IMPLEMENTATION OF VARIOUS SCHEDULING ALGORITHM

---

1. **ROUNDROBIN:** This scheduling algorithm was already implemented beforehand in the given codebase of XV6. This algorithm iterates through the list of processes (Proc in our case) and goes on running every RUNNABLE process for 1 tick timeslice. After each 1 tick timeslice the control is given back to CPU by calling yield function , which basically restores the context of CPU and then gives back the control to scheduler. Then the scheduler schedules the next process in list to run for next 1 tick timeslice. This loop keeps on going forever in scheduler function.

   1.1 **IMPLEMENTATION:** To implement this, there runs a inner loop iterating through the RUNNABLE processes in process list inside an infinite loop. In innerloop if a RUNNABLE process is found then it is set to run and after completion of it's running time , yield function is called to give back control to CPU and to restore the context of CPU. After the yield function returns, then the scheduler continues to schedule next RUNNABLE process for next timeslice inside the outer infinite loop.

2. **FCFS:** This scheduling algorithm works on the simple rule of first come and first serve, meaning whichever process gets listed in process list first, will run first and will keep running until it gets over. All processes arriving after the earlier arrived process will wait for the earlier arrived process to get over.

   2.1 **IMPLEMENTATION:** To implement this we iterate through the list of processes once and we pick the process which came

earliest. To compare the arrival time of processes, we check the ctime parameter stored in struct proc for every process. This parameter is initialized as ticks (which is global counter of ticks) in allocproc function in proc.c file. So, whichever process arrives early will hold the lower tick value stored in ctime parameter of it's struct. We iterate through list of processes completely once and we pick the process with lowest ctime value. Once the process is picked then we set it to run.

Also we don't call yield function in usertrap function while implementing FCFS (We remove code which calls yield function when which_dev=2), to disable the preemption of process after clock interrupts.
Once the running process gets over then other process with next lowest ctime value gets chance to run and then it runs completely until it gets over or suspended.

3. **MLFQ:** This scheduling algorithm holds 4 queues of different priorities, where queue 0 holds highest priority and queue 3 holds lowest priority. Whenever a process comes in list then it is added to the highest priority queue. If there exists more than 1 processes in highest most empty priority queue then all processes in that queue run in ROUNDROBIN mechanism. Time slice for ROUNDROBIN mechanism of all queues is as follows:
   queue 0 : 1 tick
   queue 1: 3 ticks
   queue 2: 9 ticks
   queue 3: 15 ticks

   Once a process utilizes the total time slice of a queue then it is pushed to the next lower queue in priority.

Since this can cause starvation of processes, so we do aging to pull up a starving process in next higher queue in priority if that process remained for more than AGE time in a single queue.

3.1. **IMPLEMENTATION:** To implement this we made a queue struct which holds the struct proc * for a process at front , struct proc* for a process at rear , total no. of processes in that particular queue and slicetime for that particular queue.
So this queue is basically a linkedlist of processes. To form the linkedlist of processes , we store struct proc* next and struct proc* prev in sturct proc of each process. So that the process will get linked to the new process by setting it's next to new process and by setting prev of new process to the process itself.

We wrote down two new functions as regular enqueue and dequeue functions for queues, to insert processes in queue and remove processes from queue. Here dequeue function differs a bit, because it iterates through the linkedlist of processes and when it finds the to be removed process in linkedlist then it removes that process.

We have an array of queues of size 4 named as queues, where index of each queue represents it's rank in priority. So whenever a process comes then it is inserted into the queues[0].

We added some codes in usertrap function , so that whenever timer interrupts, then the aging function will get called. In aging function we iterate over all processes which are in any queue already and if any process is present in a particular queue for more than AGE time then we pop it out of queue and we add this process to the next higher queue in priority.

After aging we increment the ticks_used counter of process (which holds the number of ticks for which process has been run till now in that particular process), and we check if the ticks_used exceeds the slicetime of that queue. If it exceeds the slicetime of it's present queue, then we push it to the lower queue in priority.

In scheduler , we first iterate through all processes in process list and if we find any process which is RUNNABLE and not included in any queue, then we add that process to it's assigned queue number. After adding processes to the queue, we check for the non empty queue of most highest priority. Then we select the process at front from that queue , and we set it to run.
If a process has not utilized it's slicetime yet then yield function Is not called. We call yield function only when the slicetime of a process gets over.

