CS452 – WINTER 2016
KERNEL 1

BILL COWAN
UNIVERSITY OF WATERLOO

A. 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 priority queues.

It is not necessary to implement memory management or protection, but you must not have any task accessing the kernel's address space at any time.[*] (It is necessary for the kernel to access user tasks' address spaces because the kernel must copy messages that are presented to it in the form of pointers.)

Hint. My kernel at the end of this assignment has less than thirty lines of assembly code. (During development it had at various stages many more, but all the diagnostic code has been removed.)

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 context switch, your task scheduling and your task creation.

Comment. RedBoot provides you with a starting environment in which the memory is mapped flat, the caches are disabled, and interrupts are disabled. This environment is right for the first part of the kernel.

B. DESCRIPTION

B.1. KERNEL

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

- `int Create( int priority, void (*code) ( ) ),`
- `int MyTid( ),`
- `int MyParentTid( ),`

[*] It will happen as a result of programming bugs that you inadvertently over-write the kernel from a user task. This is considered to be a bug, and usually, but not always, acts like a bug.

- `void Pass( )`, and
- `void Exit( )`.

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

(Pass( ) and Exit( ) do not play a role in an operating kernel. They are 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 a 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.* The other tasks, of which there will be four instances, are remainder of the application.

B.2. User tasks.

The following user tasks test your kernel.

B.2.I. *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: <taskid>.' on the RedBoot screen.
- After creating all tasks the first user task should call Exit, immediately after printing 'FirstUserTask: exiting'.

B.2.II. *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 Pass.
- They print the same line a second time.
- Finally they call Exit.

C. Hand in

Hand in the following, nicely formatted and printed.

* 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, 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.

1. A description of how to operate your program in a `readme` file, including the full pathname of your executable file which we will download for testing.
2. A description of the structure of your kernel. We will judge your kernel primarily on the basis of this description. Describe which algorithms and data structures you used and why you chose them.
3. The location of all source code you created for the assignment, which must be in a directory readable and writable by your group, which includes the TAs, and MD5 hashes of each file. The code must remain unmodified after submission until the assignments are returned.
4. The output produced by your program and an explanation of why it occurs in the order it does.