The following policies are implemented for the xv-6 project:-

1.FCFS(First come first serve):- It is a policy that selects the process with the lowest creation time (creation time refers to the tick number when the process was created). The process will run until it no longer needs CPU time.

Following is the implementation of it:-

1. **Initialization**: First I have initializes a **least_creation_process** pointer to **0**, which will be used to keep track of the process with the least creation time (i.e., the first to arrive).
2. **Run Through Processes**:
    - The code uses a **a** loop to iterate through all processes.
    - It acquires a lock associated with each process to ensure exclusive access to process data and no other data interfers.
3. **Process Selection based on Creation Time**:
    - For each runnable process, the code checks if it's in the **RUNNABLE** state.
    - If **least_creation_process** is not set (i.e., it's the first runnable process encountered), it sets **least_creation_process** to the current process and continues to the next process.
    - If **least_creation_process** is already set and the current process has a lower creation time (**ctime**), it releases the lock on the previous **least_creation_process**, updates **least_creation_process** to the current process, and continues to the next process.
4. **Release Lock**:
    - After checking and potentially updating **least_creation_process**, the code releases the lock associated with the current process.
5. **Execute Selected Process**:
    - If **least_creation_process** is not **0** (i.e., there is a runnable process selected based on FCFS), it sets the state of **least_creation_process** to **RUNNING**.

- The **c->proc** pointer is updated to point to **least_creation_process**, indicating that this process is currently executing on the CPU core.

6. **Context Switch**:
    - The code performs a context switch operation, switching from the current process (if any) to **least_creation_process**.
    - Context switch involves saving the current process's context and restoring the context of **least_creation_process**.
7. **Cleanup and Release Lock**:
    - After the context switch, control returns to this code.
    - The **c->proc** pointer is reset to **0** to indicate that no process is currently running on the CPU core.
    - The lock associated with **least_creation_process** is released.
8. Preemption disable:- Finally the preemption is disabled by commenting out the yield in trap.c.

2.MLFQ:-

- In MLFQ, processes are initially placed in a queue with a certain priority level. The priority level can change based on a process's behavior.
- Lower priority queues allow shorter time slices for execution, while higher priority queues provide longer time slices.
- If a process uses up its entire time slice without blocking or voluntarily yielding the CPU, it is moved to a lower priority queue.
- If a process frequently yields the CPU or becomes I/O bound, it is promoted to a higher priority queue.

Now here is its implementation:-

1. **Decay Priority and Aging**:

- The code first iterates through all processes and checks if they are in the **RUNNABLE** state and whether they've exceeded a time threshold (**tickforage**) since their last execution.
- If a process qualifies, its priority level may be reduced (promoted to a higher priority queue), and it is removed from its current queue.
- The **ticks_present** field is updated to track the time since the last execution.

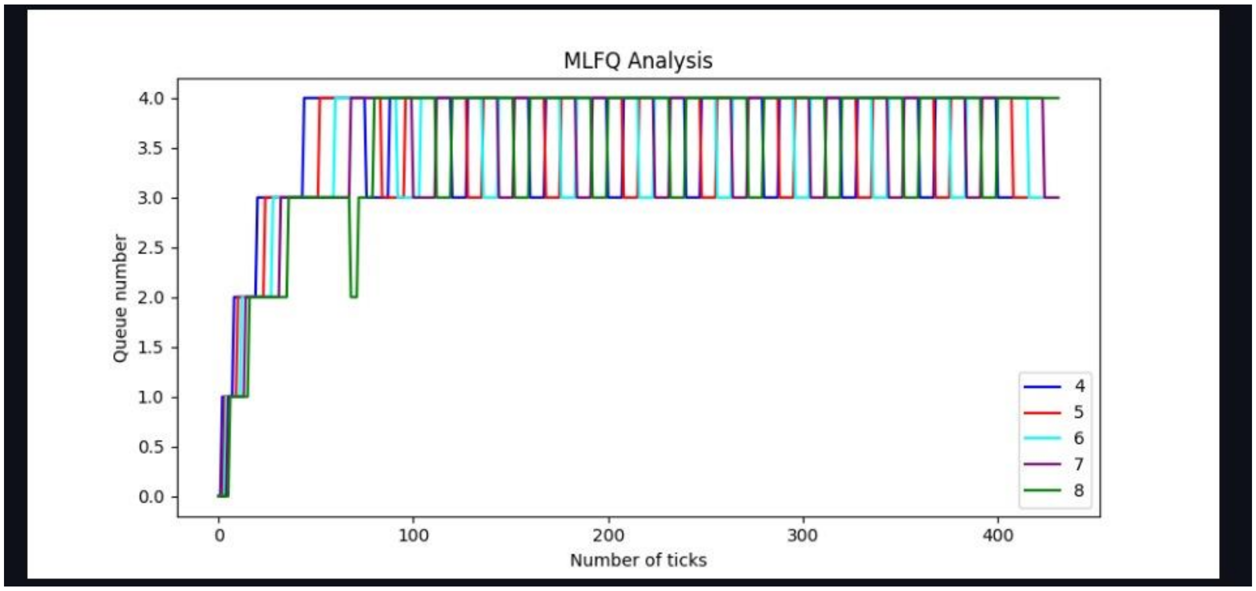
2. **Requeue Runnable Processes**:
   - The code iterates through all processes again, identifying those in the **RUNNABLE** state and not present in any queue.
   - These processes are then placed back into the appropriate queue based on their updated priority level.
3. **Select a Process to Run**:
   - The code goes through the queues, from highest to lowest priority, searching for the highest-priority runnable process.
   - Once a runnable process is found, its state is set to **RUNNING**, and it's marked for execution.
   - The **flag** variable indicates whether a runnable process was found.
4. **Context Switch and Execution**:
   - If a runnable process is identified, it acquires its lock, updates its state, and performs a context switch.
   - The selected process is executed, and the **c->proc** pointer is updated to point to it.
   - The process's **ticks_present** and **change_queue** fields are updated to track its execution history.
   - After execution, the process's lock is released, and control returns to this code.

