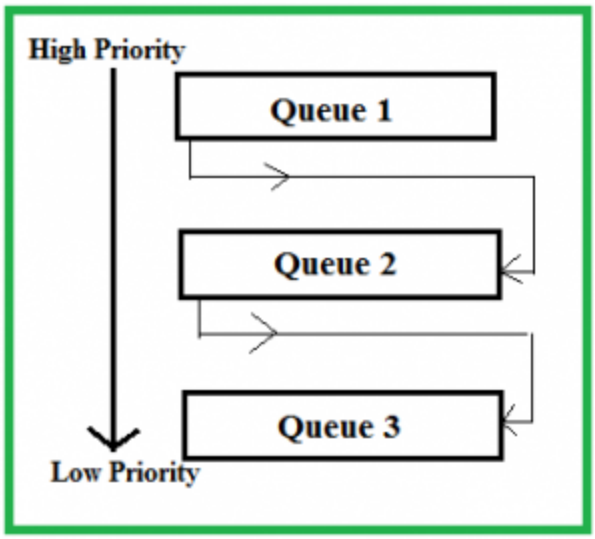
OPERATING SYSTEM-
Multilevel Feedback Queue (MLFQ) Scheduling



Introduction

Multilevel Feedback Queue Scheduling (MLFQ) is a CPU scheduling algorithm that assigns processes to multiple queues based on their priorities and allows them to move between queues dynamically. MLFQ adapts to the behavior of processes, making it more efficient and flexible. Processes start in high-priority queues with shorter time slices. If they exhaust their allocated time without completing, they are moved to lower-priority queues with longer time slices. This feedback mechanism ensures that short processes are handled quickly while longer ones are gradually shifted down, optimizing both response and turnaround times. Additionally, processes waiting too long in low-priority queues may have their priorities boosted to prevent starvation, ensuring fairness.

MLFQ employs features like dynamic priority adjustments, preemption, and time slicing to manage CPU allocation effectively. While this algorithm is highly flexible and prevents starvation, it introduces complexity and CPU overhead due to frequent priority adjustments and queue management. It is particularly useful when the runtime of processes is unknown, as MLFQ learns from their past behavior to predict their needs. Despite being complex, MLFQ strikes a balance between responsiveness and throughput, making it a preferred choice for modern operating systems where process behavior can vary widely.

Code details

Field Explanations in proc.h:

- **`queue_level`**: Indicates which MLFQ queue the process is currently in. It helps in determining the process's priority.
- **`time_in_queue`**: Tracks how long the process has been in its current queue, helping to decide when to demote or promote the process.
- **`ctime`**: Records the creation time of the process, useful for various scheduling and monitoring tasks.
- **`rtime`**: Accumulates the total time the process has been actively running on the CPU.
- **`etime`**: Marks the end time when the process finishes execution.
- **`iotime`**: Tracks the total time the process has been in the I/O state, which can affect scheduling decisions.
- **`last_run`**: Stores the tick count of the last time the process was run, useful for aging mechanism.

In proc.c:

- **`NQUEUES`**: Defines the total number of priority queues in the MLFQ scheduler. Each queue has different priority levels.
- **`BOOST_TICKS`**: Specifies the number of system ticks after which all processes are boosted to the highest priority queue to prevent starvation.
- **`queues[NQUEUES][NPROC]`**: A two-dimensional array storing pointers to processes, where each sub-array represents a priority queue.
- **`proc_in_queue[NQUEUES]`**: An array that maintains the count of processes in each queue, useful for managing the scheduling process.
- **`time_slices[NQUEUES]`**: Specifies the maximum time a process can run in each queue before it is preempted and potentially moved to a different queue.
- **`AGING_FACTOR`**: Adjusts the wait threshold based on queue level to ensure that processes in lower priority queues are gradually promoted if they wait long enough.
- **`BASE_WAIT_THRESHOLD`**: The minimum wait time before considering a process for promotion, used in combination with the aging factor to manage queue promotions.

