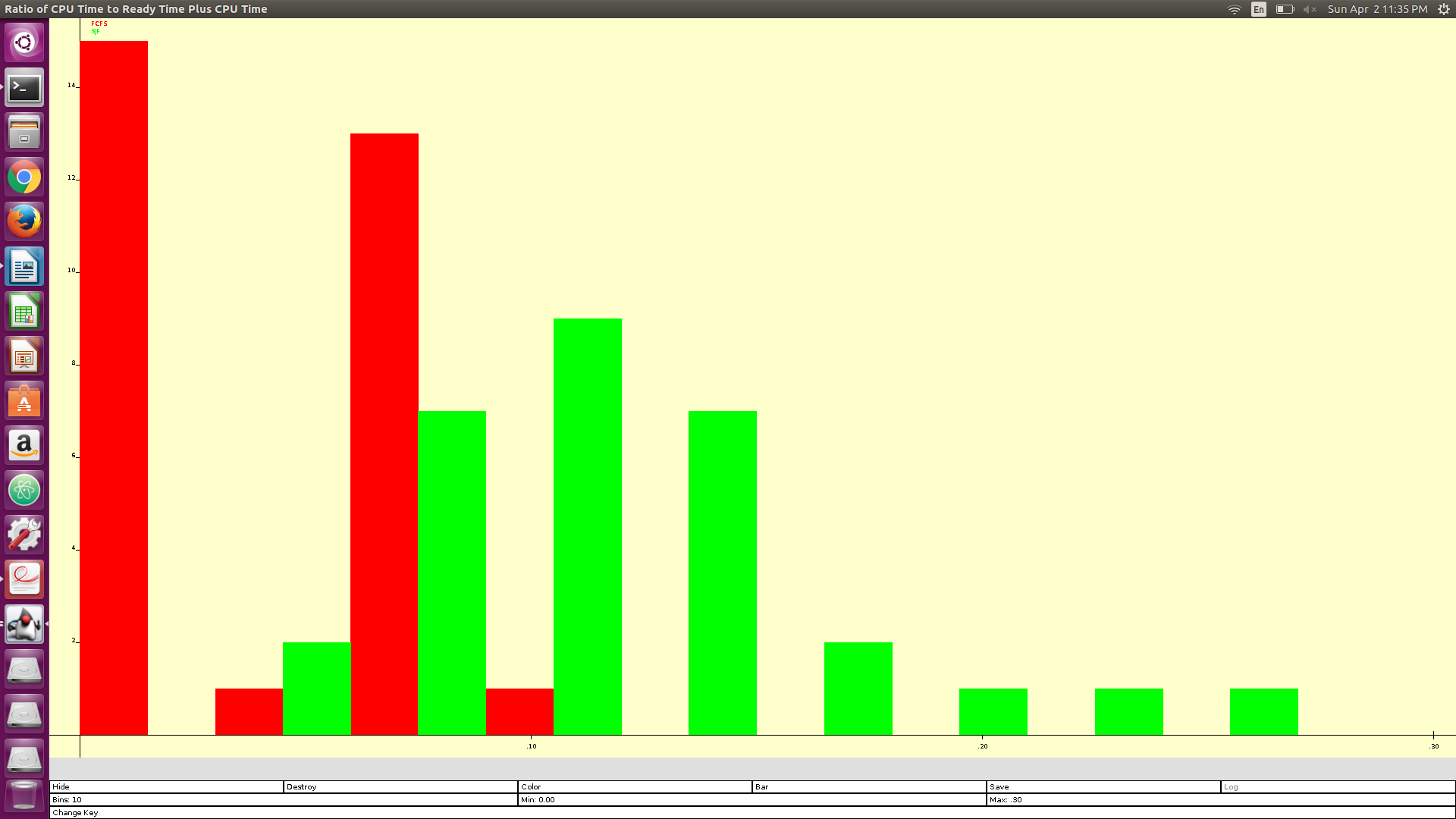
7a) Waiting time:



Waiting time box:



Waiting Ratio:

