NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4186 - Operating Systems

Exercise 2

Scheduling, Processes, and Threads

Professor:
Di Liu
Teaching Assistant:
Roman K. Brunner
Student Assistants:
Eik Hvattum Røgeberg
Halvor Linder Henriksen
Ole Ludvig Hozman

Handout: 26.01.2023

Introduction

This weeks exercise is focussing on processes, threads, and how to schedule them. While we are not covering all topics in this exercise, you see a few different types of exercises. We provide more exercises than we expect you to solve this week, but we suggest you can use them to prepare for the exam in the end of the semester.

1 Processes

We first take a look at processes, their states and abstractions they provide.

1. Which abstraction do processes provide? Why don't we just run an executable directly on the processor and why do we keep all this additional state that comes along with a process?
2. You learned about the fork() + exec() way of starting a new process under Unix like systems. Give two advantages and two disadvantages of handling process creation using the fork/exec paradigm. *Sidenote: Not all systems opted for the fork/exec paradigm. Windows for example has a CreateProcess function that creates a new process from scratch, loading the specified binary.*
3. Which processor modes do exist? Why do we want them and what are the differences between a normal program and a kernel with respect to those modes?
4. How can we invoke functionality only the OS can provide? How does such an invocation differ from a normal procedure call?
5. Draw a state diagram of the fork+execute creation of processes as seen from the User Space processes.
6. Which possibilities do we have to switch between multiple processes? What are the advantages and disadvantages of the different approaches?
7. What option does an OS (using the cooperative scheduling approach) have to deal with a process that does not behave? Is process, memory, and I/O isolation still guaranteed?
8. Describe a timer interrupt-induced context switch as seen by a) process A that is executed before the interrupt, b) process B that will be executed after the interrupt, and c) the OS kernel. What is different for process A after it gets to run again after being scheduled out?

2 Scheduling

For the following exercises, we will discuss scheduling. Please refer to the information in Table 1 for all the tasks.

Job ID	Arrival Time	Execution Time
1	0	50
2	5	250
3	7	3
4	10	50
5	150	50

Table 1: All jobs to be scheduled

1. Assume all jobs arrive at time 0, which of the scheduling algorithms presented in class (FIFO, SJF, STCF, RR) is the best with respect to the average response time? If strict ordering

is required, use the job id as tie breaker. Assume 10 for every constant that hasn't been specified otherwise.

2. Taking the arrival time into account, which of the scheduling algorithms presented in class is the best with respect to response time?

3. What would be the average turnaround time when we run jobs 1-4, using the STCF scheduling strategy? Is it optimal?

4. Assume you are designing a system that has three jobs that it needs to execute. The jobs are recurring, so they are executed again after a certain time interval. Assume all jobs arrive at the same time in the beginning. Can you come up with a schedule that allows all the jobs to finish before their respective deadline? Can you develop a new algorithm on how to come up with such a schedule (Think about the ones you already know)? Can you reuse some of the ideas of the scheduling strategies presented in class?

Job ID	Execution Time	Deadline	Interval
1	3	12	12
2	1	2	4
3	3	4	6

Table 2: Jobs with deadline and interval