Computer Systems Engineering

Xv6-Scheduling

Performance comparision between different schedulers-Sachin Kumar Danisetty (2018101108)

Introduction

This is an experimental result comparing four algorithms namely Round Robin(which was given by default), First Come First Serve(FCFS), Priority-Based Scheduling(PBS), Multi-Level Feedback Queue(MLFQ).

Implemented Algorithms

Round Robin

The default algorithm implemented in xv6 it's one of the simplest (with FCFS) and relies on the Round-Robin policy, basically, it loops through all the process which are available to run (market with the RUNNABLE) state and gives access to CPU at each one of them one at a time. To schedule processes fairly, a round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum (its allowance of CPU time), and interrupting the job if it is not completed by then. The job is resumed next time a time slot is assigned to that process. If the process terminates or changes its state to waiting during its attributed time quantum, the scheduler selects the first process in the ready queue to execute. In the absence of time-sharing, or if the quanta were large relative to the sizes of the jobs, a process that produced large jobs would be favored over other processes. Round-robin scheduling is simple, easy to implement, and starvation-free.

To enable it and see how Round Robin works use this command when compiling xv6:

```
$ make qemu SCHEDFLAG=RRB
```

First Come First Serve(FCFS)

First come first served (FCFS), is the simplest scheduling algorithm. FCFS simply queues processes in the order that they arrive in the ready queue. The scheduling overhead due to using this policy is minimal since context switches only occur upon process termination, and no reorganization of the process queue is required. Throughput can be low, because long processes can be holding CPU, waiting for the short processes for a long time, so short processes that are in a queue are penalized over the longer ones (known as convoy effect). By using this policy we have no starvation because each process gets a chance to be executed after a definite time. Turnaround time, waiting time and response time depends on the order of their arrival and can be high for the same reasons above. There isn't prioritization, so using this policy we cannot force certain processes to be completed first which means that this system has trouble meeting process deadlines. The lack of prioritization means that as long as every process eventually completes, there is no starvation. In an environment where some processes might not complete, there can be starvation since the processes that come next to the one which might not complete are never executed.

To enable it and see how FCFS works use this command when compiling xv6:

```
$ make qemu SCHEDFLAG=FCFS
```

Priority Based Scheduling(PBS)

The priority scheduling algorithm (SML) represents a preemptive policy that executes processes based on their priority. The scheduling policy first selects the runnable process with the lowest value of priority and executes it, after that it finds the one with the second-lowest value of priority and executes it and so on until we have finished all the processes. This scheduling policy allows the user to mark some processes which we want to be completed first in a simple but fast way. Priority range in this algorithm is 1-20 (default is 10) where we give priority equals 1 for the processes which we want to be completed first.

The following system call will change the priority queue of the process with a specific pid process:

```
int chpr(int pid, int priority)
```

In this case, priority is a number between 1 and 100 which represents the new process priority.

To enable it and see how PRIORITY BASED SCHEDULING works use this command when compiling xv6:

```
$ make qemu SCHEDFLAG=PBS
```

Multi-Level Feedback Scheduling(MLFQ)

MLFQ scheduler allows processes to move between different priority queues based

on their behavior and CPU bursts. If a process uses too much CPU time, it is pushed

to a lower priority queue, leaving I/O bound and interactive processes for higher

priority queues. Also, to prevent starvation, it implements aging.

To enable it and see how MLFQ works use this command when compiling xv6:

```
$ make qemu SCHEDFLAG=MLFQ
```

Benchmark Program

I wrote a benchmark/test program named ftest.c which can be executed as a shell command. This is used to compare the performances of the scheduling algorithms and demonstrate using that program.

RRB

	ctime	etime	rtime	wtime
child-1	5650	6550	447	452
child-2	5648	6560	458	454
child-3	5648	6632	519	465
child-4	5651	6663	552	460
parent	5646	6663	3	1

FCFS

	ctime	**etime**	**rtime**	**rtime**
child-1	**2095**	2551	455	1
child-2	**2095**	2599	504	0
child-3	**2096**	3027	490	504
child-4	**2096**	3091	504	504

parent	2087	3091	3	0

PBS

	ctime	**etime**	**ritme**	**wtime**
child-1	**2095**	**2551**	**455**	**1**
child-2	**2095**	**2599**	**504**	**0**
child-3	**2096**	**3027**	**490**	**455**
child-4	**2096**	**3091**	**504**	**504**
parent	**2087**	**3091**	**3**	**0**