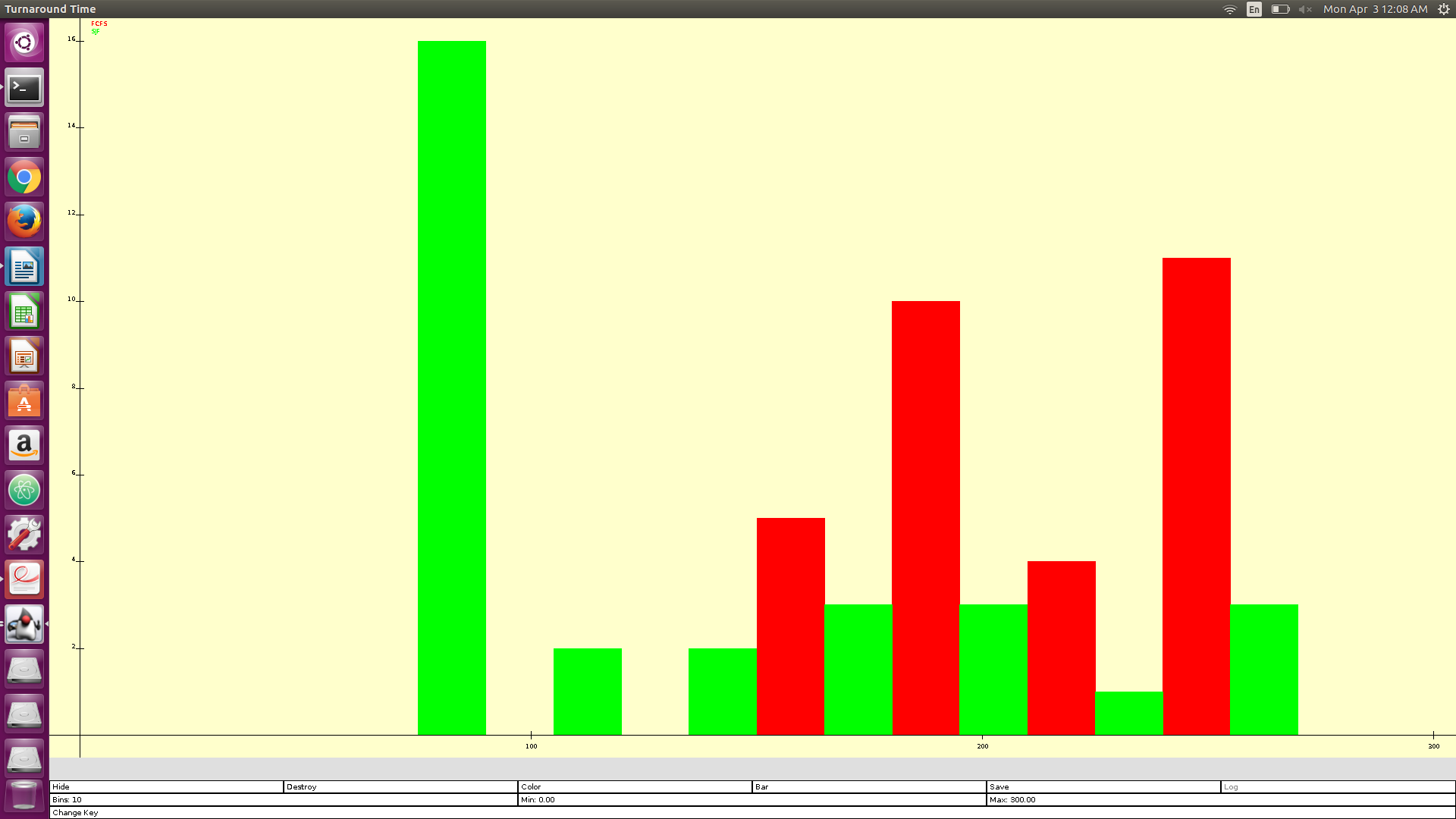


i) Shortest job first has the shorter average wait time. This would be due to the slight speed bump that comes along with doing the processing first to the shorter process.

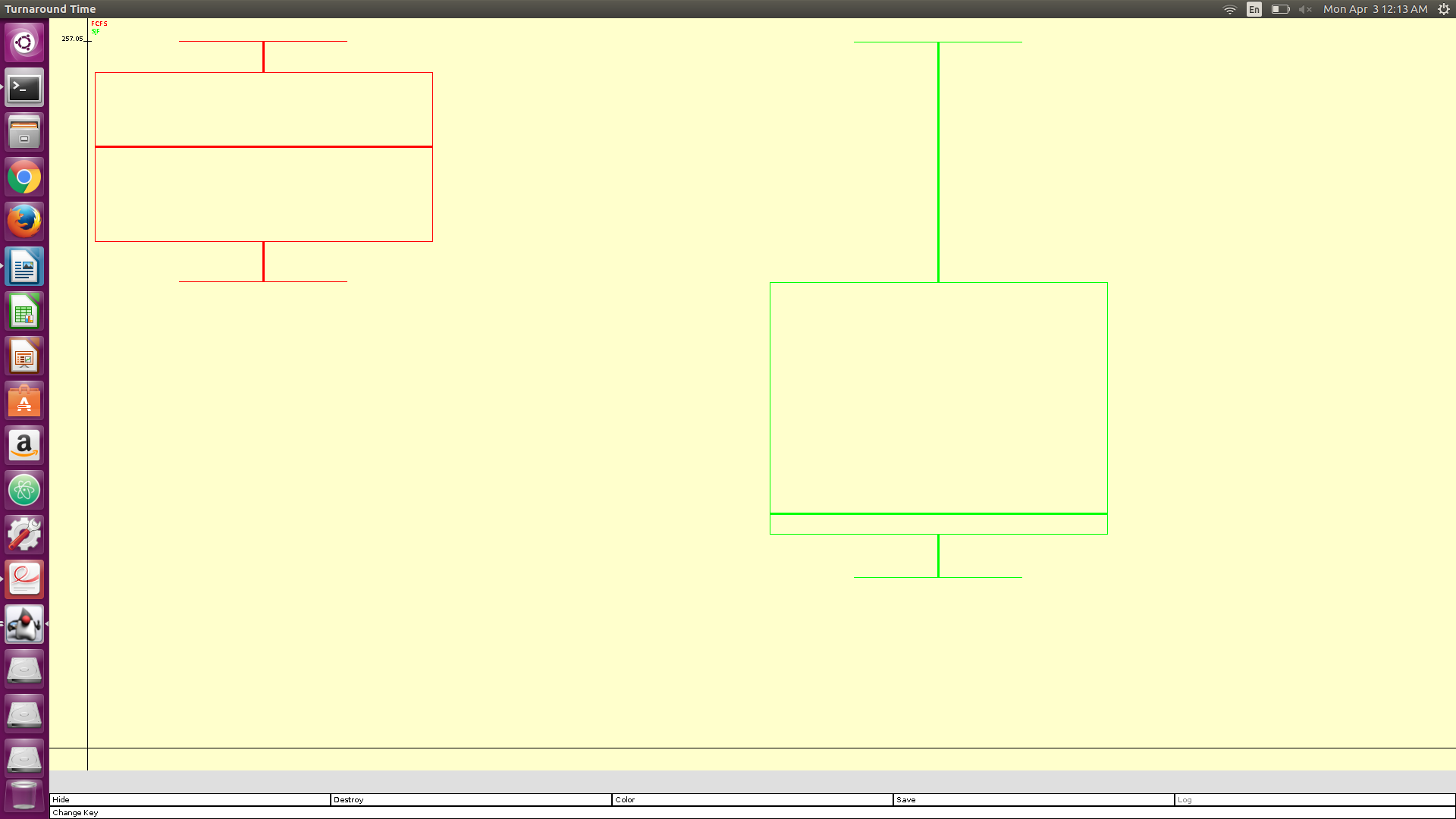
ii) Due to the SJF's habit of making the largest thing wait until the end. The max wait time is high and that’s the drawback.

iii) In my opinion I believe that both algorithms have their benefits, for FCFS it’s fast like very fast it hits things as they come and if the processes are of similar sizes then they are being handled effectively. The con is that the algorithm doesn’t accommodate well for things of varying sizes. If you have a large process taking up resources for a long period of time it causes an unnecessary wait time for the whole batch of jobs. Now in comparison the SJF algorithm, allows for that by allowing the smaller jobs to go before the bigger ones so that’s it’s strength. It’s negative is that it becomes awful when all the jobs are the same size because it wastes resources trying to scan them all for smaller ones.

