THE UNIVERSITY OF TEXAS AT AUSTIN

WHAT STARTS HERE CHANGES THE WORLD

Implementing system calls in the Y86 model

Soumava Ghosh

System Calls

- The standard method for user mode programs to request services from the OS kernel.
- Execution involves a change in privilege level, and transfer of control to the kernel which executes certain instructions as per the programs requirement and returns control to the user mode (with the desired result).

System Calls (cont.)

open()

- User Mode program calls open()
- Subroutine call to __libc_open

syscall/ sysenter

- __libc_open is the libc wrapper
- Multiple macros expand to invoke assembly code that contains the syscall instruction

…

- Faster transfer of control to kernel (as compared to earlier INT 80H)
- System call id (__NR_open = 3) expected in %eax, other parameters in %edi, %esi, %edx registers.

sysret/sysexit

- Kernel finishes up its work and calls sysret, that returns the control back to the user mode.

Modeling system calls

- Programs that run on the Y86 model are user mode programs
- Need a mechanism to interact with the kernel from ACL2, and retrieve results
- Will not be modeling the kernel mode code that performs the actual action of the system call – our syscall instruction will be a combination of the actual syscall (UM) + sysret (KM).

Options considered

- ACL2 sys-call:
  - Executes system commands as in a shell
  - Not actually a direct system call
  - Does not support interactive input
- CLISP modules:
  - POSIX and OS modules have methods that map to most Unix system calls
  - Preferred Common Lisp for ACL2 is CCL, which wouldn’t have these modules - so a CLISP option wouldn’t be the best solution

Options considered (contd.)

- CCL Foreign Function Interface (FFI)
  - Enables Common Lisp code to call functions outside of Lisp (e.g. C libraries)
  - FFI interface translator provides information about the entry point, argument and return types
  - Lisp data should be copied to a foreign representation before calling the foreign function (eq. string → pointer)
  - Supports interactive console and file input

CLISP Foreign Function Interface

- Load libraries

```
(ccl::open-shared-library “/path/to/library.dylib”)
#<SHLIB /path/to/library.dylib #x30200000E76D>
```

- List entry points

```
(ccl::external “_write”)
#<EXTERNAL-ENTRY-POINT “_write” (#x00007FFF8FBB04A0) /
usr/lib/system/libsystem_kernel.dylib #3020007138CD>
```

- Convert to foreign data types

```
(setq ptr (ccl::make-cstring “xyz”))
#<A Foreign Pointer #x1000E0>
```

CCL FFI (contd.)

- Allocate memory for reading in data

```
(multiple-value-bind (lstr lptr)
    (ccl::make-heap-ivector nbytes '(unsigned-byte 8))
        (setq str lstr)
        (setq ptr lptr))
```

- Invoke entry points

```
(external-call “_write” :unsigned-int 1 :address
ptr :unsigned-int nbytes)
Hello NIL
```

Making system calls from ACL2

- Every POSIX system has the syscall() API defined as below:

```
int syscall (int number, ...);
```

- The first parameter indicates the system call id, and the subsequent variable arguments are mandatory arguments of the particular system call

Making system calls from ACL2 (contd.)

- Observation: It looks like loading the system library is not required as it was found to always be loaded.

- This works out well as the name and location of the system library could differ across systems.

The prototype

- Initial prototype includes the 4 most basic system calls:
  - `int open(const char *path, int oflags)`
  - `int close(int filedes)`
  - `size_t read(int fildes, void *buf, size_t nbytes)`
  - `size_t write(int fildes, const void *buf, size_t nbytes)`

- Raw lisp code was written to use the CCL FFI to invoke the system calls and return the required data in a native lisp format.

The prototype (contd.)

Example: the write() system call

```
;; size_t write(int fildes, const void *buf, size_t nbytes)
(defun syscall-write (clk filedes buffer nbytes)
  (declare (ignore clk))
  (setq ptr  (ccl::make-cstring buffer))
  (setq ret  (ccl::external-call "syscall"
                                 :unsigned-int 1
                                 :unsigned-int filedes
                                 :address ptr
                                 :unsigned-int nbytes
                                 :unsigned-int))
  (ccl::dispose-heap-ivector ptr)
  (cons ret nil))
```

The prototype (contd.)

- The ACL2 interface consists of a stub definition of the same methods as the common lisp ones.

- The model restricts the theorem prover from proving anything else other than the theorems that have been explicitly stated about the stubs. (or more with Matt's latest work)

- Matt will talk about this part in detail later.

Integrating with the Y86 model

- The syscall instruction requires the arrangement of parameters in the following format:
  - System call id in %eax
  - Parameters in %edi, %esi, %edx respectively

- An instruction #xD0 was added to the Y86-basic model. On %eip = #xD0, the y86-step function calls into the y86-syscall method, the syscall handler

Integrating with the Y86 model (contd.)