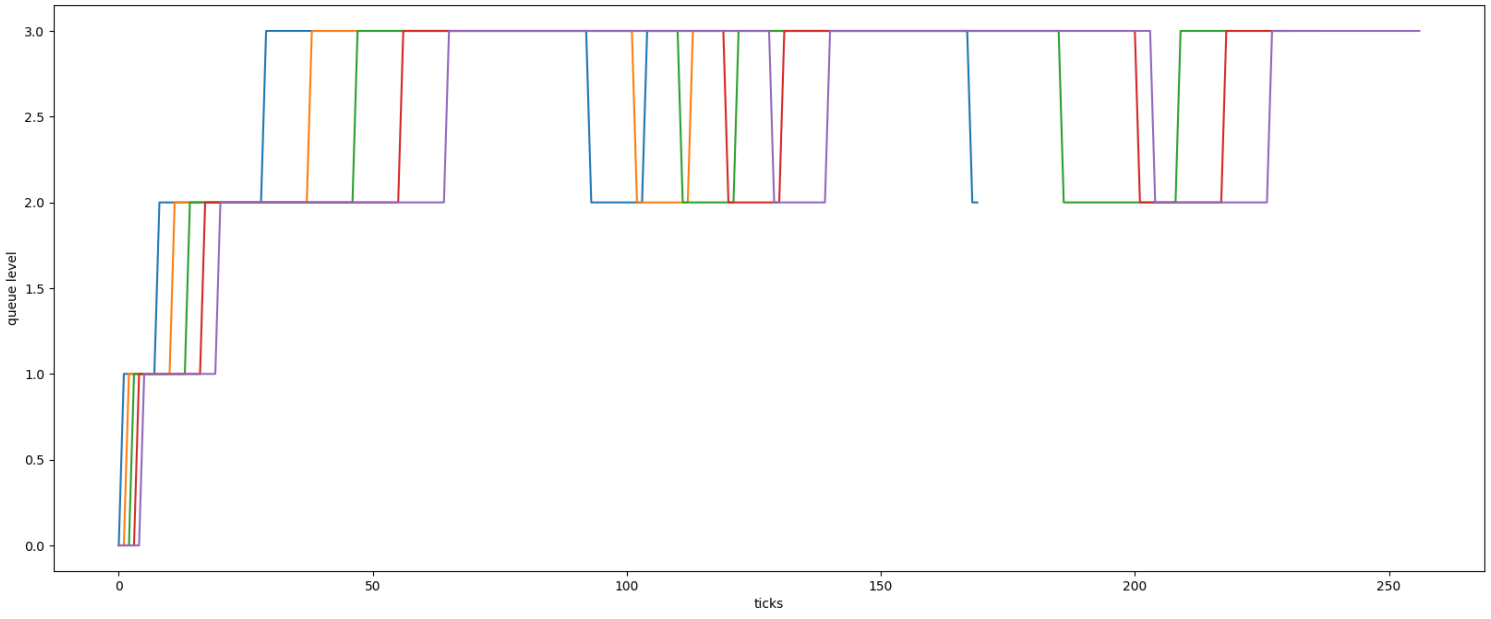
To implement all this , we added queue_num (queue number of the process), ticks_used (for how many ticks the process has been run in the particular queue), queue_enter_time (when did the process enter in queue)
ticks - queue_enter_time for a process gives the total time for which it has been in the present queue. We use this in aging function to compare it to the AGE time.

COMPARISON AMONG THE ABOVE MENTIONED ALGORITHMS

Algorithm	Run time	Wait time
RR	12	150
FCFS	12	125
MLFQ	13	143

We can see that run time and wait time for FCFS is minimum here, because all processes in schedulertest are of same length. So it takes extra overhead time in RR and MLFQ while doing context switch which is not required here because all processes are same and of same length.

MLFQ SCHEDULING ANALYSIS



Here processes are going up when they enter in low priority queue after they use total timeslice of the present queue. They come down when they enter in higher priority queue after they exceed AGE time in present queue ,by aging function.