7b) Turnaround:



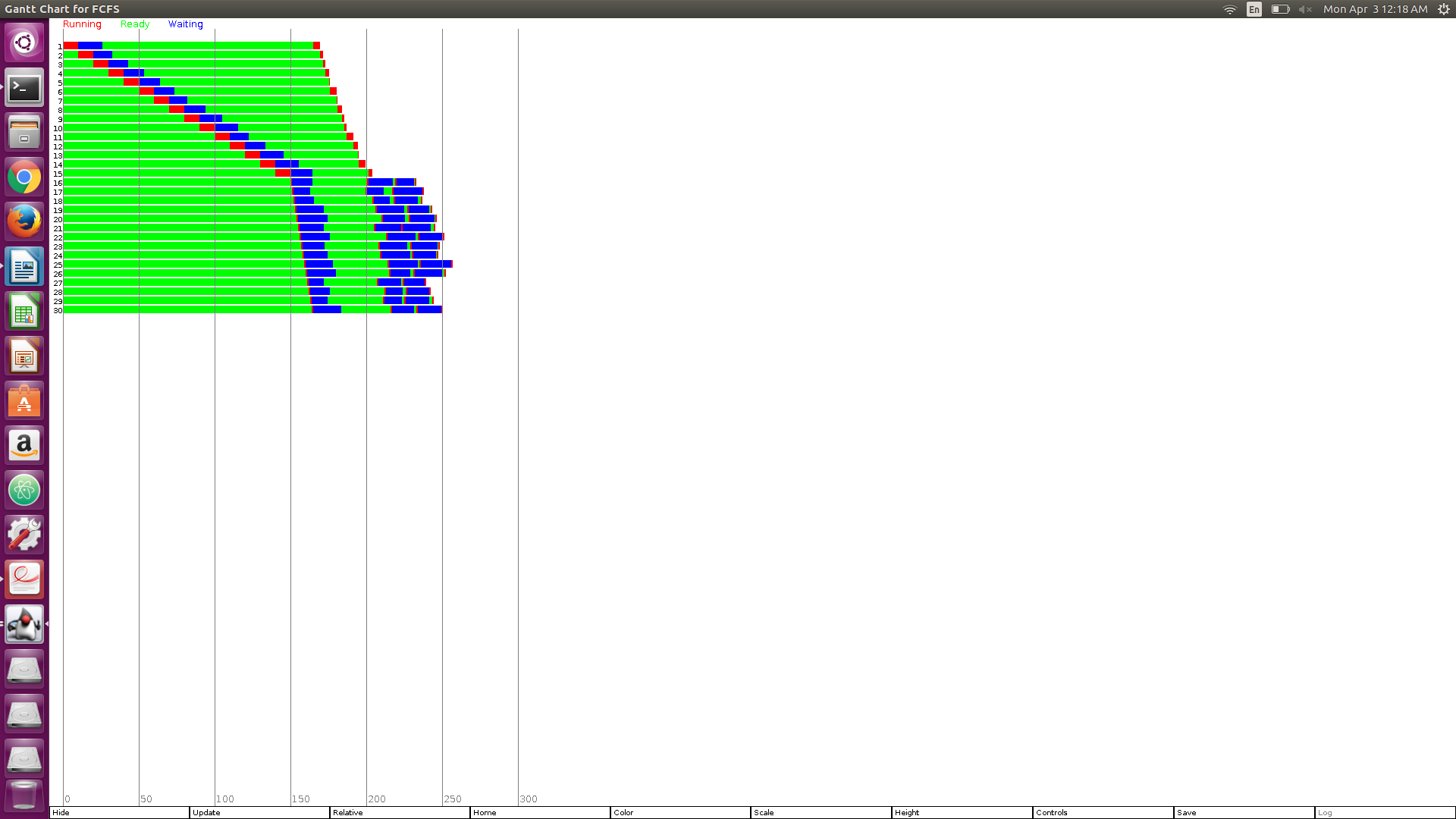
Turnaround box:



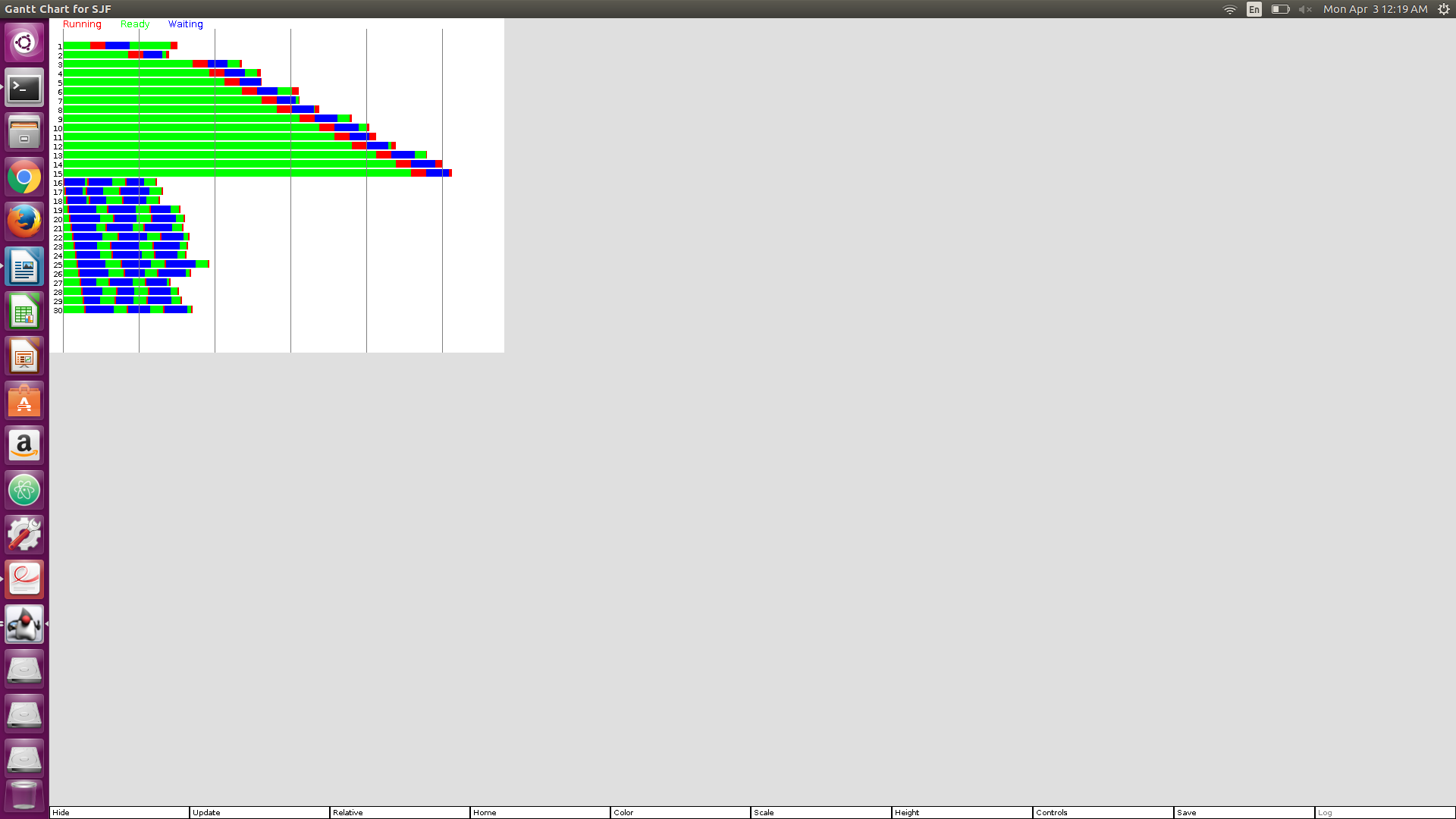
i) Turnaround time is the amount of time it takes to submit a process to the CPU and for the CPU to compute and return an output of that process

ii) The SJF has the lower turn around time and it’s because shorter processes naturally return faster so this would keep the turnaround time as minimal as possible.

