CS 503 Lab 5

Signal Handling
Event Registration and Notification

Shuheng Guo

1. Introduction

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.

In this lab, I am going 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 itself.

To realize this functionality, I implemented two new system call 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);

2. Implementation

The general idea in my mind is to add a new item in process table to play the role of event table. In this case, every process has an event table to store its event handler. In addition, event tables are initialized when the system boots every time. Every process' event table is cleared when it is terminated so that the table can be reused when other processes are created. The specific functions are discussed as follows.

2.1 regevent.c

```
#include <xinu.h>

extern pid32 eventprocesspid;

/*------------------------------------------------------------------------
 *  regevent  -  Register an event handler
 *------------------------------------------------------------------------
 */
syscall regevent(
      void (*eventhandler)(umsg32 event),   /* Pointer to event */
                        /*   handler        */
      uint32 event          /* Event to register for    */
    )
{
    intmask mask;
    struct  procent *prptr;
    mask = disable();
    if (event <= 0 || event > MAX_EVENTS ) {
        restore(mask);
        return SYSERR;
    }
    prptr = &proctab[currpid];
    prptr->event_table[event - 1] = eventhandler;
    restore(mask);
    return OK;
}
```

Figure 1. regevent.c

Function regevent is to register a given event handler to current process' event table. So when the event is sent, the event process can find its handler from proper place.

2.2 sendevent

```
#include <xinu.h>

extern pid32 eventprocesspid;

/*------------------------------------------------------------------------
 *  sendevent  -  Sends an event to the given process
 *------------------------------------------------------------------------
 */
syscall sendevent(
      pid32     pid,        /* ID of recipient process  */
      uint32    event       /* Event to send        */
    )
{
    int32 msg = pid;
    intmask mask;
    struct  procent *prptr;
    mask = disable();
    if (isbadpid(pid) || event < USER_EVENT_BOUNDARY || event > MAX_EVENTS ) {
        restore(mask);
        return SYSERR;
    }

    prptr = &proctab[pid];
    if (prptr->event_table[event - 1] == 0) {
        restore(mask);
        return SYSERR;
    }
    msg = (msg << 16);
    msg = msg + event;
    send(eventprocesspid, msg);
    restore(mask);
    return OK;
}
syscall OS_sendevent(
      pid32     pid,        /* ID of recipient process  */
      uint32    event       /* Event to send        */
    )
{
    int32 msg = pid;
    intmask mask;
    struct  procent *prptr;
    mask = disable();
    if (event >= USER_EVENT_BOUNDARY || event <= 0) {
        restore(mask);
        return SYSERR;
    }

    prptr = &proctab[pid];
    if (prptr->event_table[event - 1] == 0 ) {
        restore(mask);
        return SYSERR;
    }
    msg = (msg << 16);
    msg = msg + event;
    send(eventprocesspid, msg);
    restore(mask);
    return OK;
}
```

Figure 2. sendevent

Function sendevent is 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.

2.3 eventprocess

```
#include <xinu.h>

pid32 eventprocesspid;

/*------------------------------------------------------------------------
 *  eventprocess  -  Process for calling event handlers
 *------------------------------------------------------------------------
 */
void    eventprocess(void)
{
    umsg32 rcvd_msg;
    void (*eventhandler)(umsg32 event);

    while(1) {
        rcvd_msg = receive();
        uint32 event = (rcvd_msg & 0xFFFF);
        rcvd_msg = (rcvd_msg >> 16);
        struct  procent *prptr;
        prptr = &proctab[rcvd_msg];
        eventhandler = prptr->event_table[event - 1];
        eventhandler(event);
    }
}
```

Figure 3. eventprocess

Function eventprocess is to receive the message from sender and call event handler from the given process' event table.

3. Result and Analysis

After carefully designing and running test case, I got the result as follows:

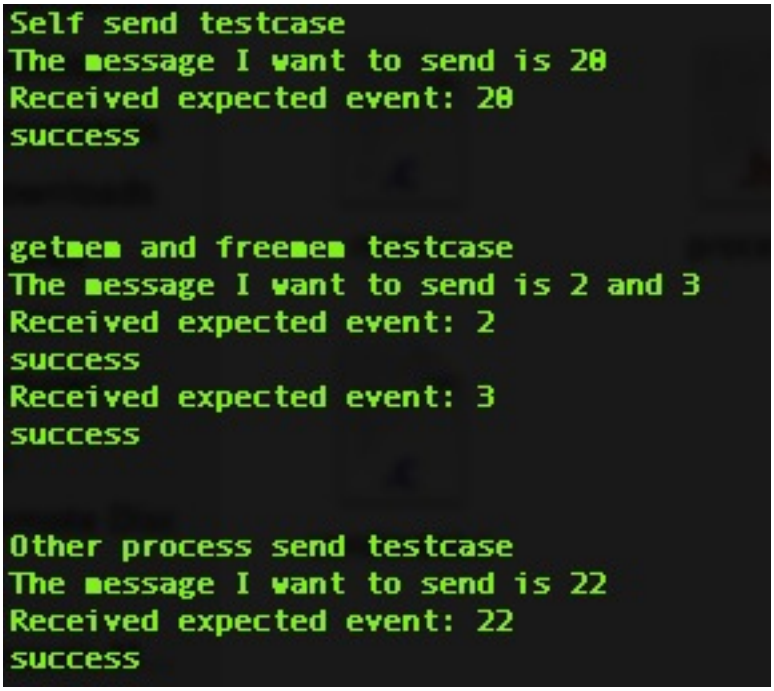

Figure 4. Result

We can see from the result that the tests of other self send testcase or OS system send testcase run successfully. That means the event is sent and event handler was invoked to run after its corresponding event was received. Basically, the functions I write should be correct.

4. Discussion

Q : Describe your test cases. How to they ensure that your code correctly meets the requirements?

A: I designed different test cases to test self send and OS send respectively. Also, I tested the case in which a process registers for an event then starts another process to send it the event. Please go to the previous chapter for details in the results.

Q: 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?

A: Considered the requirement where the event handler must be run in the context of the process that registered it. To implement this, I will modify the eventprocess to send the event handler as a message to the process who registered this event. So the process who registered the event will go to receive message after the registration. In this case any other process can send the event except the process in receiving state after registering that event.

Q: 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.

A:

PROCESS_END_EVENT - Sent when the process ends either by a kill call or though

completion of normal execution

PROCESS_WAITSEM_EVENT - Sent when a process waits on a semaphore, but is required to go onto the semaphore wait queue.

PROCESS_CREATE_EVENT – Sent when a process creates another process.

PROCESS_SIGNAL_EVENT – Sent when a process is signaled and dequeued from a semaphore and go onto the ready list.

PROCESS_SUSPEND_EVENT – Sent when a process is suspended.