CSC C69: OPERATING SYSTEMS

Tutorial 4
Monday, Feb 14th, 2011
Ioan Stefanovici
(ioan@cs.toronto.edu)
WHAT IS OS161?

- OS/161 is an operating system developed for educational purposes that runs on a simulated machine (a MIPS R3000).
- The code we will provide has two components:
  - OS/161: the operating system that you will augment in subsequent homework assignments. The OS/161 distribution contains a full operating system source tree, including some utility programs and libraries. After you build the operating system you boot, run, and test it on the simulator.
  - System/161: the machine simulator which emulates the physical hardware on which your operating system will run. This course is about writing operating systems, not designing or simulating hardware. Therefore you may not change the machine simulator. If, however, you think you have found a bug in System/161, please let the course staff know as soon as possible.
OS161

OS161 Source

CS161 Toolchain

gcc (native) compiler

Sys161 Source

gcc (native) compiler

gcc-161 (MIPS) compiler

Sources

Native Binaries

MIPS Binaries

Kernel

sys161
Learning how the code is structured will be a significant but necessary cost.

Use tools to speed up the process

- IDE: Eclipse (for example)
- UNIX tools: grep, find, ctags
  - Should have seen some of these before (CSC 209 / B09)
  - Look up examples and tutorials online and use the man pages
  - grep: search through contents of files
  - find: find files with filenames that include specific keywords (and couple with other options like “exec”)
- Debuggers: gdb (cs161-gdb for OS/161)
Debugging in OS/161 requires two terminals
- Run OS161 in one terminal with the debug flag:
  - `sys161 –w kernel`
- Connect the debugger to it from another terminal:
  - `cs161-gdb kernel`
    > `target remote unix:.sockets/gdb`
- More details are in the Getting Started document

Use `.gdbinit` files to make your life easier!
- GDB executes the files in the `.gdbinit` files upon startup
- You can define blocks of statements to be executed when called at the command line in GDB
  - (example shortly...)
SYSTEM CALLS

- User level applications are isolated from the OS in a different protection domain.
- System calls are a way of requesting the services of the OS.
  - Method calls define in user-level libraries that “trap” from user-space to the OS in kernel-space.
- Once control “traps” to the kernel, a kernel-level system call handler is invoked to service the system call and return the appropriate results to user-space.
- Once the system call is complete, control is given back to the user-level application.
OS161 System Call Example: time

User Space

__time()

syscalls.S

_KERNEL_SPACE

/kernel/arch/mips/locore/exception-mips1.S

asm code to save registers

mips_trap()

syscall()

sys__time()

User Space

/src/user/bin/sh/sh.c (or sbin and testbin pgms)
(Invokes system call)

/src/user/lib/libc/arch/mips/syscalls-mips.S
(auto generated!)
(Sets up syscall number and traps to kernel)

/kernel/arch/mips/locore/trap.c
(trap handler, determines the trap is for a syscall)

/kernel/arch/mips/syscall/syscall.c
(syscall handler, determines which syscall is requested)

/kernel/syscall/time_syscalls.c (time handler)

Key:
C code

call

Assembly

return
Creating a New System Call in OS161

- Define the new system call code in src/kern/include/kern/syscall.h
- Define the prototypes for the new syscall
  - Kernel space: src/kern/include/syscall.h
  - User space: src/user/include/unistd.h
- Write the source code for it in src/kern/syscall/new_syscall.c
  - Be sure to include this new file in src/kern/conf/conf.kern! (so it’s included in the build path)
- If necessary, define any new error codes in src/kern/include/kern/errno.h
- Add a case in the handler switch statement in src/kern/arch/mips/syscall/syscall.c
- Create a test program in src/testbin
- Rebuild both kernel and user level programs
  (Read the documentation on the course website for more details!)