7c) Gantt FCFS:



Gantt SJF:

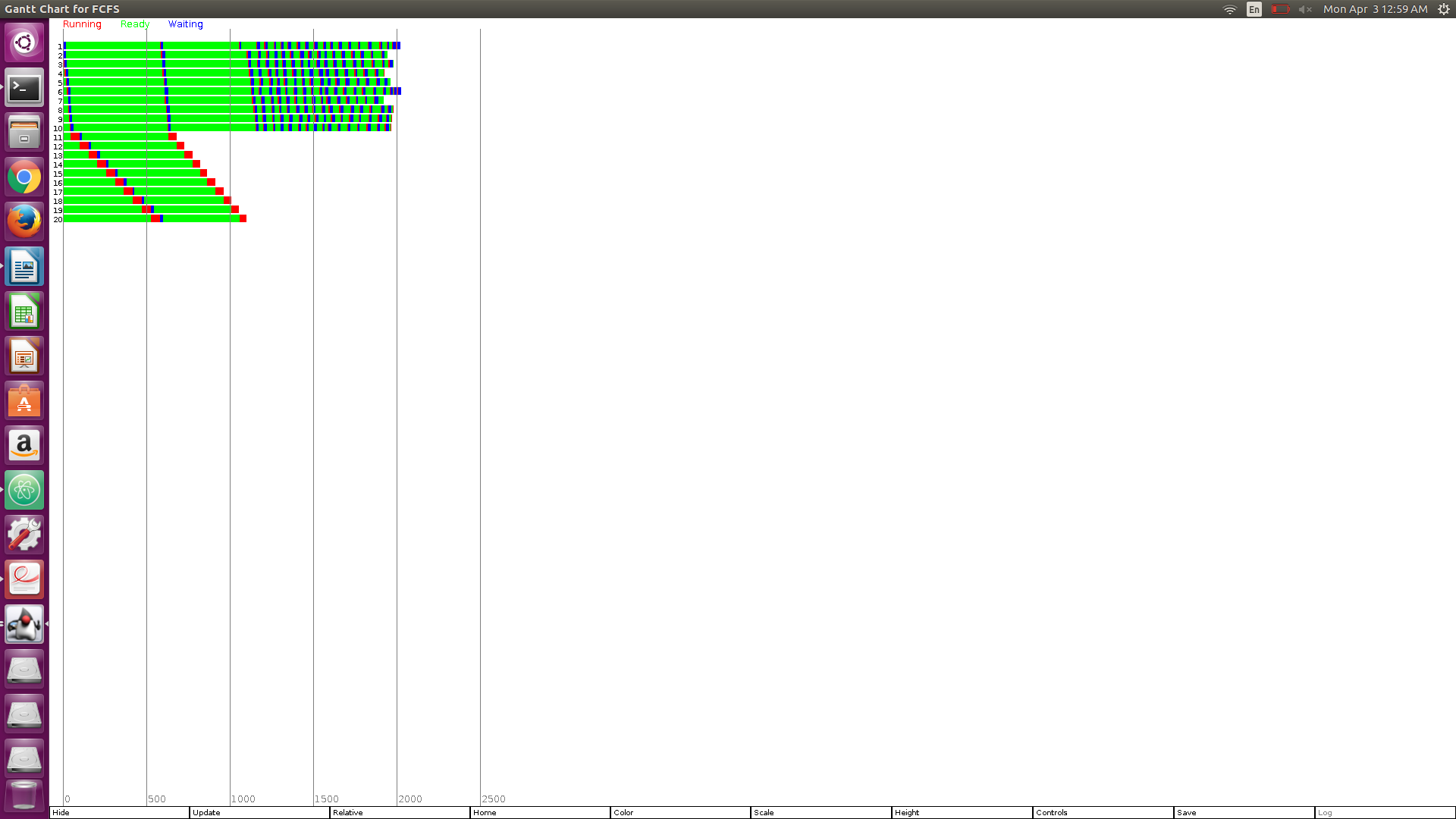


**Task2**

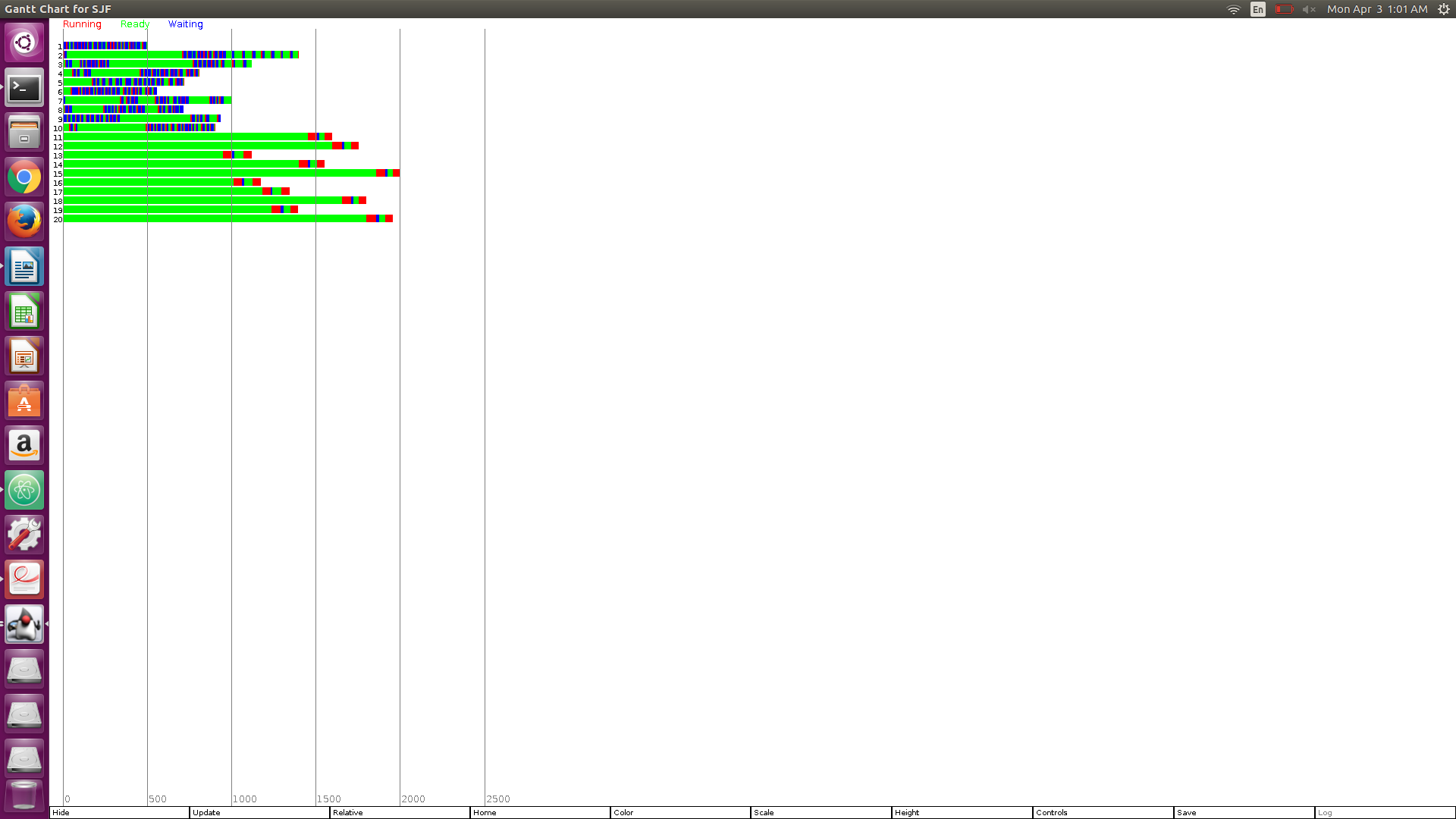
No deliverable for this task

**Task3**

Gantt Chart for FCFS:



Gantt chart for SJF:



**Task4**

myexp.exp is below here:

name myexp

comment This experiment contains 2 runs

run myrun algorithm FCFS key "FCFS"

run myrun algorithm SJF key "SJF"

myrun.run is below here

name myrun

comment This contains two types of processes

algorithm SJF

seed 5000

numprocs 10

firstarrival 0.0

interarrival constant 0.0

duration constant 100

cpuburst uniform 2 8

ioburst uniform 10 20

basepriority 1.0

numprocs 10

firstarrival 0.0

interarrival constant 0.0

duration constant 100

cpuburst uniform 50 60

ioburst uniform 10.0 20.0

basepriority 1.0

Yes there is a difference, first off SJF barely reaches the 2k mark while FCFS goes over it just a bit. Also in SJF the wait times are mostly bundled towards the end while in SJF they are bundled towards the beginning. In both algorithms the jobs get done but how they go about being done is fairly different.

**Task 4**

myexp.exp is below here:

name myexp

comment This experiment contains 2 runs

run myrun algorithm RR 1 key "RR 1"

run myrun algorithm RR 10 key "RR 10"

run myrun algorithm RR 15 key "RR 15"

run myrun algorithm RR 50 key "RR 50"

run myrun algorithm RR 100 key "RR 100"

run myrun algorithm RR 101 key "RR 101"

myrun.run is below here:

name myrun

comment This contains two types of processes

algorithm SJF

seed 5000

numprocs 10

firstarrival 0.0

interarrival constant 0.0

duration constant 1000

cpuburst constant 101

ioburst constant 100

basepriority 1.0

numprocs 10

firstarrival 0.0

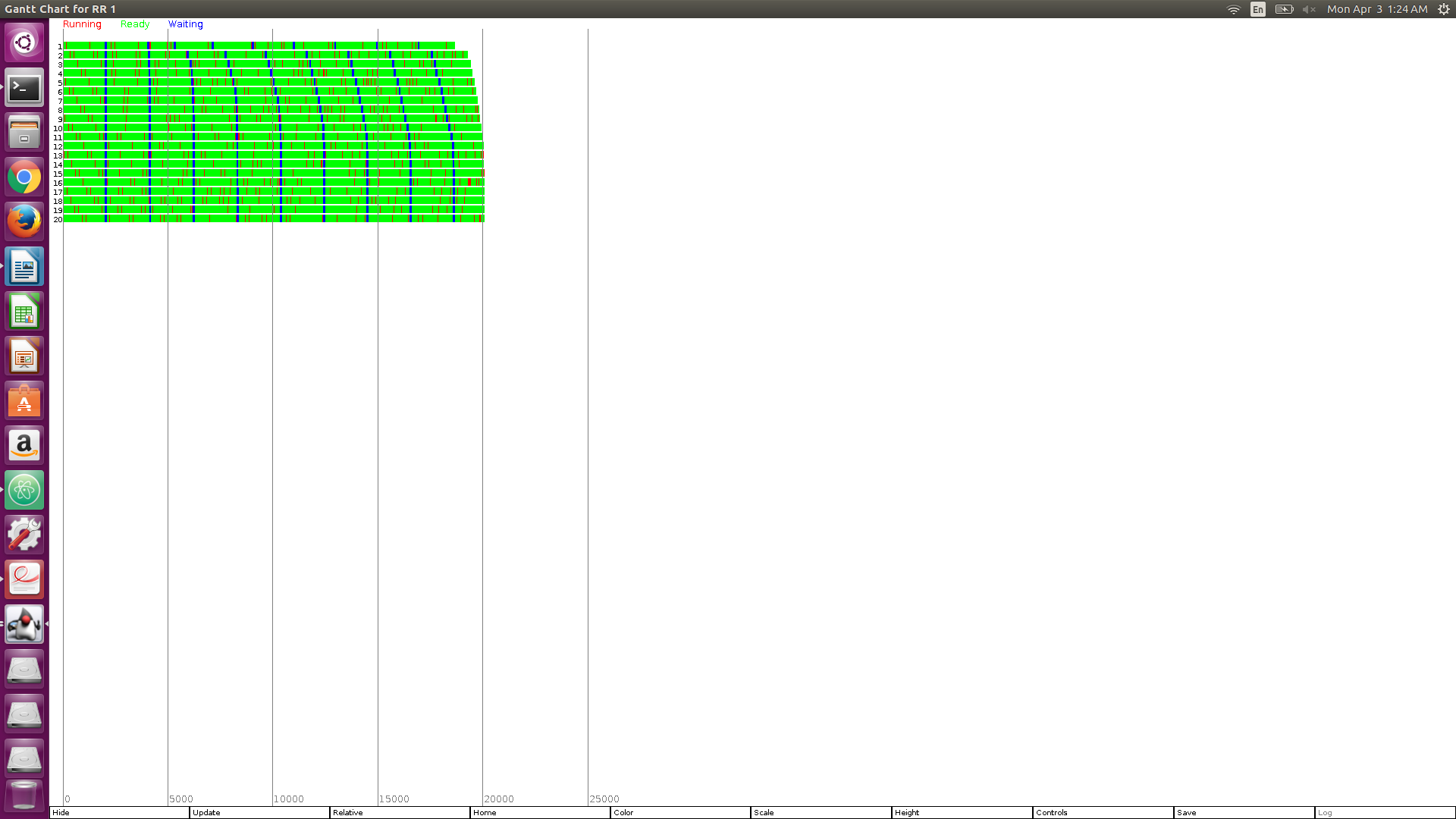
interarrival constant 0.0

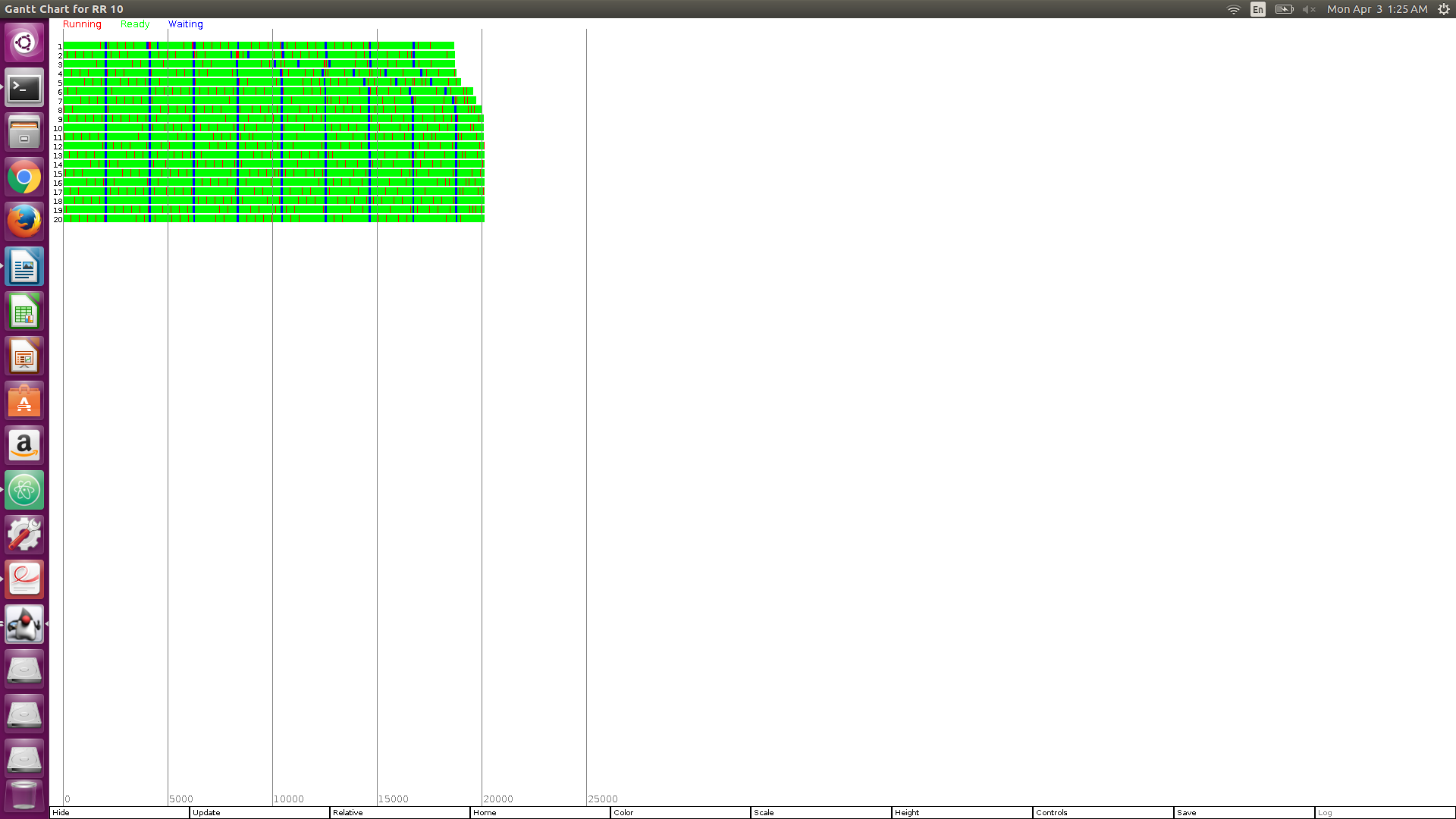
duration constant 1000

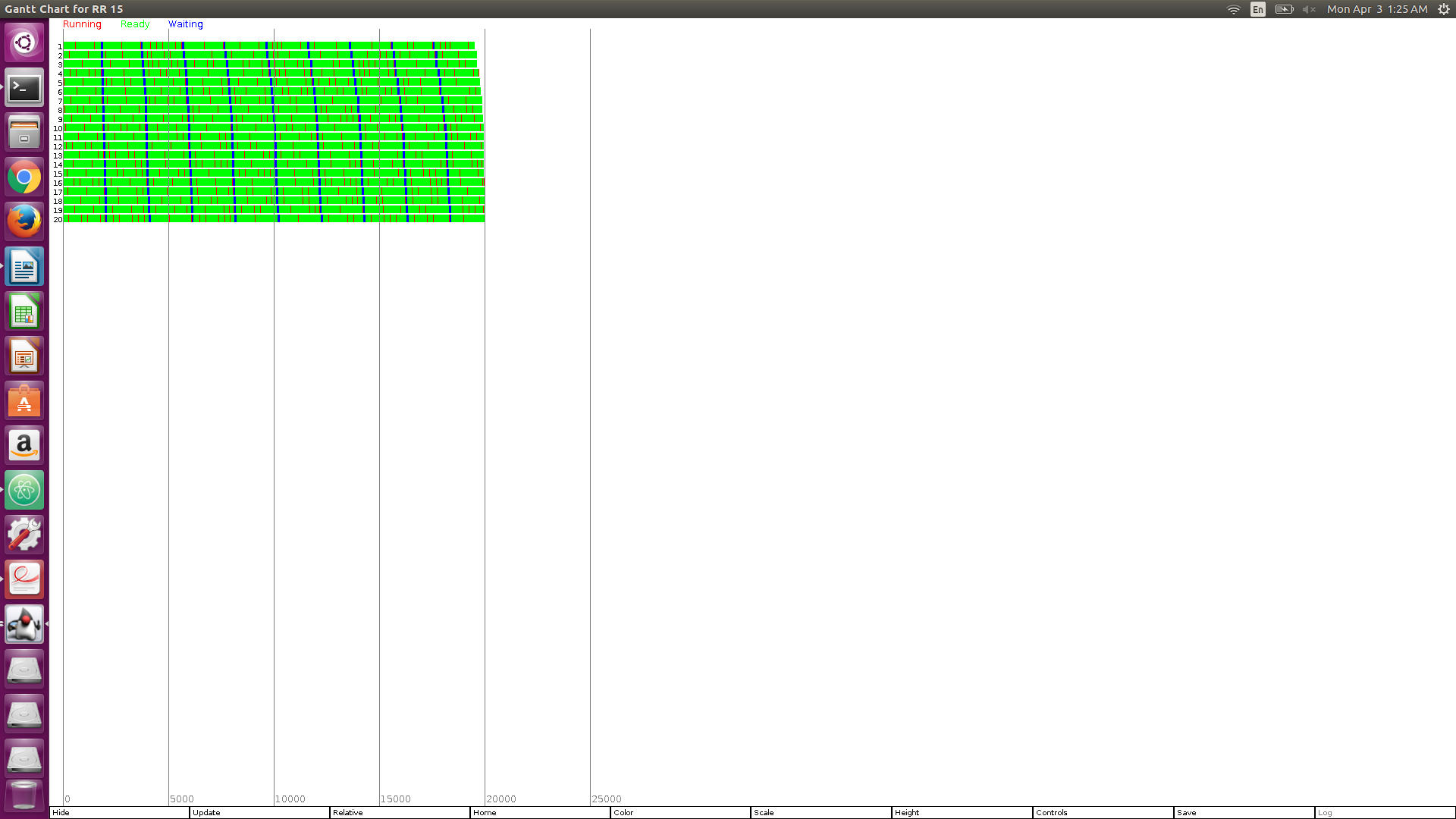
