COMS20001 lab. worksheet: week #14

S2[A]. Searching for documentation can be hard, because older instances which relate to older kernel and processor types still crop up: this demands care, because the system calling convention changes as newer instance of both emerge.

The following solution assumes use of an x86-based processor (since the majority is written using inline x86 assembly language), *and* that it interacts with (i.e., executes under control of, so makes the system call into) the Linux kernel. At least two possibilities exist:

- A solution for older x86-32 processors and older kernel versions, per

  http://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_32.tbl

  might be

```
int32_t sys_write( int32_t fd, const char* x, int32_t n ) {
  int32_t id = 4, r;

  asm volatile( "movl %1, %%eax ; \
                 movl %2, %%ebx ; \
                 movl %3, %%ecx ; \
                 movl %4, %%edx ; \
                 int  $0x80     ; \
                 movl %%eax, %0 ; "
                : "=g" (r)
                :  "0" (id), "g" (fd), "g" (x), "g" (n)
                : "eax", "ebx", "ecx", "edx" );

  return r;
}

int main( int argc, char* argv[] ) {
  int32_t r = sys_write( 1, "hello world\n", 12 );

  return 0;
}
```

- A solution for newer x86-64 processors and newer kernel versions, per

  http://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl

  might be

```
int64_t sys_write( int64_t fd, const char* x, int64_t n ) {
  int64_t id = 1, r;

  asm volatile( "movq %1, %%rax ; \
                 movq %2, %%rdi ; \
                 movq %3, %%rsi ; \
                 movq %4, %%rdx ; \
                 syscall        ; \
                 movq %%rax, %0 ; "
                : "=g" (r)
                :  "0" (id), "g" (fd), "g" (x), "g" (n)
                : "rax", "rdi", "rsi", "rdx" );

  return r;
}

int main( int argc, char* argv[] ) {
  int64_t r = sys_write( 1, "hello world\n", 12 );

  return 0;
}
```

Although a similar principle and mechanism will probably apply to other processors and kernels, some details may clearly differ. For example,

- the instruction used to invoke a system call will differ for an ARM-based processor, and
- the way the system call identifier and the arguments (i.e., the system calling convention, cf. function calling convention) are communicated will differ on a non-Linux (e.g., Windows) kernel.