- The syscall handler reads %eax and calls the appropriate Y86 system call method.

- Example: (Y86-syscall-open clk x86-32)
  - %edi → pointer to file path
  - %esi → flags
  - retrieve the actual null-terminated path string
  - call the ACL2 interaface (syscall-open)
  - set the return value to %eax

Integrating with the Y86 model (contd.)

- Other minor changes:
  - Changes to the recognizer: y86-prog
  - Changes to the byte code writer: y86-asm

```
(defun y86-syscall-open (clk x86-32)
  (declare (xargs :stobjs (x86-32)
                  :guard (natp clk)))
  (b*
   ((pc (eip x86-32))

    ;; Memory Probe
    ((if (< *2^32-2* pc))
     (!ms (list :at-location pc
                :instruction 'syscall-open
                :memory-probe nil
                :reason 'pc-overflow)
          x86-32))

    (path-ptr (rgfi *mr-edi* x86-32))

    ;; Path-ptr sanity check
    ((if (< *mem-size-in-bytes* path-ptr))
     (!ms (list :at-location pc
                :instruction 'syscall-open
                :reason 'ptr-overflow)
          x86-32))
```

```
 (oflags (rgfi *mr-esi* x86-32))
   (path (y86-read-string-null-term x86-32 path-
ptr))
   (ret (syscall-open clk path oflags))

   ;; Save return code to eax
   (x86-32 (!rgfi *mr-eax* (car ret) x86-32))
    (x86-32 (!eip (+ pc 1) x86-32)))
   x86-32))

-------------------------------------------------
---------------------

(defun syscall-open (clk pathname flags)
 (declare (ignore clk))
 (setq ptr
   (ccl::make-cstring pathname))
 (setq ret
   (ccl::external-call "syscall"
                       :unsigned-int 2
                       :address ptr
                       :unsigned-int flags
                       :unsigned-int))
 (cons ret nil))
```

Testing the Y86-basic : CAT

- The most basic CAT implementation in assembly code
  - Hardcoded file name in memory
  - Open the file
  - Read the file, write the bytes read to stdout
  - If the number of bytes read was less than the number of bytes requested, break the loop
  - Close the file

```
(!! cat-code
    '(cat
      (pushl %ebp)       ; Save superior frame pointer
      (rrmovl %esp %ebp) ; Set frame pointer

      (irmovl 2 %eax)    ; Set eax = 2 -> syscall-open
      (irmovl #x3000 %edi) ; Set edi = path ptr
      (xorl %esi %esi)   ; Set esi = 0 (O_RDONLY)
      (syscall)          ; call open to open the file

      (rrmovl %eax %ebx)  ; store the file descriptor

      ;; Loop while !eof
      loop
      (xorl %eax %eax)     ; 0 -> eax
      (rrmovl %ebx %edi)   ; file descriptor to edi
      (irmovl #x4000 %esi) ; buffer location to esi
      (irmovl #x64 %edx)   ; buffer size = 100 to edx
      (syscall)            ; actually call read

      (rrmovl %eax %edx)   ; store readBytes

      (irmovl 1 %eax)      ; 1 -> eax
      (irmovl 1 %edi)      ; file-descriptor of stdout
      (irmovl #x4000 %esi) ; buffer location to esi
      (syscall)            ; syscall-write

      (irmovl #x64 %eax)    ; Move 100 to eax
      (subl %edx %eax)     ; subtract 100 from readBytes
      (je loop)            ; loop if zero
```

```
      ;; Subroutine close and leave
      cat_close_leave
      (irmovl 3 %eax)    ; Set eax = 3
      (rrmovl %ebx %edi) ; file descriptor to edi
      (syscall)          ; syscall-close

      ;; Subroutine leave
      cat_leave
      (rrmovl %ebp %esp) ; restore stack ptr
      (popl %ebp)        ; restore base ptr
      (ret)              ; return from subroutine

      ;; Main
      (align 16)       ; align to 16-byte address
      main             ; main program
      (irmovl stack %esp)  ; initialize stack ptr
      (rrmovl %esp %ebp)   ; initialize base ptr

      (call cat)       ; call the cat function
      (halt)           ; halt the machine

      ;; Stack
      (pos 8192)       ; position 8192
      stack      ; mark this position as 'stack'

      ;; String data
      (pos #x3000)
      (string "//u//soumava//a//ACL2-devel//books//
models//y86//y86-basic//y86//syscalls//systemcalls-
raw.lsp")
      (byte 0)
      ))
```

Demo!