Lab #2
⋆
Interrupts and Alarms

November 12, 2012

Note: All the software and documents are stored at `http://www.irccyn.ec-nantes.fr/~bechennec/trampoline`

1 Goal

Real-Time systems are reactive systems which have to do processing as a result of events. You have seen in Lab #1 how to start processing as a result of an internal event of the system: by activating a task (`ActivateTask` and `ChainTask` services) or by setting an event (`SetEvent` service). In this lab, you will see how to trigger processing as a result of an external event (a hardware interrupt) or as a result of time passing (expiration of an Alarm). This lab uses the following concepts: ISR, alarm, counter. Go into the `trampoline/labs/lab2` directory.

2 Hardware interrupts simulation

A Unix process may be interrupted by a set of software interrupts called signals. When a signal is raised, a function (called signal handler) is immediatly executed. This is used to simulate a hardware interrupt. In Trampoline/posix, an ISR is tied to a signal. So hardware interrupt simulation is done by sending a signal to Trampoline process. To focus on real-time, Unix system programming is already done in `lab2.c` and `main.c`:

- The terminal is put in raw mode. This means that characters are sent directly to Trampoline without being displayed. Ctrl-C does not work anymore and 'q' is used to quit Trampoline. By the way, a return is not automatically inserted when a printf occurs and you should put `\r\n` at the end of strings. The terminal is put back in normal mode (cooked mode) when exiting. If a strange behavior occurs, kill the terminal and start another one;

- When 'a' key is pressed, software interrupt `SIGPIPE` is sent to Trampoline;
- When 'b' key is pressed, software interrupt `SIGTRAP` is sent to Trampoline.

To connect and ISR2 to a signal, you have to specify a `SOURCE = SIGUSR2` or `SOURCE = SIGTRAP` in the ISR description in the OIL file and to write the ISR code in the `lab2.c` file.

3 First application

Using the application of lab2 as a starting point, program an application which does a computation when an interrupt occurs. You will use `SIGUSR2` and a task named `t_process`, priority 3 that display "processing triggered by SIGUSR2" and does an empty for loop 10 millions times. Switch on the Pre and PostTaskHook to print the id of the task that gets and looses the CPU.

Question 1 *If you press 3 or 4 times quickly on key 'a', what problem appears ? How to solve it ?*

Question 2 *What problem has the previous solution ?*

4 Second application

Program an application with 2 periodic tasks: `t1` (priority 2, period 1s) and `t2` (priority 1, period 1.5s). Each display its name then does an empty for loop 1 million times.

Question 3 *Without running your program, give by hand the 10 first lines displayed by the execution of the application with the display date of each line (0 being the application startup date). Is the whole system periodic ? If yes, what is the period and the behavior.*

Question 4 *The application needs a counter and 2 alarms. In Trampoline/posix a counter is connected to a timer with a 10ms cycle time. What `TICKSPERBASE` do you use to fulfill the application requirements ?*

Question 5 *How are the alarms configured to fulfill the application requirements ? Declare the counter and both alarms and write the application. Verify it works.*

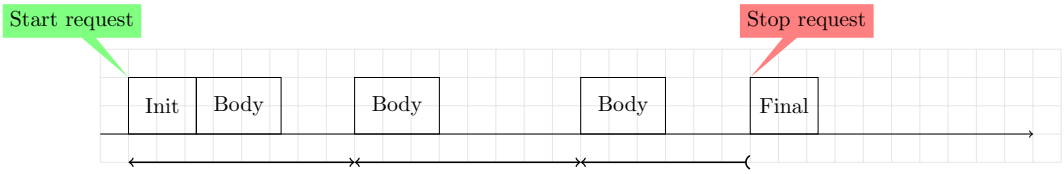
5 Third application

In the third application, alarms, counters and ISR are mixed. This application is a system with a push button and a switch. After starting the system waits. When the button is pressed, the system start a function F that is implemented using a periodic

task (period = 1s). When the button is pressed again, function F is stopped. When the switch is pressed, the system is shutdown as quickly as possible.

Question 6 *Design and program this application using Trampoline. Computation in the periodic task is simulated using an empty for that loops 1 million times.*

Requirements change. Now function F implementation needs an Init code (runs once when the F is started) and a Final code (runs once when F is stopped). This corresponds to the following diagram:



Question 7 *Modify the application to take the new requirements into account. Use 3 basic tasks to implement function F. Init and Final simulates the workload by an empty for looping 100 000 times.*

Question 8 *Same question but with only one extended task to implement function F.*

6 Fourth application

In this part, you will implement a watchdog. It is a mechanism that allows to stop a processing or the waiting for an event when a deadline occurs.

Question 9 *In your application, each time 'a' is pressed, 'b' must pressed within 2 seconds. In such case, you print the time between the two occurrences. Otherwise, an error message is displayed. To simplify, we suppose 2 'a' may not be got within 3 seconds.*

Question 10 *What is happening if the timeout occurs just after 'b' has been pressed but before the waiting task got the event ? (draw a Gantt diagram of this scenario) If your application does not handle correctly this scenario, modify it.*