CS452 – Spring 2010
Kernel 3

Bill Cowan
University of Waterloo

A. Introduction

In the third part of your development of the kernel, you add processing of interrupts generated by a counter-timer to your kernel. You use this capability to create a clock server. To do so you need to have working

1. an implementation of AwaitEvent,
2. implementations of a clock server and a clock notifier, and
3. implementations of Delay, Time and DelayUntil as wrappers for Send to the clock server.

Tasks that want to obtain service from the clock server must do so by obtaining its tid from the name server.

In addition to the kernel primitives you must program some clients. The clients play the game against one another by interacting with the kernel.

Comment. It is not uncommon to discover weaknesses in the first part of the kernel as you work on the later parts. So it is good strategy to start early.

B. Description

B.1. Kernel

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

- `int AwaitEvent( int eventType )`.

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

In addition you must program a first user task, which

- creates the name server,
- creates the clock server, and
- creates the clock server's clients.

B.2. The Clock Server and Notifier

The clock server should be implemented as described in the kernel documentation.

The clock notifier should wait on events from a timer, and Send notifications that the timer ticked to the clock server. As discussed in class,

there are two options for implementation of the kernel and notifier. Students may implement either of the two options.

B.3. Client tasks.

A single type of client task tests your kernel and clock server. It is created by the first user task, and immediately sends to its parent, the first user task, requesting a delay time and a number of delays. It then uses `WhoIs` to discover the `tid` of the clock server.

It then delays the given number of times using the given time interval. After each delay it prints its tid, its delay interval, and the number of the just completed delay on the RedBoot terminal. Finally it exits.

B.4. First user task

The first user task creates the clock server, and four client tasks. It then executes Receive four times, and Replies to each client task in turn. It then exits.

The following table shows the parameters to be given to the clients.

Priority (smaller is higher)	Delay Times (ticks)	Number of Delays
3	10	20
4	23	9
5	33	6
6	71	3

C. Hand in

Hand in the following, nicely formatted and printed.

1. A description of how to operate your program, including the full pathname of your executable file which we will download for testing.
2. A description of the structure of your kernel so far.* 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 and a set of MD5 hashes of each file. The code must remain unmodified after submission until the assignments are returned.
4. A listing of all files submitted.
5. Output produced by your client tasks and an explanation of why it occurs in the order it does.

* You should be accumulating your hand in documents as you go along because we expect a description of your complete kernel when its development is complete. But now hand in only a description of what is new or changed.