The enqueue and dequeue functions:

The enqueue function adds a process to a specified priority queue and resets its counter, while the dequeue function removes a process from a queue by shifting the remaining processes to maintain queue order. This system ensures efficient management of processes across different priority levels.

Example of dequeue:

Initial State:

Queue Level	**Processes (`queues[level][i]`)**	**Process Count (`proc_in_queue[level]`)**
0	p1, p2, **p3**, p4	4
1	p5, p6	2

`dequeue(p3, 0)`:

1. Loop finds p3 in `queues[0]`.
2. Replaces p3 with p4 (the last process in the queue at level 0).
3. Decreases `proc_in_queue[0]` from 4 to 3.

Final State:

Queue Level	**Processes (`queues[level][i]`)**	**Process Count (`proc_in_queue[level]`)**
0	p1, p2, p4	3
1	p5, p6	2

In allocproc():

In the `allocproc` function, modifications have been made to initialize MLFQ-related attributes for a newly allocated process. The process is set to start in the highest priority queue (`queue_level = 0`), with its `time_in_queue` and `counter` reset to 0, ensuring a fresh start in the scheduler. The creation time (`ctime`) is recorded as the current tick, while `etime` (end time) and `rtime` (runtime) are initialized to 0. Additionally, `last_run` is set to the current tick, preparing the process for accurate tracking and aging within the MLFQ system. These changes ensure that each process begins with a consistent and well-defined state, optimizing scheduling behavior.

In scheduler():

The `scheduler` function is the core component of the multi-level feedback queue (MLFQ) scheduling system in the xv6 operating system. It continuously selects and executes processes based on their priority and runtime behavior, ensuring fair and efficient CPU allocation.

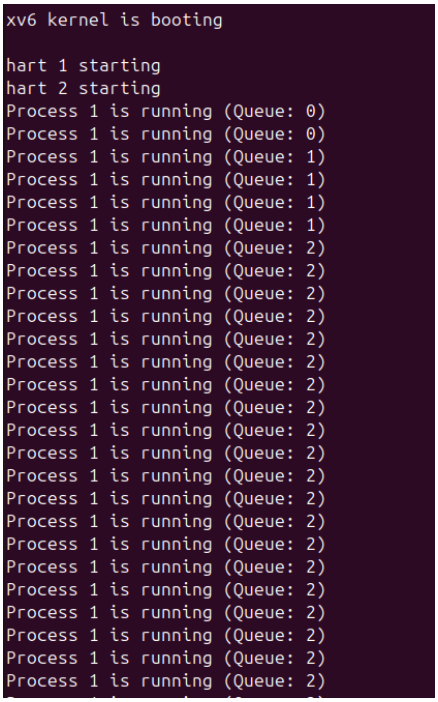
Changes in `scheduler`:

