**Assignment 5: Event Handle and Signaling (100 points) – CSE 438/598, Fall 2014**

**Assignment Objectives**

1. To learn the basic programming technique for input devices in Linux kernel.
2. To learn event handling and signaling in Linux
3. To develop program patterns for imprecise computation.

***Project Assignment***

In Linux, the signals arrived from input devices, such as keyboard and mouse, are handled by interrupt routines. For instance, when you press a key, an interrupt is triggered and the corresponding interrupt service routine reads in the key information from hardware interface and passes it to the driver and event handlers. All the processing is done in interrupt mode (either in ISRs or bottom-half routines). So, if you press ctrl-c, the key information is received and recognized. Then, a signal is sent to terminate the running process. The implementation of this interrupt process involves several modules from interface adapters, e.g., USB controller for USB keyboard and mouse, input core, and event deriver. The inputs can eventually be saved in a buffer for user programs to read, or result in asynchronous actions such as kill. It is flexible to reassign which key input to trigger the kill and to install additional event handlers for other inputs.

At user space, program runs sequentially. To deal with asynchronous events, such as exceptions, the Linux signaling mechanism can be used. When a signal is issued, the tasks that registers to receive the signal stops the running program and calls the corresponding signal handlers. When the handler is done, the stopped execution can be resumed, terminated, or switched to different entry points. The signal handler runs in the task's context. Thus, it shares the same memory space as the task and all static variables are accessible by the handler.

The need of employing asynchronous handling in user programs can be demonstrated by the imprecise computation model for real-time systems. In imprecise computation, the accuracy of the computation result can be improved after iterations of computation. For real-time systems, computation results must be put out before the associated deadline or upon a request. Hence, if needed, we may need to abort the current iteration of computation and send out the results done in the previous iteration.

In this assignment, you are asked to complete the following three tasks related to event handling and signaling.

1. Setjmp and longjmp

   Use Linux signal handling mechanism, setjmp and longjmp to demonstrate an example imprecise computation programming pattern in which executing computation will be terminated and the existing results are put out when we double click the right mouse button.

2. Signal handling in Linux

   In vxWorks' Kernel API Reference Manual, it is stated that "If a task is pended (for instance, by waiting for a semaphore to become available) and a signal is sent to the task for which the task has a handler installed, then the handler will run before the semaphore is taken. When the handler returns, the task will go back to being pended (waiting for the semaphore). If there was a timeout used for the pend, then the original value will be used again when the task returns from the signal handler and goes back to being pended. If the handler alters the execution path, via a call to longjmp( ) for example, and does not return then the task does not go back to being pended."

This description of the signal delivery mechanism is certainly OS dependent. In task 2 of the assignment, you are requested to develop a program or several pieces of program to test what the Linux's signal facility does precisely and to show the time that a signal handler associated to a thread gets executed in the following conditions:

a. The thread is running.

b. The thread is runnable (i.e. the running thread has a higher priority).

c. The thread is blocked by a semaphore (i.e. sema_wait() is called) or a file IO (e.g., read).

d. The thread is delayed (i.e., nanosleep() is called).

To show the time that a signal handler is invoked, you are required to use "*trace_cmd*" to collect events from the Linux internal tracer *ftrace*. The traced records can then be viewed via a GUI front end *kernelshark* in a Linux host machine. A report should be compiled that includes *kernelshark's* report images to illuminate the execution of signal handlers.

3. Signal delivery to all threads of a process [Required for CSE 598 students]

In Linux signal man page, it describes: *"A process-directed signal may be delivered to any one of the threads that does not currently have the signal blocked. If more than one of the threads has the signal unblocked, then the kernel chooses an arbitrary thread to which to deliver the signal."* This suggests, no matter how many threads having the signal unblocked, the signal will be delivered once to an arbitrary thread.

Please write a test program to demonstrate this statement is correct for SIGIO and then build a set of routines (library) that allows SIGIO signal being delivered to all threads that are registered to receive the signal.

**Note**

You can use any Linux machine to develop your programs for this assignment. This is no need to do it on Galileo board. In fact, we are not able to run kernelshark in a Linux host with the trace collected from Galileo board.

**Due Date**

This is assignment is due on Friday, Dec. 5, at 11:59pm.

**What to Turn in for Grading**

- Create a working directory that has 3 subdirectories for tasks 1 to 3. Each consists of any source files of user-level testing programs, makefiles, and readme file. Please make sure your approaches for tasks 1 and 3 are explained in the readme file. For task 2, a report should be included in the subdirectory.
- Comment your source files properly and rewrite the readme file to describe how to use your software.
- Compress the directory into a zip archive file named cse438(598)-lastname-assgn05.zip.
- Submit the zip archive to Blackboard by the due date and time.
- Failure to follow these instructions may cause an annoyed and cranky TA or instructor to deduct points while grading your assignment.