CIS 657 (POS) fall 2013

Lab 5: Software Interrupt Implementation

# 1. Software Interrupts

#### The mechanism for doing lower-priority processing is called a [software interrupt](mk:@MSITStore:D:\E%20books\The%20Design%20And%20Implementation%20Of%20The%20FreeBSD%20Operating%20System%20(2004).chm::/gloss01.htm#gloss01entry371). Typically, a high-priority interrupt creates a queue of work to be done at a lower-priority level. As with hardware devices in FreeBSD 5.2, each software interrupt has a process context associated with it. The software-interrupt processes are generally given a lower scheduling priority than the device-driver processes but a higher priority than those given to user processes. Whenever a hardware interrupt arrives, the process associated with device driver will attain the highest priority and be scheduled to run. When there are no device-driver processes that are runnable, the highest priority software-interrupt process will be scheduled to run. If there are no software-interrupt processes that are runnable, then the highest priority user process will run. If either a software-interrupt process or a user process is running when an interrupt arrives and makes its device-driver process runnable, the scheduler will preempt the software-interrupt or user process to run the device-driver process.

#### Clock Interrupts

Interrupts for clock ticks are posted at a high hardware-interrupt priority. After switching to the clock-device process, the hardclock() routine is called. It is important that the hardclock() routine finish its job quickly: If hardclock() runs for more than one tick, it will miss the next clock interrupt. So the time spent in hardclock() is minimized, less critical time-related processing is handled by a lower-priority software-interrupt handler called softclock().The remaining time-related processing involves processing timeout requests and periodically reprioritizing processes that are ready to run. These functions are handled by the softclock() routine. When hardclock() completes, if there were any softclock() functions to be done, hardclock() schedules the softclock process to run.

(Excerpts from the textbook: The Design and implementation of FreeBSD Operating System by Marshall Kirk McKusick)

# 2. swi\_add() and swi\_sched() function calls

The swi\_add() function is used to register a new software interrupt handler. The ithdp argument is an optional pointer to a struct ithd pointer. If this argument points to an existing software interrupt thread, then this handler will be attached to that thread. Otherwise a new thread will be created, and if ithdp is not NULL, then the pointer at that address to will be modified to point to the newly created thread. The name argument is used to associate a name with a specific handler. This name is appended to the name of the software interrupt thread that this handler is attached to. The handler argument is the function that will be executed when the handler is scheduled to run. The arg parameter will be passed in as the only parameter to handler when the function is executed. The pri value specifies the priority of this interrupt handler relative to other software interrupt handlers. If an interrupt thread is created, then this value is used as the vector, and the flags argument is used to specify the attributes of a handler such as INTR\_MPSAFE. The cookiep argument points to a void \* cookie. This cookie will be set to a value that uniquely identifies this handler, and is used to schedule the handler for execution later on.

The swi\_sched() function is used to schedule an interrupt handler and its associated thread to run. The cookie argument specifies which software interrupt handler should be scheduled to run.

Source: <http://www.unix.com/man-page/FreeBSD/9/swi/>

populates

swi\_add()

[ithread\_create](http://fxr.watson.org/fxr/ident?v=FREEBSD52;im=excerpts;i=ithread_create)()

[ithread\_add\_handler](http://fxr.watson.org/fxr/ident?v=FREEBSD52;im=excerpts;i=ithread_add_handler)()

struct ithd

struct [intrhand](http://fxr.watson.org/fxr/ident?v=FREEBSD52;i=intrhand)

(cookiep)

populates

calls

swi\_sched()

[ithread\_schedule](http://fxr.watson.org/fxr/ident?v=FREEBSD52;im=excerpts;i=ithread_schedule)()

interrupt thread (ithd)

executes

calls

# 3. Relevant Files

* [interrupt.h](http://fxr.watson.org/fxr/source/sys/interrupt.h?v=FREEBSD52): Struct interrupt thread (ithd) and interrupt handler(intrhand) are defined here
* [kern\_intr.h](http://fxr.watson.org/fxr/source/kern/kern_intr.c?v=FREEBSD52): swi\_add() and swi\_sched() functions are defined here. Also refer function start\_softintr() function for usage of swi\_add()
* [kern\_clock.c](http://fxr.watson.org/fxr/source/kern/kern_clock.c?v=FREEBSD52): Refer function hardclock() to see usage of swi\_sched()

# 4. Tasks [90]

In this lab, students need to implement a Software interrupt via system calls. Following is the list of tasks to be accomplished:

1. Create an interrupt handler routine function, which will print the priority of interrupt thread.
2. Create a system call, which will add a software interrupt using function swi\_add(). It will pass the handler created in step 1 as an argument to swi\_add(). This system call should take software interrupt name as argument.
3. Create another system call, which will execute the software interrupt using function swi\_sched().
4. Create a user level program which will invoke the system calls created in step 2 and 3. It should pass software interrupt name as argument to system call created in step 2.
5. Your observation on priority printed by interrupt handler routine function, is it better or worse than user level priority and Why?

# 5. Submission [10]

Create and attach a README (txt/word/pdf) file at the end of the lab. It doesn't need to be comprehensive, but it should at least cover the following content:

Which tasks are done, and which are not?

What’s your basic idea to achieve this task?

Where is your main function? Which files you have modified and under which function?

If you can only finish some of the tasks in this project, please make sure that your code can at least be compiled and installed and also clearly state in the README file about the missing parts of your project.

You also need to attach the .tgz file.To create this file, there is script on the website named “tar and Gzip Source code” under resources tab.

Checklist: To submit your project, you need to:

• Attach the \*.tgz file(make sure kernel compiles and runs with this snapshot) and user level program to invoke system calls.

• Create and attach a README file report.

• Send this email to the TAs keeping Dr. Chapin < chapin@syr.edu> in the CC with subject line “CIS657: Lab 5”

Saurabh Sabnis < spsabnis@syr.edu >

Prasoon Pandya < [pdpandya@syr.edu](mailto:pdpandya@syr.edu) >