CS 503 - Fall 2014

Lab 5
Signal Handling
Event Registration and Notification

Due Sunday, October 12th, 2014 at 11:59 PM

Objectives:

By the end of this lab you will be able to:

- Understand how to create an infrastructure for event driven programming in an operating system.
- Create a new system call to register for events.

Background

Today operating systems such as Android have a mechanism for user programs (processes) to receive notifications from the operating system when various events occur. For example, in Android when the user "clicks" on the screen an event is generated and sent to the program by calling the onClick function.

Your job is to add a similar functionality to XINU by allowing a proces to register a function pointer which will be called when events are signaled. Events can be signaled by either a user process or the operating system ifself.

To do this you will create two new systems call called regevent and sendevent. The format for regevent is as follows:

```
syscall regevent(void (*event_handler)(uint32 event), uint32 event);
```

This system call registers the event handler (event_handler) with the operating system for a given event. The event handler itself is a function with the following prototype:

```
void event_handler(uint32 event);
```

This function takes a message as a parameter which will correspond to the event being handled. The format for the sendevent system call is as follows:

```
syscall sendevent(pid32 pid, uint32 event);
```

NOTE: this system call is very similar to the send system call already implemented in XINU, however this function will cause the registered event handler to be called instead of delivering a message to the process.

Setup

In /homes/cs503/xinu there is a file called xinu-fall2014-lab5.tar.gz that contains a start to the code. Unpack:

```
tar zxvf /u/u3/cs503/xinu/xinu-fall2014-lab5.tar.gz
```

This will create a directory called xinu-fall2014-lab5.

Along with the main code for XINU, this tarball contains the following files (additional explanation of the contents of the files is in the following sections).

- system/regevent.c - function declaration for the regevent function.
- system/sendevent.c - function declaration for the sendevent function.
- system/eventprocess.c - function declaration for the eventprocess function. (See the following sections for an explanation)
- include/prototypes.h - modified prototypes.h file which includes the regevent and sendevent functions.
- include/event.h - declaration of operating system and user events.

What's in system/regevent.c

The regevent.c file contains an empty function declaration for the regevent function. Your job is to fill in code to store the function pointer to be called when the event happens. If a process already has an event handler registered for a particular event, additional calls to regevent overwrites the exiting event handler for that event.
NOTE: the regevent system call only registers an event handler for a particular event (not all events). Consider creating an event table which records which processes have an event handler registered for each event.

What's in system/sendevent.c

The sendevent.c file contains an empty function declaration for the sendevent function. Your job is to fill in code to send an event to the particular process. If the target process identifier is not valid or does not have an event handler registered, the sendevevent system call should return SYSERR. If the sender tries to send an operating system reserved event or some other invalid event, SYSERR should also be returned.

What's in include/event.h

The event.h file contains declarations for operating system events which you must implement. They are as follows:

```
PROCESS_END_EVENT - Sent when a process finishes execution either through a kill call or by a normal return.
PROCESS_GETMEM_EVENT - Sent when a process successfully allocates memory from the heap using getmem.
PROCESS_FREEMEM_EVENT - Sent when a process succesfully frees memory to the heap with freemem.
PROCESS_WAITSEM_EVENT - Sent when a process calls wait on a semaphore but the current value of the semaphore
                        causes the process to go onto the wait queue.
USER_EVENT_BOUNDARY - This is the lowest value that a user process can use for the event sent with sendevent.
                      All events lower than this value are reserved for use by the operating system.
                      If a caller of sendevent tries to send an event with a value less than USER_EVENT_BOUNDARY
                      then SYSERR should be returned.
MAX_EVENTS - This is the maximum number of events the system can support.  An event must be less than this value.
```

How do I call the event handler?

NOTE: that at the time an event is sent, the process that has the event handler registered is not the current process that is executing on the CPU.

So how do you get the event handler to run? For this lab, a new process is created who's job is to run the event handlers. This process will wait on a message using the receive system call. When an event is sent, it should be sent a message to wake up and call the appropriate event handler.

The process identifier for the event process is store in a global variable called eventprocesspid. This variable is declared in eventprocess.c. To use the value in another source file just define it externally:

```
extern pid32 eventprocesspid;
```

The event process is created in initialize.c right before main is created using the following:

```
eventprocesspid = create(eventprocess, 4096, 100, "Event process", 0, NULL);
resume(eventprocesspid);
```

Additional Requirements:

- When an operating system event happens, the process that has an event handler registered gets their event handler called. Remember under normal circumstances user process are not allowed to run with interrupts disabled.
- Make sure to remove all debug output from your system calls. When the TAs run your submitted code, calling a system call directly should not produce any output.
- Provide a set of test cases to ensure that your code works as required. Put these test cases in main.c
- The TAs will be replacing main.c with their own test cases after running your submitted test cases. Make sure you do not define any dependent variables in main.c. You are free to modify any other file(s) to implement the lab requirements. Just make sure that there are no dependent declarations in main.c.
- If your submitted code does not compile (either the exact submitted code or the code after the TA's replace any test case files), you will receive zero (0) points for code execution. If this happens, you will be allowed to resubmit for half credit only.
- Please run "make clean" prior to submission so that you don't submit object files
- NOTE: When you make xinu for this lab the make file will generate two files in the compile directory:
  - xinu - this is the file you will download to the xinu backend (you will NOT use xinu.xbin)

    - xinu.elf - this is the executable and linkable format version of the xinu binary. This is not the format to use when sending to the backends. It is useful for low level (assembly) debugging only.
- Some tips on writing an affective report can be found here.

Extra Credit:

Add another system call with the following prototype:

```
syscall broadcastevent(uint32 event);
```

Which sends an event to all processes that have an event handler registered for the input event. The order in which the event handlers are invoked is not important.

Grading Criteria

The lab will be graded out of 100 total points with 10 additional points possible from the extra credit:

- 25 Points - How well your test cases verify that your code has met the requirements.
- 50 Points - The results from the TA's test cases.
- 25 Points - How well your report answers the discussion questions and your test case explanation.

What to turn in

Submit using turnin command your complete source code (all of XINU) including the any files you added to complete the lab. In the 'system' directory include a PDF file with a write up discussing:

- The details behind your implementation. As part of this discussion write answers to the following questions:
    - Describe your test cases. How to they ensure that your code correctly meets the requirements?
    - Consider an additional requirement (you do not have to implement it) where the event handler must be run in the context of the process that registered it. How would the code have to modified to handle that case? What process context is executing when sendevent is called?
    - This lab only uses a couple operating system generated events. Come up with four (4) additional operating system events that you think would make sense to add (you do not need to implement these, just name them). Explain each and why you think they would be useful.

To turn in your lab use the following command

```
turnin -c cs503 -p lab5 xinu-fall2014-lab5
```

assuming xinu-fall2014-lab5 is the name of the directory containing your code.

If you wish to, you can verify your submission by typing the following command:

```
turnin -v -c cs503 -p lab5
```

Do not forget the -v above, as otherwise your earlier submission will be erased (it is overwritten by a blank submission).

Note that resubmitting overwrites any earlier submission and erases any record of the date/time of any such earlier submission.

We will check that the submission timestamp is before the due date; we will not accept your submission if its timestamp is after the due date. Do NOT submit after 11:59 PM Eastern Time.