PCSpim I/O Support

Access to Input/Output (I/O) devices within a computer system is generally restricted to prevent user programs from directly accessing them. This prevents a user program from accidentally or maliciously doing things like:

- reading someone else's data file from a disk
- writing to someone else's data file on a disk
- etc.

However, user programs need to perform I/O (e.g., read and write information to files, write to the console, read from the keyboard, etc.) if they are to be useful. Therefore, most computer systems require a user program to request I/O by asking the operating system to perform it on their behalf.

PCSpim uses the "syscall" (short for "system call") instruction to submit requests for I/O to the operating system. The register $v0 is used to indicate the type of I/O being requested with $a0, $a1, $f12 registers being used to pass additional parameters to the operating system. Integer results and addresses are returned in the $v0 register, and floating point results being returned in the $f0 register. The following table provides details of the PCSpim syscall usage.

Service Requested	**System call code passed in $v0**	**Registers used to pass additional arguments**	**Registers used to return results**
print_int	1	$a0 contains the integer value to print	
print_float	2	$f12 contains the 32-bit float to print	
print_double	3	$f12 (and $f13) contains the 64-bit double to print	
print_string	4	$a0 contains the address of the .asciiz string to print	
read_int	5		$v0 returns the integer value read
read_float	6		$f0 returns the 32-bit floating-point value read
read_double	7		$f0 and $f1 returns the 64-bit floating-point value read
read_string	8	$a0 contains the address of the buffer to store the string $a1 contains the maximum length of the buffer	
sbrk - request a memory block	9	$a0 contains the number of bytes in the requested block	$v0 returns the starting address of the block of memory
exit	10		

```
# CalculatePowers subprogram example using MIPS register conventions and PCSpim syscalls

                .data
maxNum:         .word 3
maxPower:       .word 4
str1:           .asciiz " raised to "
str2:           .asciiz " power is "
str3:           .asciiz  "\n"                   # newline character

                .text
                .globl main
main:
                lw      $a0, maxNum             # $a0 contains maxNum
                lw      $a1, maxPower           # $a1 contains maxPower
                jal     CalculatePower

                li      $v0, 10                 # system code for exit
                syscall


################################################################################
CalculatePower:                                 # $a0 contains value of numLimit
                                                # $a1 contains value of powerLimit

                addi    $sp, $sp, -20           # save room for the return address
                sw      $ra, 4($sp)             # push return address onto stack
                sw      $s0, 8($sp)
                sw      $s1, 12($sp)
                sw      $s2, 16($sp)
                sw      $s3, 20($sp)

                move    $s0, $a0                # save numLimit in $s0
                move    $s1, $a1                # save powerLimit in $s1

for_1:
                li      $s2, 1                  # $s2 contains num
for_compare_1:
                bgt     $s2, $s0, end_for_1
for_body_1:

for_2:
                li      $s3, 1                  # $s3 contains pow
for_compare_2:
                bgt     $s3, $s1, end_for_2
for_body_2:
                move    $a0, $s2                # print num
                li      $v0, 1
                syscall
```

```
                la      $a0, str1               # print " raised to "
                li      $v0, 4
                syscall

                move    $a0, $s3                # print pow
                li      $v0, 1
                syscall

                la      $a0, str2               # print " power is "
                li      $v0, 4
                syscall

                move    $a0, $s2                # call Power(num, pow)
                move    $a1, $s3
                jal     Power

                move    $a0, $v0                # print result

                li      $v0, 1
                syscall

                la      $a0, str3               # print new-line character
                li      $v0, 4
                syscall

                addi    $s3, $s3, 1
                j       for_compare_2
end_for_2:

                addi    $s2, $s2, 1
                j       for_compare_1
end_for_1:

                lw      $ra, 4($sp)             # restore return addr. to $ra
                lw      $s0, 8($sp)             # restore saved $s registers
                lw      $s1, 12($sp)
                lw      $s2, 16($sp)
                lw      $s3, 20($sp)

                addi    $sp, $sp, 20            # pop call frame from stack
                jr      $ra
end_CalculatePowers:
```

```
#############################################################################
Power:                                  # $a0 contains n (we never change it during the
                                        #       recursive calls so we don't need to save it)
                                        # $a1 contains e
                addi    $sp, $sp, -4
                sw      $ra, 4($sp)     # save $ra on stack

if:
                bne     $a1, $zero, else_if
                li      $v0, 1          # $v0 contains result
                j       end_if
else_if:
                bne     $a1, 1, else
                move    $v0, $a0
                j       end_if
else:                                   # first parameter is still n in $a0
                addi    $a1, $a1, -1    # put second parameter, e-1, in $a1
                jal     Power           # returns with value of Power(n, e-1) in $v0
                mul     $v0, $v0, $a0   # result = Power(n, e-1) * n
end_if:
                lw      $ra, 4($sp)     # restore return addr. to $ra
                addi    $sp, $sp, 4     # pop call frame from stack
                jr      $ra
end_Power:
```

Snap-shot of the Console window after the program executes: