Assignment 2

Satvik Chauhan (Y9521),Pankaj More (Y9402)

Simulation

The Simulation part of the assignment was written in **Haskell** (pure functional programming language).Some of the implementation details are given below:

- We have used **State Monads** to simulate the machine.
- A virtual machine is implemented which requires a scheduler to run.
- The scheduler function requires the readyQueue and returns the pair (selected process,Alloted Burst time).
- The Alloted burst time is same as the next CPU burst for all the schedulers except round robin.
- The time quanta taken for round robin in 6 by default. It can be varied on users choice.
- The parameters to generate the initial data are varied like heavy CPU bound processes or IO bound processes .
- An assumption is made that the process starts with a CPU burst and ends with a CPU burst . So the number of CPU bursts is one more than the number of IO bursts.

We have written a python script to generate the distributions for inter arrival times , priority , IO and CPU bursts

- The most popular and fast python library called NumPy was used to generate these distributions.
- The value of λ for exponential and poisson distributions is taken to be 10. IO Bursts are taken from uniform distribution in the range 0 to 9. Besides, the uniform distribution for generating priorities is from the range 0 to 9. Number of processes is N = 20. Number of cpu bursts is 500 and io burst is 499.

We have drawn charts also using the **haskell chart library** . We have shown the Bar graph comparison of average response , turnaround and waiting times for all the standard algorithms.To run use the following steps .

- python *seed_data.py*
- runhaskell *process.hs*

- runhaskell *drawBar.hs*
- runhaskell *drawBarP.hs*
- runhaskell *drawChart.hs*

Then we have shown the comparison of average response .waiting and turnaround times for three priority categories

NACH OS

- Our system can run upto 16 threads at the same time without crashing.Thereafter the behaviour is erratic as documented in the source code.
- The kernel is made preemptive by using timer interrupts every 100 ticks. Every 100 ticks , Alarm::CallBack(), our timer interrupt handler is called.
- The main thread must have a response time of 0. There after the minimum response time is 10 ms due to context switching. Process with priority 9 is guaranteed to have a response time of 10 ms.
- Our implementation consists of multi level queues , highQueue for priority 9 process , midQueue for process between priority 5 and 8 , and the lower priority processes are put in lowQueue.
- Excluding the time spent in servicing priority 9 processes , the cpu spends 80 percent time in mid queue and 20 percent time in low queue.
- This and rr scheduling guarantees that mid queue processes will have a finite response time.
- The idea of cpu time division between mid and low queues makes the scheduler realistic and saves the low priority processes from starvation.

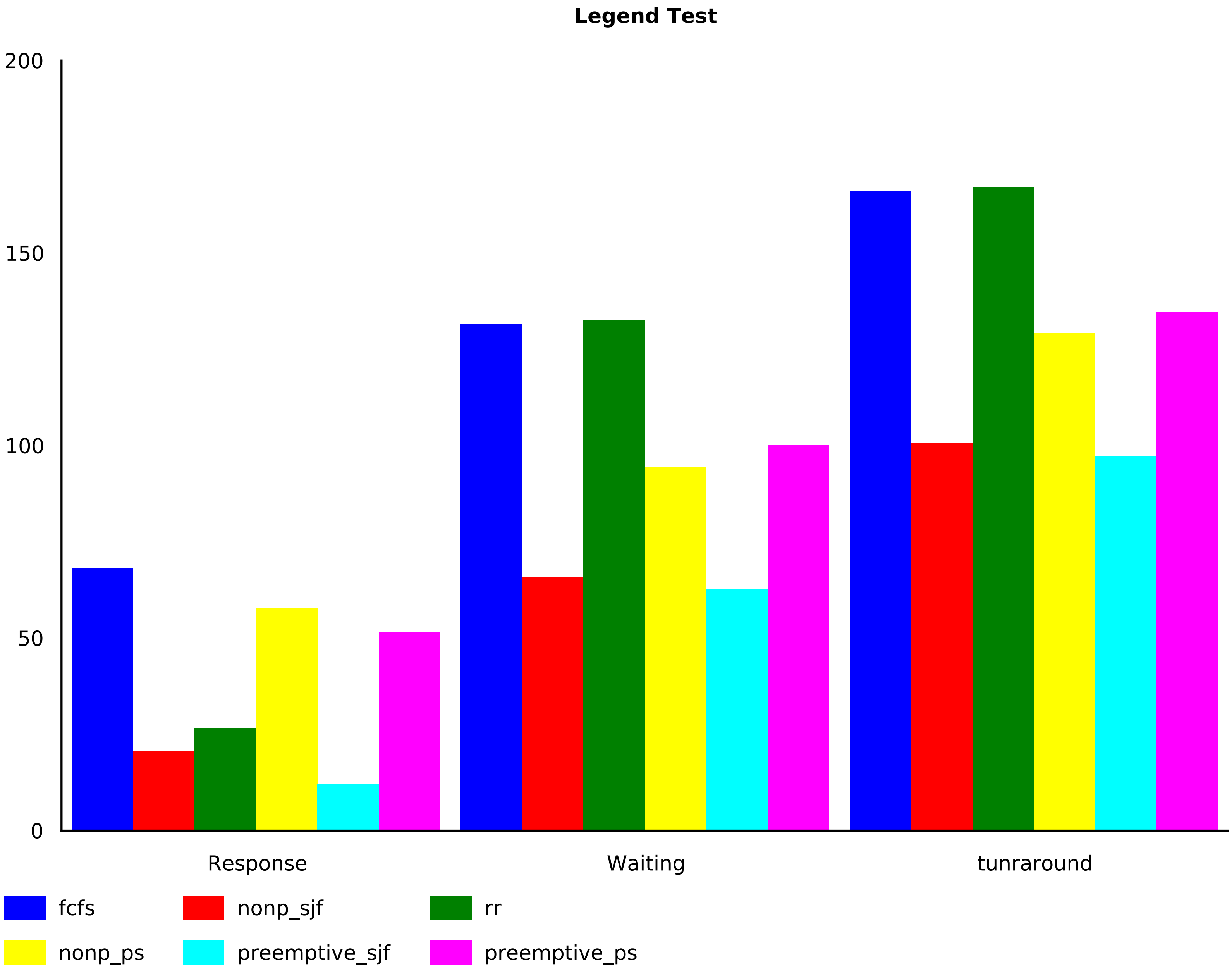

Figure 1: *Averages . Preemptive sjf clearly perform better than others . Round robin has a good response time*

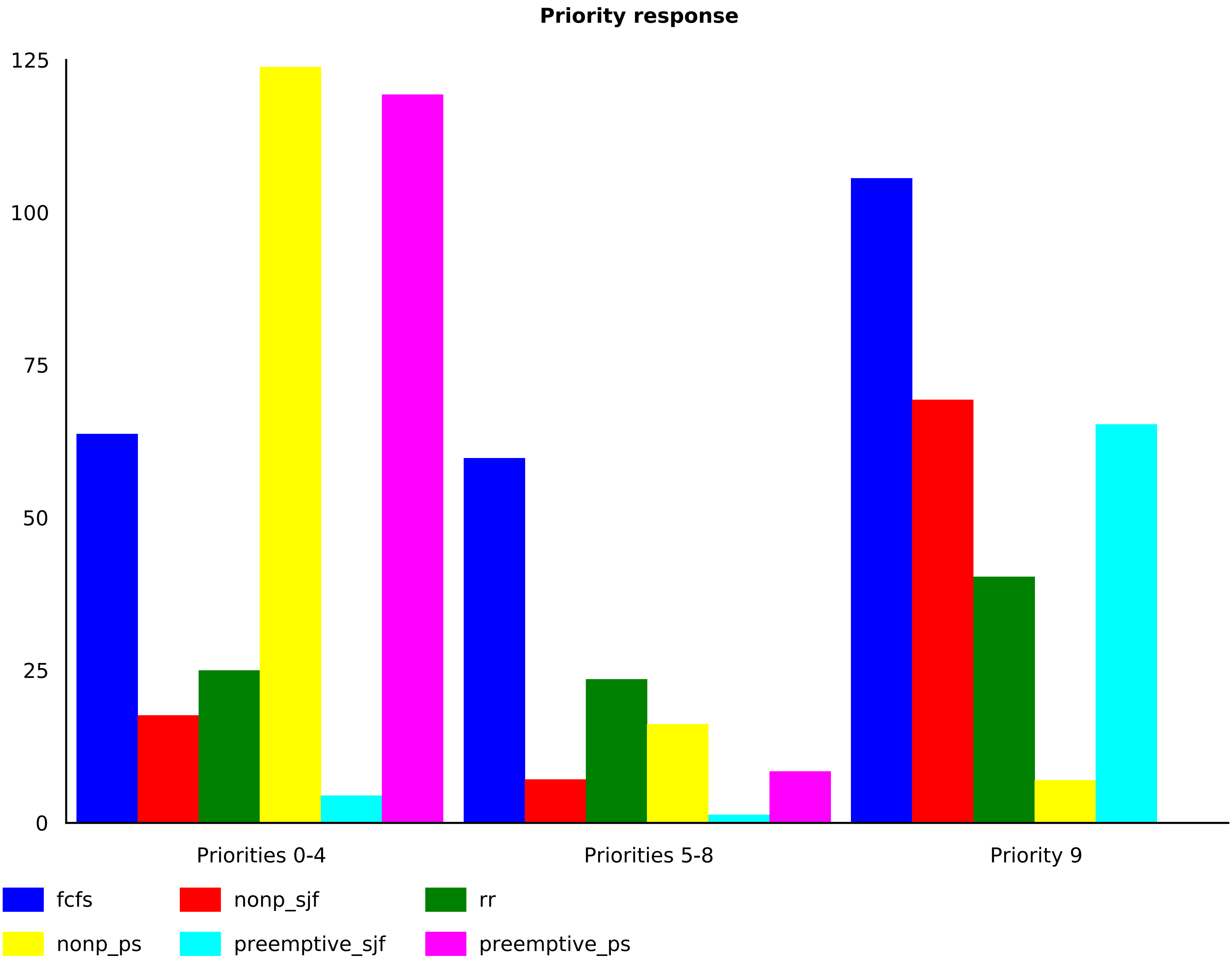

Figure 2: *Average Response time . It can be clearly seen that the average response time for priority 9 is zero for preemptive priority scheduling*

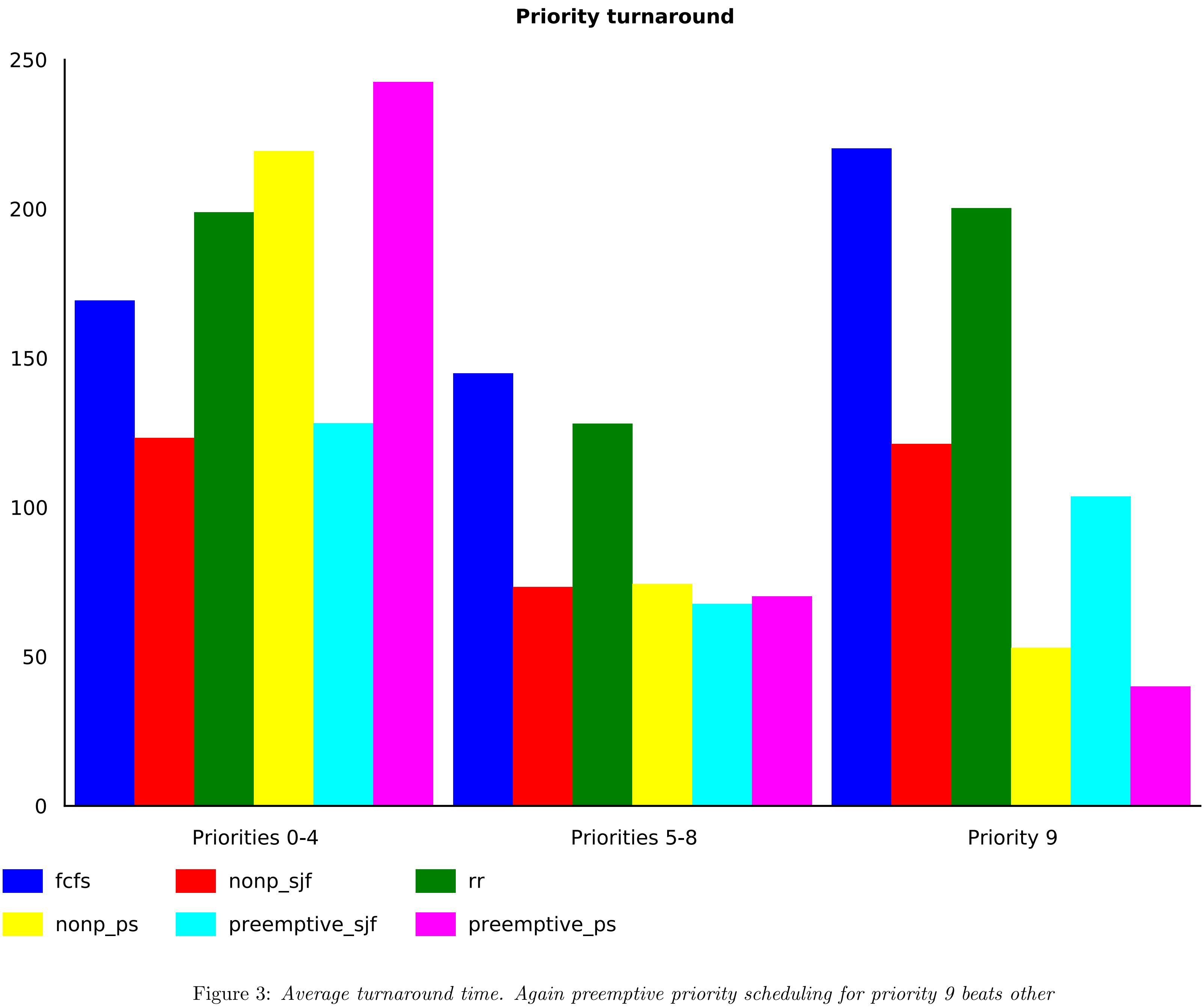

Figure 3: *Average turnaround time. Again preemptive priority scheduling for priority 9 beats other*

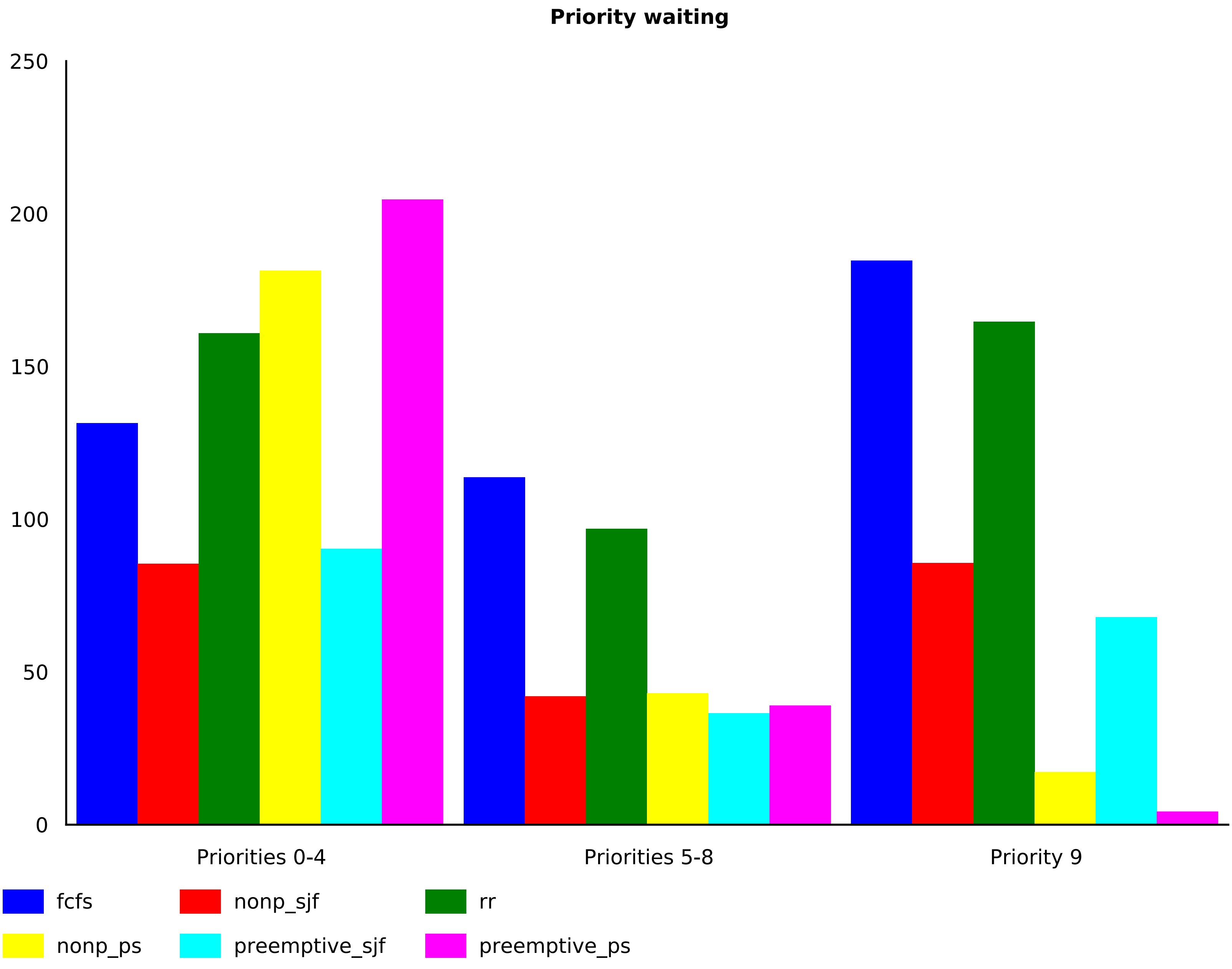

Figure 4: *Average waiting time*