Iran University of Science and Technology
Operating Systems (Autumn 2023) - Dr. Reza Entezari-Maleki

Project 1: Completely Fair Scheduler on XV6

Designer
Aryan Abdollahi Sabet Nejad
aryan_abdollahi@comp.iust.ac.ir
aryansabetnejad@gmail.com

1 Objectives

The primary objectives of this project are:

Familiarizing with a Real Scheduler To gain practical experience and familiarity with a real-world scheduler used in operating systems, and understand its design and impact on system performance and resource allocation.

Team Collaboration To foster teamwork and collaboration among project members, simulating real-world software development scenarios where multiple contributors work on a shared codebase.

Comprehensive Testing To develop and execute some test programs that ensure the reliability and performance of the implemented scheduling algorithm.

Resume Enhancement To create a well-documented scheduling project that can be presented as a sample project on a resume. This includes detailed project documentation, code comments, and a clear explanation of the implemented scheduling algorithm.

2 Introduction

2.1 Background

The Linux Completely Fair Scheduler (CFS) is a widely used process scheduler in the Linux kernel, known for its focus on providing fair CPU time allocation to processes and tasks. It is an essential component of modern Linux systems, ensuring that system resources are allocated efficiently and fairly among running processes. The CFS has been designed to prioritize fairness, responsiveness, and low latency in the scheduling of tasks, making it a crucial part of Linux's success as an operating system.

2.2 Significance

Understanding and working with the CFS scheduler offers a unique opportunity to dive deep into the core of the Linux kernel. This project aims to explore, analyze, and gain insights into the CFS scheduler, its algorithms, and its impact on system performance. By investigating this essential component, you will not only broaden your knowledge of operating systems but also have a practical understanding of a scheduler used in real-world scenarios.

3 Installation of XV6 & Git

The following procedure has been tested on Debian, and should also work on Debian-based distributions such as Ubuntu or Mint. It is strongly suggested that you create an isolated workspace using tools like VirtualBox and a clean Ubuntu. You can also install it on your daily driver if you don't mind.

First, make sure to have all the dependencies :

```
user@user:~$ sudo apt-get update && sudo apt-get install
    --yes build-essential git qemu-system-x86
```

After successfully installing the required programs, clone the Github repository of xv6 source code:

```
user@user:~$ git clone https://github.com/mit-pdos/
    xv6-public
```

Now just compile the kernel:

```
user@user:~$ make qemu
```

Each team is required to fork the repository to their respective GitHub accounts rather than cloning it. Both team members are expected to actively collaborate on GitHub. The individual contribution of each team member to the shared repository significantly influences their respective scores. As an efficient means of collaboration, it is strongly recommended to install GitHub Desktop or Git CLI and utilize it for this objective.

4 Implementation Details

1. **Implement a Red-Black Tree data structure:** The heart of CFS is a Red-Black Tree, which maintains a balanced structure of processes based on their virtual runtimes. Your task is to implement this data structure in C for the XV6 operating system.
2. **Implement process insertion and retrieval functions:** Develop functions that allow you to insert processes into the Red-Black Tree while maintaining the Red-Black Tree properties. You should also have functions to retrieve processes with the smallest virtual runtime.
3. **Calculate process weights:** Calculate each process's weight based on their "nice" value. The formula for weight calculation is provided:

$$timeslice_k = \frac{weight_k}{\sum_{i=0}^{n-1} weight_i} \times sched\_latency \tag{1}$$

$$vruntime_i = vruntime_i + \frac{weight_0}{weight_i} \times runtime_i \tag{2}$$

4. **Implement fairness and preemption logic:** Ensure that the scheduler distributes CPU resources fairly among processes based on their weights and scheduling latencies. Implement the logic to determine if a process should be preempted or allowed to continue running.
5. **Test and integrate:** Test your CFS implementation with a variety of scenarios and process workloads to ensure that it operates correctly and fairly allocates CPU resources. Integrate your scheduler into the XV6 operating system.

While platforms like YouTube and AI-powered assistance are valuable tools, it's crucial to emphasize that the depth of knowledge gained through this project will be a critical factor. The project encourages hands-on learning and practical experience, allowing you to deepen your understanding of the subject matter beyond what external resources alone can provide."

By emphasizing the importance of hands-on learning and practical experience, you convey the significance of the project in acquiring a deep understanding of the subject matter.

5 Hints

Most of the code for the scheduler is quite localized and can be found in *proc.c*; the associated header file, *proc.h* is also quite useful to examine. To change the scheduler, not much needs to be done to the other files; study its control flow and then try some small changes. You'll also need to figure out how to use traps on *trap.c*.

6 Recommended resources

It is highly recommended to dive into the following resources to enhance your understanding of the subject matter:

Operating Systems: Three Easy Pieces (OSTEP) - Chapter 6 & Chapter 9 Chapter 9 of *Operating Systems: Three Easy Pieces (OSTEP)* is an excellent resource for understanding various scheduling algorithms, including lottery, stride, and the Completely Fair Scheduler (CFS). This chapter is particularly relevant to this project. You can download Chapter 9 of OSTEP from here. also chapter 6 will be useful to understand kernel mode & context switching which can be downloaded from here.

xv6 Book: Timer Trap Section and Scheduling Chapter The xv6 book, authored by the same team that ported xv6 to x86, provides valuable insights into the xv6 operating system. We recommend paying particular attention to the 'Timer Trap' section and the 'Scheduling' chapter within this book. These sections offer in-depth knowledge about scheduling mechanisms in xv6, which aligns with the objectives of this project. You can download the xv6 book from here.

7 Documentation Preparation

7.1 Markdown Documentation

Students are required to provide comprehensive and well-structured Markdown documentation that explains the codebase's functionality and design in their repository. The **README.md** should include:

- Overview: An introduction to the codebase, its purpose, and the scheduling algorithm used.
- Code Explanation about the changed parts using comments on code sections & descriptions.
- Formula Explanation using MathJax provided by markdown itself. more details on here.
- Explanation about Test programs provided & visualization of some of them using the Gantt charts on Mermaid.js. more details on here.

7.2 Submission Documentation

Students are also required to provide documentation about problems faced during the process & anything else that needs to be mentioned in Persian. Make sure that you include your repository in the compressed file beside the documentation.

8 Submission

To submit your work, simply commit all your changes (adding new files as needed) and push your work to your GitHub repository. also, upload a zipped file of the Persian documentation & repository on Quera. Make sure you sign the filename with your student IDs. If you don't do this we won't be able to associate your submission with you!