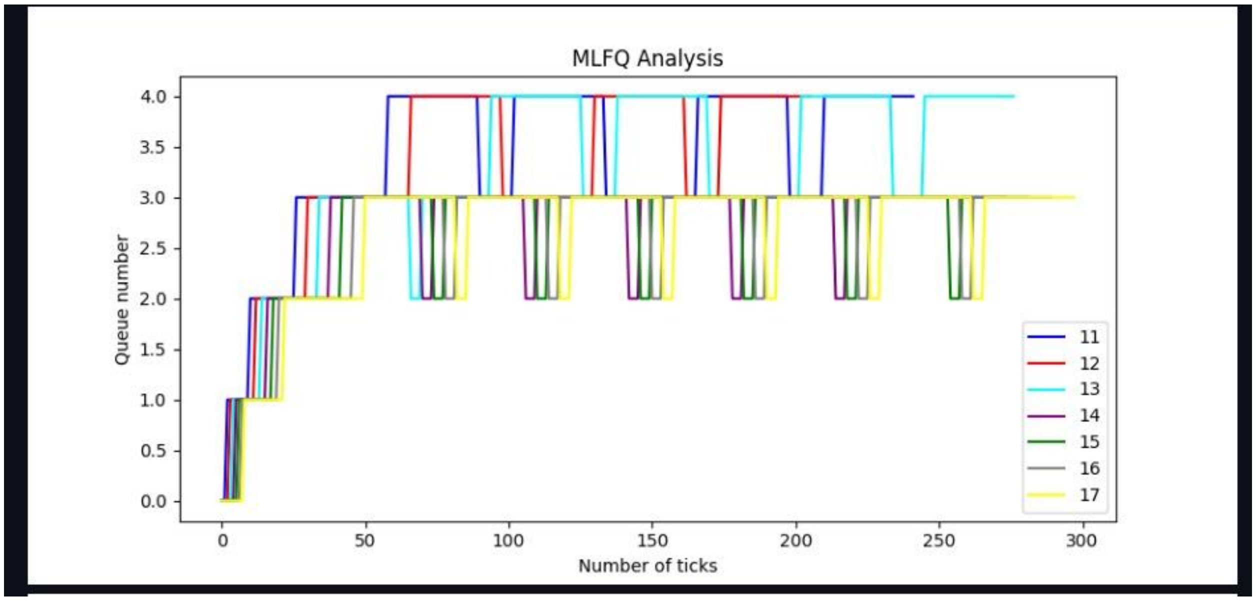
5. Preemption:- We also make changes in the trap.c file. We check if the process is aged. If yes, we preempt the process and take that process to the next queue.

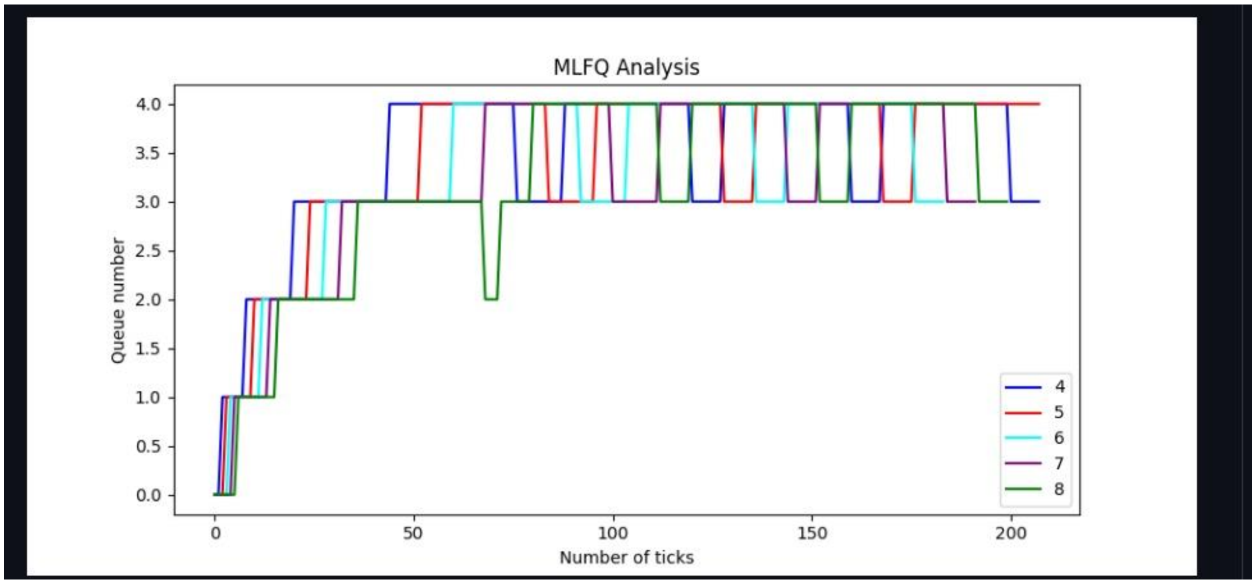
Comparision:-

Policies	Average waiting time	Running time
1.FCFS	9	114
2.MLFQ	10	139

Analysis graph




MLFQ Analysis
Queue number
Number of ticks
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
50
100
150
200
250
300
11
12
13
14
15
16
17


MLFQ Analysis
Queue number
Number of ticks
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
50
100
150
200
4
5
6
7
8