CS 452/652 Fall 2023 - Kernel (Part 1)

(Version: 1)

Due Date: Thu, Sep 28, 8:30am

Introduction

In the first part of your development of the kernel, you make it possible to create and run a task. To do so you need to have working

1. a context switch in and out of the kernel,
2. a collection of task descriptors,
3. a request mechanism for providing arguments and returning results,
4. priority queues for scheduling, and
5. kernel algorithms for manipulating task descriptors and ready queues.

It is not necessary to implement memory management or protection, but you must not have any task accessing the kernel's or another task's memory at any time. It is necessary for the kernel to access task memory, because the kernel must copy messages that are presented to it in the form of pointers. It might happen as a result of programming bugs that you inadvertently overwrite kernel memory from a user task. This is considered to be a bug, and usually, but not always, acts like a bug.

In addition to the kernel primitives, you must program the first user task, and another task. These tasks will provide the marker with a test of your task creation, your context switch, and your task scheduling.

Description

Kernel

To accomplish this part of the kernel you must have the following kernel primitives operating:

`int Create(int priority, void (*function)())`,
`int MyTid()`,
`int MyParentTid()`,
`void Yield()`, and
`void Exit()`.

See the kernel description for details of how these primitives should operate.

`Yield()` does not play a role in future assignments. It is implemented in this part of kernel development in order to make it possible to test the kernel in its incomplete state.

In addition you must program a first user task, which is the task automatically started by the kernel as part of its initialization, and another user task, which is to be instantiated four times.

The first user task is responsible for bootstrapping the application into existence by creating other user tasks. An operating system similar to single-user Unix could be built on top of your kernel by having the first user task login the user and providing an interactive shell after a successful login. You are not required to make your first user task login or a shell, but you are also not forbidden to do so. The other tasks, of which there will be four instances, are remainder of the application.

User Tasks

The following user tasks test your kernel.

First User Task

The first user task should create four instances of a test task.

- Two should be at a priority lower than the priority of the first user task.
- Two should be at a priority higher than the priority of the first user task.
- The lower priority tasks should be created before the higher priority tasks.
- On return of each `Create()`, busy-wait IO should be used to print "Created: <task id>" on the terminal screen.
- After creating all tasks the first user task should call `Exit()`, immediately after printing "FirstUserTask: exiting".

The Other Tasks

The tasks created by the first user task have the same code, but possibly different data.

- They first print a line with their task id and their parent's task id.
- They call `Yield()`.
- They print the same line a second time.
- Finally they call `Exit()`.

Submitting Your Assignment

Hand in the following, nicely formatted.

1. Your readable and documented code should be in your git.uwaterloo.ca git repository, readable by the TAs and instructor. Your repo should include a README-K1 file, which describes how to make and operate your program. Each group should be working from a single repository.
2. Each group should submit a single PDF file into the K1 dropbox for this course in Learn. The PDF should include the following information:
    - The names, userids, and student IDs of your group members.
    - The SHA of the commit from your group's git repository which we should be using for this assignment. The commit must be dated before the assignment deadline.
    - A description of the structure of your kernel so far. We'll judge your kernel primarily on the basis

of this description. Describe which algorithms and data structures you used and why you chose them. Describe your choices for critical system parameters and limitations, such as stack size, maximum number of tasks, etc.
    - A description of the output produced by your program, and an explantion for why it occurs in the order it does.

We may perform code reviews with students, in which case a TA/instructor and will contact you to set up a meeting.