Laboratory Exercise 5
A Real-Time Scheduler

Goals

After this laboratory exercise, you should understand how it is possible to use interrupts to implement a simple real-time kernel.

Literature

- Brorsson 5.4–5.7
- Patterson and Hennessy: Appendix A.7.
- MIPS Lab Environment Reference Manual

Preparation

Read the literature and this laboratory exercise in detail and solve the home assignments.

Home Assignment 1

Study the source code of the small real-time kernel ITS Lite. The real-time kernel source code is located in the directory ITS_Lite. It is written in both C and assembler.

- *its.c*
  Most of the kernel routines
- *its.h*
  Declarations of kernel routines, to be included in files where ITS Lite is used
- *its_asm.s*
  Those kernel routines that are written in assembler, for example, saving registers and changing processor state
- *interrupt.s*
  Routines for manipulation of the status and cause registers
- *util_asm.s*
  Routines to read and write registers in C programs
- *util.h*
  Declarations for the routines above
- *test.c*
  An example of how to use the ITS Lite kernel

ITS Lite has the following functions:

- `ITS_init();`
  Create the internal data structures of the kernel. This call must be made before the other kernel functions can be used.
- `ITS_create_thread(void (*entry), void *arg, unsigned int *stack);`
  This call creates a thread. The first argument is the name of the subroutine which contains the code for the thread. The second argument is a pointer to a data structure which will be passed as an argument to the subroutine, and the third argument is a pointer to an array, which will be used as the stack space for the thread.
- `ITS_enter_critical();`

Call this function when the thread needs to enter a critical region.

- `ITS_leave_critical();`
  Call this function when the thread needs to leave a critical region.
- `ITS_yield();`
  Call this function when the thread wants to wait and leave the execution to another thread.

Make sure that you understand the code and the tasks of all the different programs and subroutines. Answer the following questions:

- What are the tasks of the subroutines in each of the files above?
- In what file are the data structures that store thread information located?
- Which subroutines in which files are called during startup of the kernel?
- Which subroutines are called in what order when a timer interrupt occurs?
- What are the main tasks of each subroutine called at a timer interrupt?

Home Assignment 2

The real-time kernel is controlled by timer interrupts from the external timer on the lab board. You can set the frequency of the timer with a dial. Every time a timer interrupt occurs, the executing thread is interrupted and another thread is started instead.

Study the real-time kernel code and add a function called `WaitTime(n)`, which will cause the tread that called the function to wait `n` ticks (timer interrupts) before it can be executed again.

A Simple Real-Time Kernel

The scheduler code given above can be executed and there is a test program which starts three threads and runs them on the lab computer.

Assignment 1

Copy the code of the real-time kernel to your local directory. Create a C minimal project, add the source code of the kernel, build it and run it on the lab computer. Study the output of the different threads when the threads are executing.

Assignment 2

The two functions `timer_start()` and `timer_stop()` can be used to measure execution time. The function `timer_start()` tells the timing to start, while `timer_stop()` stops the timer and returns the time since the start. On the lab computer, the unit is microseconds, and in the simulator is the number of clock cycles used.

Use these two functions to measure how long it takes between two timer interrupts. Note that you can set the interrupt frequency with a dial on the lab board. Change the frequency and verify that you can measure how the timer interrupt interval changes.

Assignment 3

Add the code of Home Assignment 2 to the real-time kernel. Then add a call to `WaitTime` in one of the threads in the test program. Run the kernel and check that your code works.

Assignment 4

Whenever a timer interrupt occurs, the executing thread is replaced by another. In order to do this, the scheduler has to store the processor state of the old thread and replace it with the state of the new thread. This means that all the registers have to be replaced, called a *context switch*.

How long does a context switch take? Try to measure how long a context switch takes by counting the number of clock cycles needed on the simulator. The function `timer_stop` returns the number of clock cycles used on the simulator. You can also click on the I-Cache and see the current number of clock cycles in the variable `Cycle Count`.

Conclusions

Before you finish the laboratory exercise, think about the questions below:

- Why are threads needed?
- What is a context switch? What is a power switch?
- What is the actual state of the processor that must be saved for each thread?
- Why must the temporary registers t0–t7 be saved at a context switch?
- If a real-time kernel is too slow, what could be the alternative?