OS Course Project - Report

CFS Scheduler for xv6

G Shyam Sundar - CS18B015
N K Sagar Reddy - CS18B029

Aim of the Project

To code a CFS scheduler(inspired from Linux) for the xv6 OS which runs on risc-v.

About Xv6

Xv6 is a virtual OS which uses an x86 multiprocessor and RISC-V system.
Its default process scheduler uses the round robin scheduling algorithm,
which although works well (low average wait time and faster response time),
it has its drawbacks (high average wait time and increased context switches)

About CFS

CFS, (Completely Fair Scheduler) is the default process scheduler used by the Linux Kernel.
It uses a priority based scheduling algorithm with dynamic priorities, and dynamic timeslices.
Each process is given a vruntime (virtual runtime) and also, lower vruntime corresponds to higher priority.
CFS uses Red-black trees to store its processes(nodes) with their vruntime being their keys.
At each context switch the node/process with minimum vruntime is the next task to run.

Red-Black Trees

Red-black trees are self balancing binary search trees.
Since all operations are O(H) for Binary search trees,(O(N) in the worst case) ,
self balancing trees try to minimize the height of the tree using rotations.
These self balancing trees maintain their Height approximately O(logN), so all operations reduce to O(log N).
Each node is either Red or Black in the Red-Black tree.
Sometimes after insert,just recoloring is sufficient instead of rotating the tree everytime after insertion.
As a process scheduler uses insert and delete operations more compared to search, a Red-Black tree is a more optimal Data Structure when compared to AVL tree(although AVL tree is more balanced,they may cause more rotations during insert and delete).

Files modified

proc.c , proc.h , trap.c in the Kernel folder (xv6-riscv/kernel)

Implementation

Modifications in proc.h

In proc.h , the struct proc was modified by adding the following attributes:

1) struct proc* P(Parent) , R (Right child) , L (Left child)
   (used for RB Tree implementation)
2) int c // color of the node
3) uint64 v_runtime
4) uint64 startTime
5) Int timeslice
6) Int extraslice
7) Uint64 exectime
8) Uint64 SleepStartTime
9) Uint64 SleepTime
10) Int priority

Modifications in trap.c

usertrap()

This is done for implementing timeslices. If a timer interrupt occurs then the process has executed for an fixed interval where extraslice stores the how many times more this process should be executed for that interval. I.e if the if the process time slice is 9*interval extraslice is 9 and the process never yields until it becomes 0. Extraslice is initialised in proc.c Scheduler Function.

Modifications in proc.c

Implementation of RB tree

Both Insert() and delete (delete_by_val()) have been implemented using helper functions like Search() , findmin() , leftrotate() , rightrotate() , sibling() , red_child_exists() , inorder_successor() , replacable_node() , search_by_val() , double_black() and delete_proc()

userinit()

We set up the first user process and add it (root) to the Red-Black tree.

fork()

Each new process created in fork() will be initialised and inserted into the Red-Black tree.

scheduler()

We obtain the process with the least v_runtime (left most node) from the RB tree in O(log N), Delete it from the tree, set its starttime (to r_time) and then release the lock after execution is done. Finding the time slice based on number of processes and initialising Starttime and extraslice for the process.

yield()

We change the state of the RUNNING process to RUNNABLE and insert it into the tree.
For setting dynamic priority values of exectime and Sleeptime are used where Exectime is set with parallel to v_runtime but v_runtime is incremented with a weight and SleepTime is calculated in wakeup(). The weight multiplied in v_runtime calculation is given by sched_prio_to_weight array which is globally defined.

```
priority +=20 - ((SleepTime)/(exectime+SleepTime))*50
```

sleep()

When the process reaches SLEEPING state from RUNNABLE then v_runtime should be incremented with its execution time in that timeslice. Process SleepStartTime should be initialised here. Same dynamic priority calculation as in yield.

wakeup()

We change the state of the SLEEPING process to RUNNABLE and insert it into the tree.
Now the process is woken up So its total SleepTime should be incremented by its current sleep time.

kill()

If the process is in SLEEPING , change it to RUNNABLE and insert it into the tree.

Source Code

1. proc.c - link
2. proc.h - link
3. trap.c - link
4. Xv6-riscv folder (zipped version) - link

Presentation

You can find the video presentation at this link - link

Contributions

RB Tree implementation - Insert and related helper functions - Shyam
RB Tree implementation - Delete and related helper functions - Sagar
Proc.h - Shyam
Trap.c - Sagar
Proc.c - userinit() , fork() , scheduler() , yield() - Shyam
Proc.c - sleep() , wakeup() , kill() - Sagar