**MIPS system calls**

(from SPIM S20: A MIPS R2000 Simulator, James J. Larus, University of Wisconsin-Madison)  
  
SPIM provides a small set of operating-system-like services through the MIPS system call (syscall) instruction. To request a service, a program loads the system call code (see Table below) into register $v0 and the arguments into registers $a0, ..., $a3 (or $f12 for floating point values). System calls that return values put their result in register $v0 (or $f0 for floating point results).

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **System Call Code** | **Arguments** | **Result** |
| print integer | 1 | $a0 = value | (none) |
| print float | 2 | $f12 = float value | (none) |
| print double | 3 | $f12 = double value | (none) |
| print string | 4 | $a0 = address of string | (none) |
| read integer | 5 | (none) | $v0 = value read |
| read float | 6 | (none) | $f0 = value read |
| read double | 7 | (none) | $f0 = value read |
| read string | 8 | $a0 = address where string to be stored $a1 = number of characters to read + 1 | (none) |
| memory allocation | 9 | $a0 = number of bytes of storage desired | $v0 = address of block |
| exit (end of program) | 10 | (none) | (none) |
| print character | 11 | $a0 = integer | (none) |
| read character | 12 | (none) | char in $v0 |

For example, to print "the answer = 5", use the commands:

.data

str: .asciiz "the answer = "

.text

li $v0, 4 # $system call code for print\_str

la $a0, str # $address of string to print

syscall # print the string

li $v0, 1 # $system call code for print\_int

li $a0, 5 # $integer to print

syscall # print it

* **print int** passes an integer and prints it on the console.
* **print float** prints a single floating point number.
* **print double** prints a double precision number.
* **print string** passes a pointer to a null-terminated string
* **read int**, **read float**, and **read double** read an entire line of input up to and including a newline.
* **read string** has the same semantics as the Unix library routine fgets. It reads up to n - 1 characters into a buffer and terminates the string with a null byte. If there are fewer characters on the current line, it reads through the newline and again null-terminates the string.
* **sbrk** returns a pointer to a block of memory containing *n* additional bytes.
* **exit** stops a program from running.