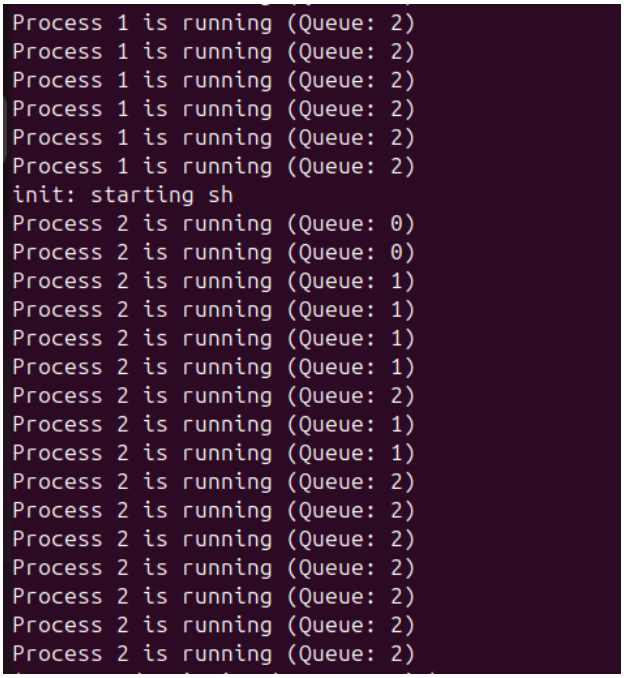
1. **Priority Boost Mechanism:**
    - `if ((ticks - last_boost) >= BOOST_TICKS)`: Checks if enough time has passed since the last boost.
    - Loops through all processes, promoting runnable ones to the highest priority queue to prevent starvation.
    - `dequeue` and `enqueue` functions are used to move processes between queues.
2. **Queue Iteration:**
    - `for (int q = 0; q < NQUEUES; q++)`: Iterates over each priority queue.
    - `for (int i = 0; i < proc_in_queue[q]; i++)`: Iterates through processes in the current queue.
3. **Process Selection and Execution:**
    - Checks if the process is `RUNNABLE` and switches its state to `RUNNING`.
    - Increments the process runtime (`rtime`) and logs the execution.
    - Performs a context switch using `swtch`.
4. **Aging and Promotion:**
    - Calculates the `wait_time` to determine if the process should be promoted based on its queue level and aging factor.

- Uses dequeue and enqueue to move the process to a higher priority queue if necessary.

5. **Time Slice Management:**
   - Tracks how long a process has been in its current queue (`time_in_queue`).
   - Moves the process to a lower priority queue or reinserts it in the same queue if the maximum time slice for that queue is exceeded.
6. **Sleeping Processes:**
   - Increments `iotime` for processes in the `SLEEPING` state, indicating time spent waiting for I/O.

Output snippets:





```
$ sha256 -s hi
Process 2 is running (Queue: 2)
Process 13 is running (Queue: 0)
argc = 3
argv[0] = sha2Process 13 is running (Queue: 0)
56
argv[1] = -s
argv[2] = hi
Ticks before: 889
FiProcess 13 is running (Queue: 0)
nal hash: 8F434346648F6B96DF89DDA91C5176B10A6D83961DD3C1AC88B59B2DC327AProcess 13 is running (Queue: 0)
A4Ticks: after  891
Final hash: %02x%02x%02x%02x%02x%02x%02x%02x%02x%0Process 13 is running (Queue: 0)
2x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02Process 13 is running (Queue: 0)
x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x
Process 2 is running (Queue: 2)
Process 2 is running (Queue: 2)
$
```

```
$
Process 2 is running (Queue: 2)
Process 17 is running (Queue: 0)
Process 2 is running (Queue: 2)
$
Process 2 is running (Queue: 2)
Process 18 is running (Queue: 0)
Process 18 is running (Queue: 0)
Process 2 is running (Queue: 2)
$
Process 2 is running (Queue: 2)
Process 19 is running (Queue: 0)
Process 2 is running (Queue: 2)
$
Process 2 is running (Queue: 2)
Process 20 is running (Queue: 0)
Process 2 is running (Queue: 2)
$


Process 2 is running (Queue: 2)
Process 21 is running (Queue: 0)
Process 2 is running (Queue: 2)
$ Process 22 is running (Queue: 0)
Process 2 is running (Queue: 2)
$ Process 23 is running (Queue: 0)
Process 2 is running (Queue: 2)
Process 2 is running (Queue: 2)
$
```

```


Process 2 is running (Queue: 2)
Process 21 is running (Queue: 0)
Process 2 is running (Queue: 2)
$ Process 22 is running (Queue: 0)
Process 2 is running (Queue: 2)
$ Process 23 is running (Queue: 0)
Process 2 is running (Queue: 2)
Process 2 is running (Queue: 2)
$ Executed priority boost at tick: 1200

Process 2 is running (Queue: 2)
Process 24 is running (Queue: 0)
Process 2 is running (Queue: 2)
$ Process 2 is running (Queue: 2)
```