Steve Babcock
10/28/2017

Here are a few assignments from my Advanced Operating System class, and instructions on how to build Xinu - an embedded system with similarities to Unix.

How to Build Xinu

1) Install gcc arm cross compiler
2) Edit compile/Makedefs so the lines specific to your operation system point to your compiler. For example, I use Mac OS X and my file contains

```
COMPILER_ROOT = /usr/local/Cellar/gcc-arm-none-eabi/20160928/bin/arm-none-eabi-
LIBGCC_LOC = /usr/local/Cellar/gcc-arm-none-eabi/20160928/lib/gcc/arm-none-eabi/5.4.1
CONF_LFLAGS = -L/usr/local/Cellar/flex/2.6.4/lib -lfl
```

The text for other operating systems is commented out.

3) Install qemu
4) cd into code/compile
5) enter “make”
6) enter “make qemu”

Assignment 1 - Process Ring

The goal is to create a countdown where different processes perform different steps, and coordinate in order to keep the countdown numerically correct.

The majority of this assignment’s code is in apps/process_ring.c and shell/xsh_process_ring.c

Run with “process_ring” and a few optional flags can be used. -p will create more processes, -r will increase the rounds, and -m will change the mode from signaling to message passing. Example: “process_ring -p 5 -r 10 -m 1” will create five processes, ten rounds, and countdown with signaling/semaphores.

Assignment 5 - Futures

The goal for this assignment is to implement functionality similar to Java's util.concurrent.Future using a producer/consumer model.

The majority of the code can be found in system/future.c

To run the new command simply type "prodcons" into the qemu terminal.

I chose to use an array based queue for each future. There are two pointers to the first and last elements of the queue. Queue functions can be found in future.c

The futures themselves are allocated on the heap, and contain a few extra properties. I felt it made more sense to have get_pid and set_pid instead of just one pid. This way, illegal processes are blocked from trying to either set or get futures when in exclusive or shared mode. Interrupts are disabled and enabled to ensure getting and setting these locks is atomic.

Assignment 3 - Array based context switch

The goal of this assignment was to convert Xinu's context switch from storing register contents on the stack to storing the contents in an array now provided in each entry of the process table.

The majority of the code for this assignment is below in the assembly function ctxsw2. The original function was ctxsw, and is also included for comparison. Though this assignment is not implemented in this version of Xinu.

It is necessary to use assembly language for context switches because the C language does not provide direct access to specific registers. This function is exposed globally, and called during resched() in C.

```
ctxsw:
        push    {r0-r11, lr}            /* Push regs 0 - 11 and lr      */
        push    {lr}                    /* Push return address                  */
        mrs     r2, cpsr                /* Obtain status from coprocess.*/
        push    {r2}                    /*   and push onto stack        */
        str     sp, [r0]               /* Save old process's SP        */
        ldr     sp, [r1]               /* Pick up new process's SP   */
        pop     {r0}                    /* Use status as argument and           */
        bl      restore                 /*   call restore to restore it    */
        pop     {lr}                    /* Pick up the return address */
        pop     {r0-r12}                /* Restore other registere      */
        mov     pc, r12                 /* Return to the new process */
```

```
ctxsw2:
        str     r0, [r0]
        str     r1, [r0, #4]
        str     r2, [r0, #8]
        str     r3, [r0, #12]
        str     r4, [r0, #16]
        str     r5, [r0, #20]
        str     r6, [r0, #24]
        str     r7, [r0, #28]
        str     r8, [r0, #32]
        str     r9, [r0, #36]
        str     r10, [r0, #40]
        str     r11, [r0, #44]
        str     lr, [r0, #48]
        str     sp, [r0, #52]
        mrs     r2, cpsr
        str     r2, [r0, #56]
        str     lr, [r0, #60]

        ldr     r2, [r1, #8]
        ldr     r3, [r1, #12]
        ldr     r4, [r1, #16]
        ldr     r5, [r1, #20]
        ldr     r6, [r1, #24]
        ldr     r7, [r1, #28]
        ldr     r8, [r1, #32]
        ldr     r9, [r1, #36]
        ldr     r10, [r1, #40]
        ldr     r11, [r1, #44]
        ldr     r12, [r1, #48]
        ldr     sp, [r1, #52]
        ldr     r0, [r1, #56]
        bl      restore
        ldr     lr, [r1, #60]
        ldr     r0, [r1]
        ldr     r1, [r1, #4]
        mov     pc, r12
```