cpuburst constant 101

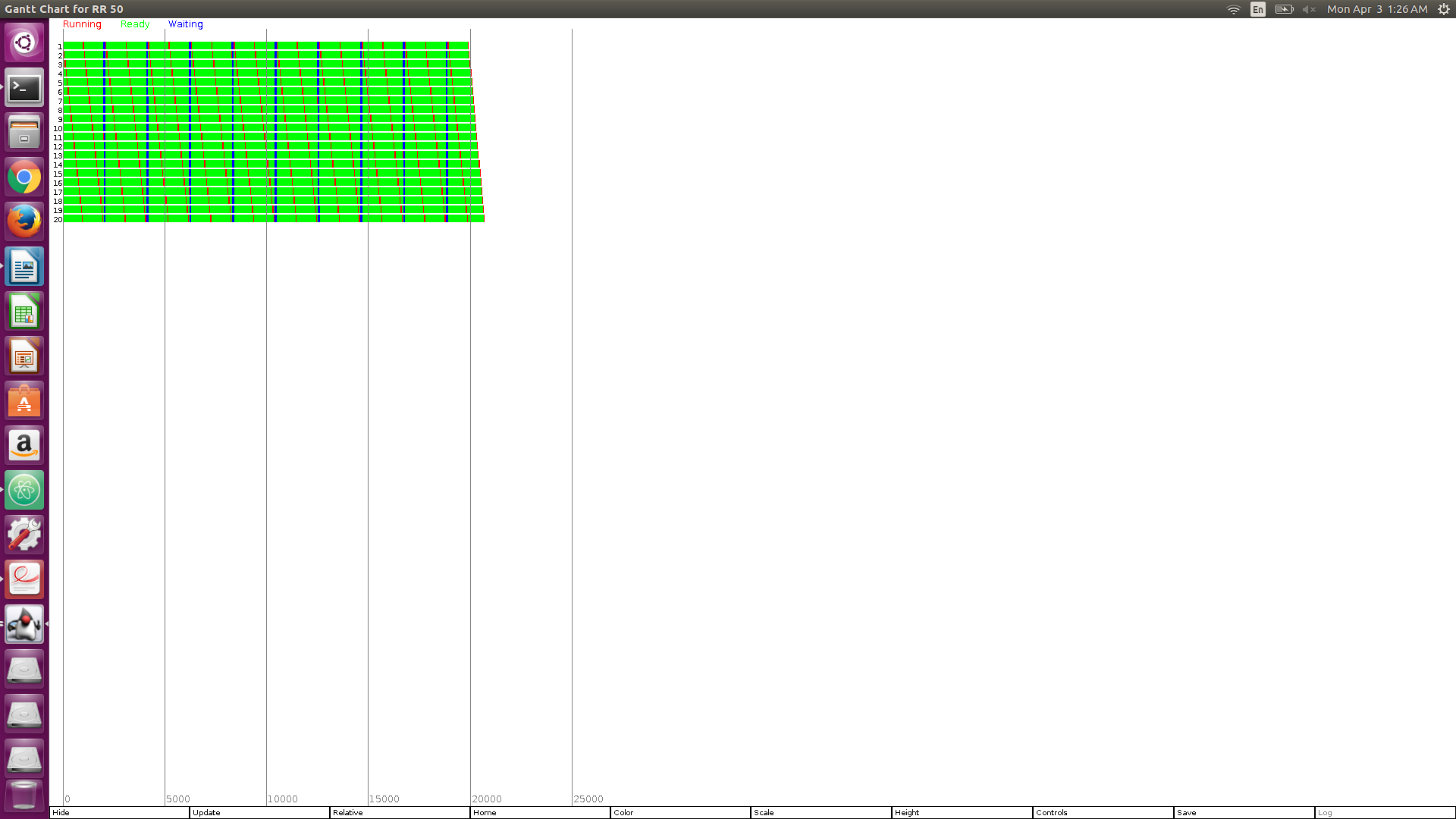
ioburst constant 100

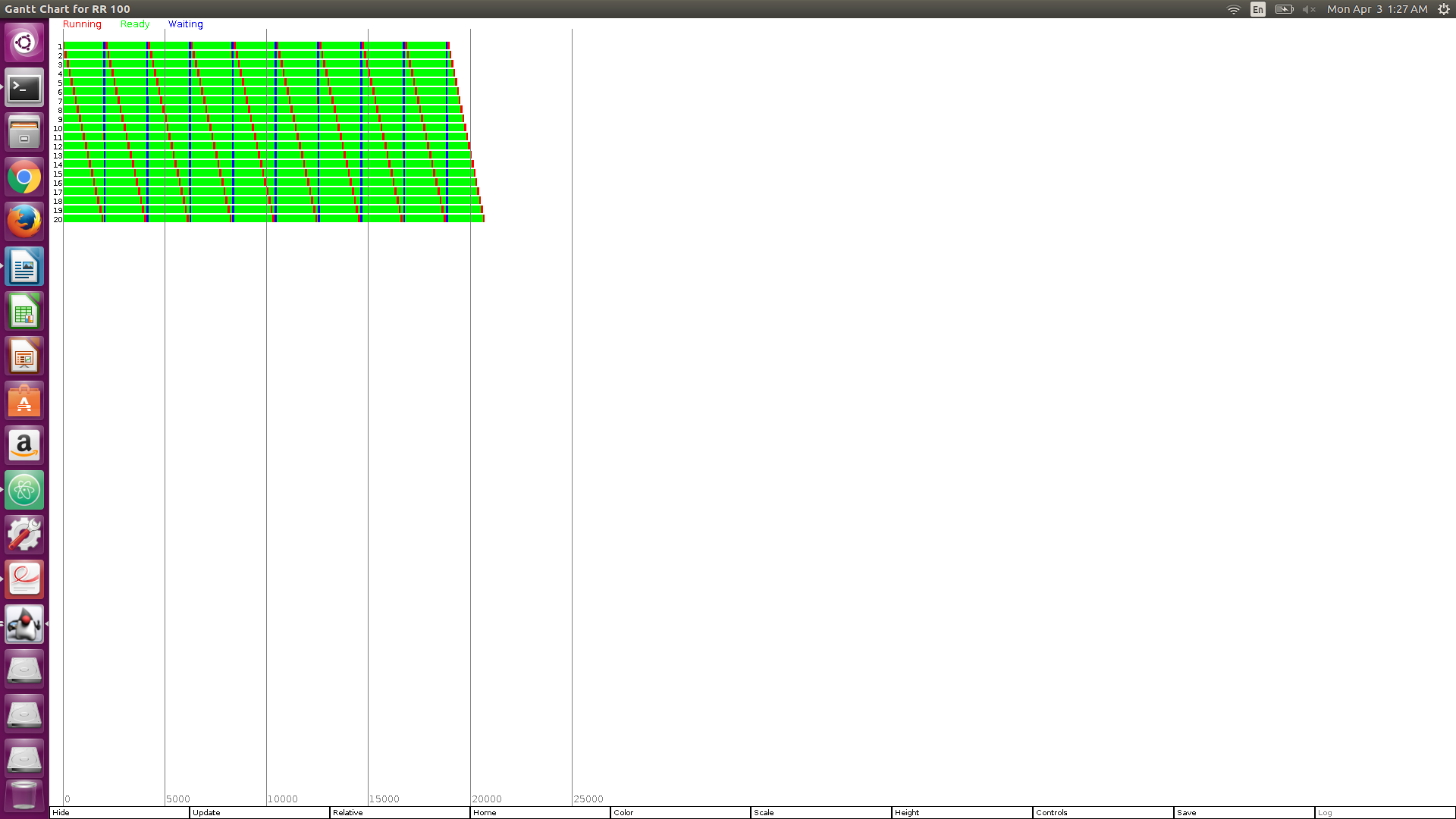
basepriority 1.0

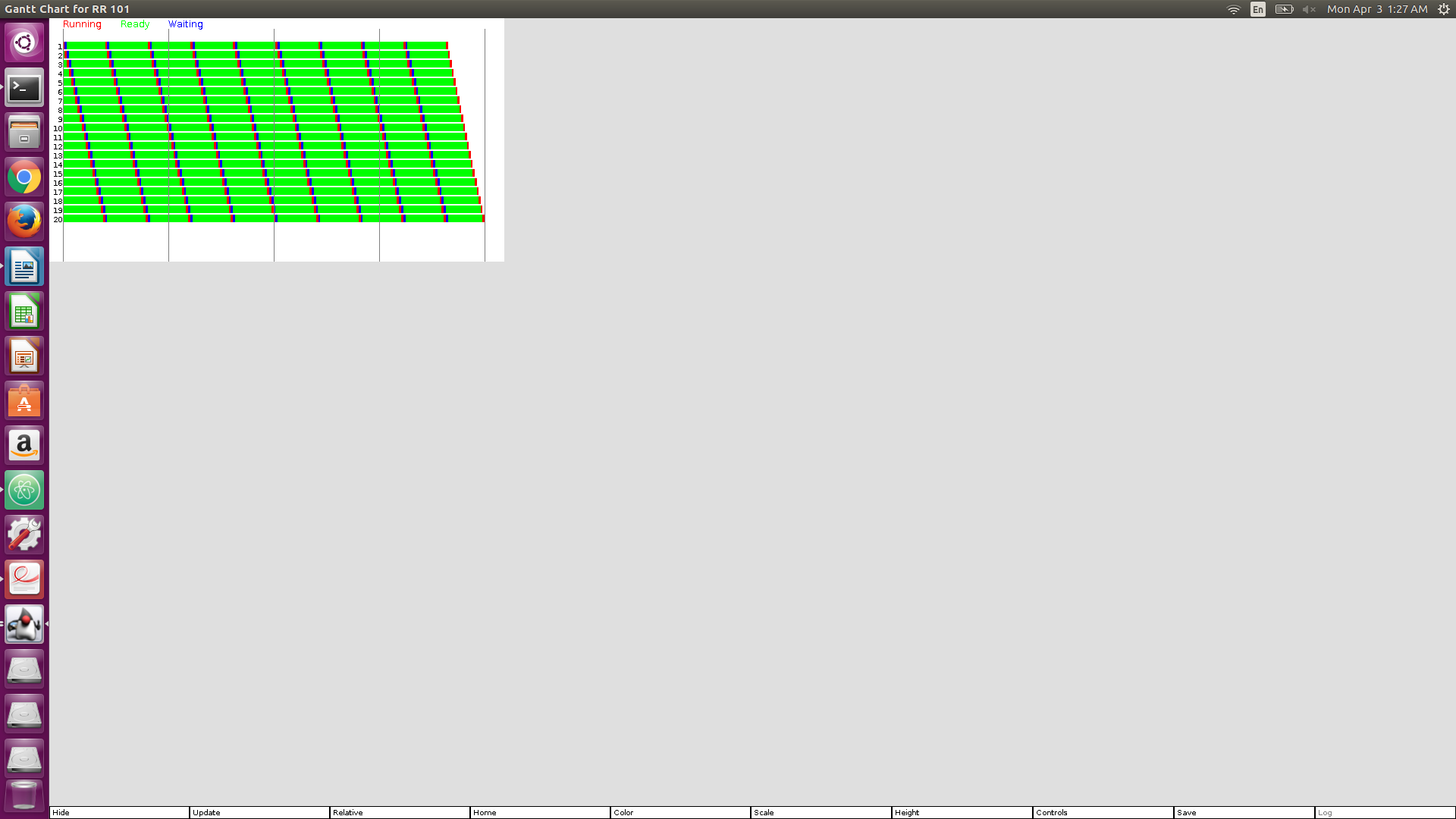












The round robin algor ithm works by FCFS but in tandem with a quantum that is passed along with it’s declaration. The role of the quantum is to give a process an assigned amount of time to use the cpu.