CSE 331 TERM PROJECT

by

Yahya KOYUNCU

İrem Tuğba SAĞSÖZ

Begüm YILDIRIM

CSE 331 Operating Systems Design

Term Project Report

Yeditepe University

Faculty of Engineering

Department of Computer Engineering

Spring 2023

ABSTRACT

TABLE OF CONTENTS

1. INTRODUCTION

In today's computing landscape, optimizing the performance of operating systems and efficiently managing workloads is of paramount importance. In this context, scheduler algorithms constitute fundamental building blocks of operating systems. This project thoroughly examines our Stride algorithm, specifically designed for comparison with Linux's default scheduler algorithm.

In this project, we compare Linux's default scheduler algorithm with our own stride algorithm according to certain criteria. First of all, we examine in detail the basic features of Completely Fair Scheduler (CFS), Linux's default scheduler algorithm, and the Stride Scheduler algorithms we developed, and determine the advantages and limitations of each algorithm. Then we use specifically designed test cases to test and compare the performance of these algorithms in specific use cases.

This project consists of two key phases that play an important role in complex operating systems. The first stage includes an infrastructure created using system calls in order to establish effective communication between the kernel and user space. At this stage, a system call that performs a simple addition operation is designed to evaluate a basic functionality. This formed the basis of the project and provided a basis for moving into the second phase. After the completion of the first phase, the second phase of the project started. At this stage, a special system call is created that will serve as a flag to be used when communicating between the kernel and user space. This system call provides a more specific and customized infrastructure for the general purpose of the project. In the second stage, the main focus of the project is the "stride algorithm", a unique scheduling algorithm. This algorithm aims to provide effective resource management by organizing and prioritizing workloads. This created algorithm was integrated into the kernel in accordance with the purpose of the project and made available through system calls.

In the design and implementation section of this document, both Linux's default scheduler algorithm and the stride algorithm we designed are mentioned in detail. In the Test and Results section, there are the tests we performed using the two algorithms and the comparison of the results we obtained from these tests, and the error rates we encountered when compared to the default scheduler. The Conclusion section includes the evaluation of the results we obtained as a result of the tests.

2. DESIGN and IMPLEMENTATION

Process management is a crucial aspect of operating systems, and it involves various components, one of which is the scheduler. The scheduler is responsible for determining which process gets to use the CPU and for how long. Operating systems use various scheduling algorithms to manage complex and diverse tasks effectively. Linux supports 3 scheduling policies: SCHED_FIFO, SCHED_RR, and SCHED_OTHER. SCHED_OTHER is the default universal time-sharing scheduler policy used by most processes; SCHED_FIFO and SCHED_RR are intended for special time-critical applications that need precise control over the way in which runnable processes are selected for execution.

Within the Linux default scheduler, each process is assigned a value known as "nice." Nice values determine the dynamic priority of the respective process. Newly created processes have a default nice value of 0. Negative nice values indicate higher priority, while positive values result in lower priority. The process priority is calculated as 20 minus the nice value. If a process experiences I/O bursts, the system decreases the nice value to boost priority. However, if a process solely undergoes CPU bursts, the nice value remains unchanged.

Time slices allocated based on process priority are directly proportional. The system manages these allocations using a counter variable assigned to each process. This counter variable indicates the number of time slices allocated to a process. When these calculated counters are exhausted, the system recalculates the counters for each process, termed an "epoch."

In summary, the Linux scheduling system dynamically adjusts process priorities using nice values, allocates time slices based on priorities, and maintains fairness through periodic recalculations during epochs. This dual approach of time-sharing and real-time algorithms ensures effective multitasking and responsiveness in diverse computing environments.

2.1. LINUX DEFAULT SCHEDULER

2.2. STRIDE SCHEDULER

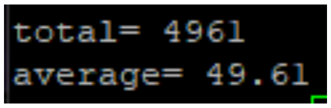
3. TESTS AND RESULTS

3.1. DEFAULT SCHEDULİNG ALGORITHM

3.1.1. AVERAGE CPU USAGE

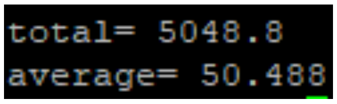
3.1.1.1. TEST CASE 1

In test case 1 we used 2 processes (p1 and p2). Each process has 10 average values. Each value 100 samples.



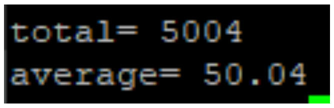
```
total= 4961
average= 49.61
```

Figure 3.1.1.1.1 Average CPU Usage of p1



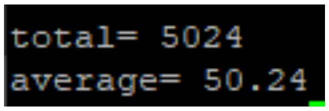
```
total= 5048.8
average= 50.488
```

Figure 3.1.1.1.2 Average CPU Usage of p1



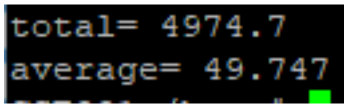
```
total= 5004
average= 50.04
```

Figure 3.1.1.1.3 Average CPU Usage of p1



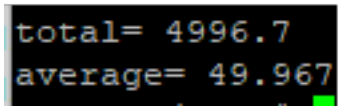
```
total= 5024
average= 50.24
```

Figure 3.1.1.1.4 Average CPU Usage of p1



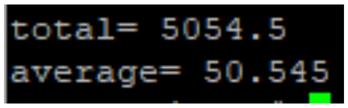
```
total= 4974.7
average= 49.747
```

Figure 3.1.1.1.5 Average CPU Usage of p1



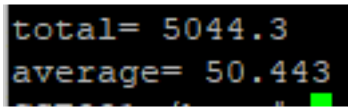
```
total= 4996.7
average= 49.967
```

Figure 3.1.1.1.6 Average CPU Usage of p1



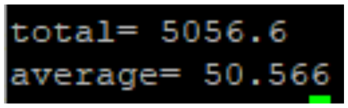
```
total= 5054.5
average= 50.545
```

Figure 3.1.1.1.7 Average CPU Usage of p1



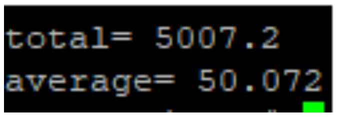
```
total= 5044.3
average= 50.443
```

Figure 3.1.1.1.8 Average CPU Usage of p1



```
total= 5056.6
average= 50.566
```

Figure 3.1.1.1.9 Average CPU Usage of p1



```
total= 5007.2
average= 50.072
```

Figure 3.1.1.1.10 Average CPU Usage of p1

3.2. MAIN SUBHEADING 2

4. CONCLUSION

5. REFERENCES