# System Calls

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Spring 2023

# System Calls

- OS defines a set of system calls
  - Programming interface to the services provided by OS
  - OS protects the system by rejecting illegal requests
  - OS may impose a quota on a certain resource
  - OS may consider fairness while sharing a resource

- A system call is a _____ procedure call
  - System call routines are in the OS code
  - Executed in the kernel mode
  - On entry, user mode → kernel mode switch
  - On exit, CPU mode is changed back to the user mode

# System Calls Example

- POSIX vs. Win32

| Category | POSIX | Win32 | Description |
|---|---|---|---|
| **Process Management** | fork | CreateProcess | Create a new process |
| | waitpid | WaitForSingleObject | Wait for a process to exit |
| | execve | (none) | CreateProcess = fork + exec |
| | exit | ExitProcess | Terminate execution |
| | kill | (none) | Send a signal |
| **File Management** | open | CreateFile | Create a file or open an existing file |
| | close | CloseHandle | Close a file |
| | read | ReadFile | Read data from a file |
| | write | WriteFile | Write data to a file |
| | lseek | SetFilePointer | Move the file pointer |
| | stat | GetFileAttibutesEx | Get various file attributes |
| | chmod | (none) | Change the file access permission |
| **File System Management** | mkdir | CreateDirectory | Create a new directory |
| | rmdir | RemoveDirectory | Remove an empty directory |
| | link | (none) | Make a link to a file |
| | unlink | DeleteFile | Destroy an existing file |
| | chdir | SetCurrentDirectory | Change the current working directory |
| | mount | (none) | Mount a file system |

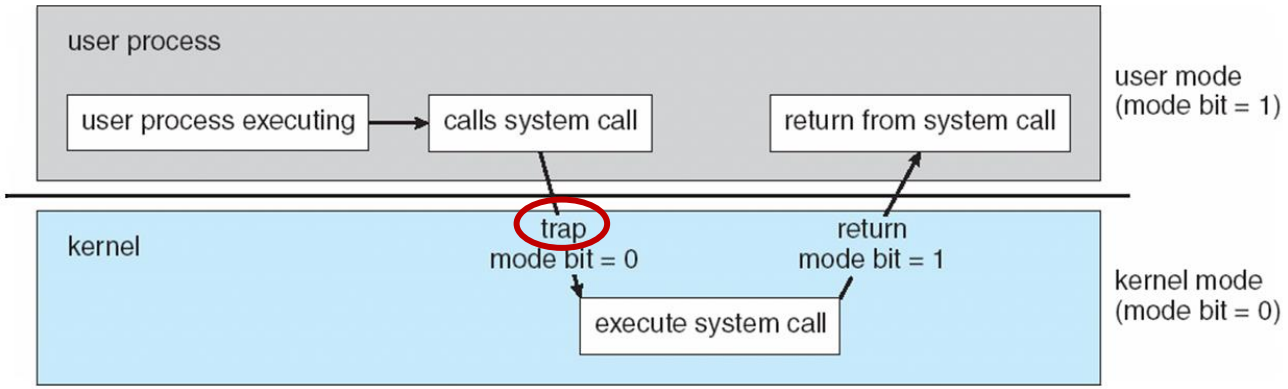
# OS Trap

- There must be a special "trap" instruction that:
  - Causes an exception, which invokes a kernel handler
  - Passes a parameter indicating which system call to invoke
  - Saves caller's state (registers, mode bits)
  - Returns to user mode when done with restoring its state
  - OS must verify caller's parameters (e.g., pointers)

*Examples:*

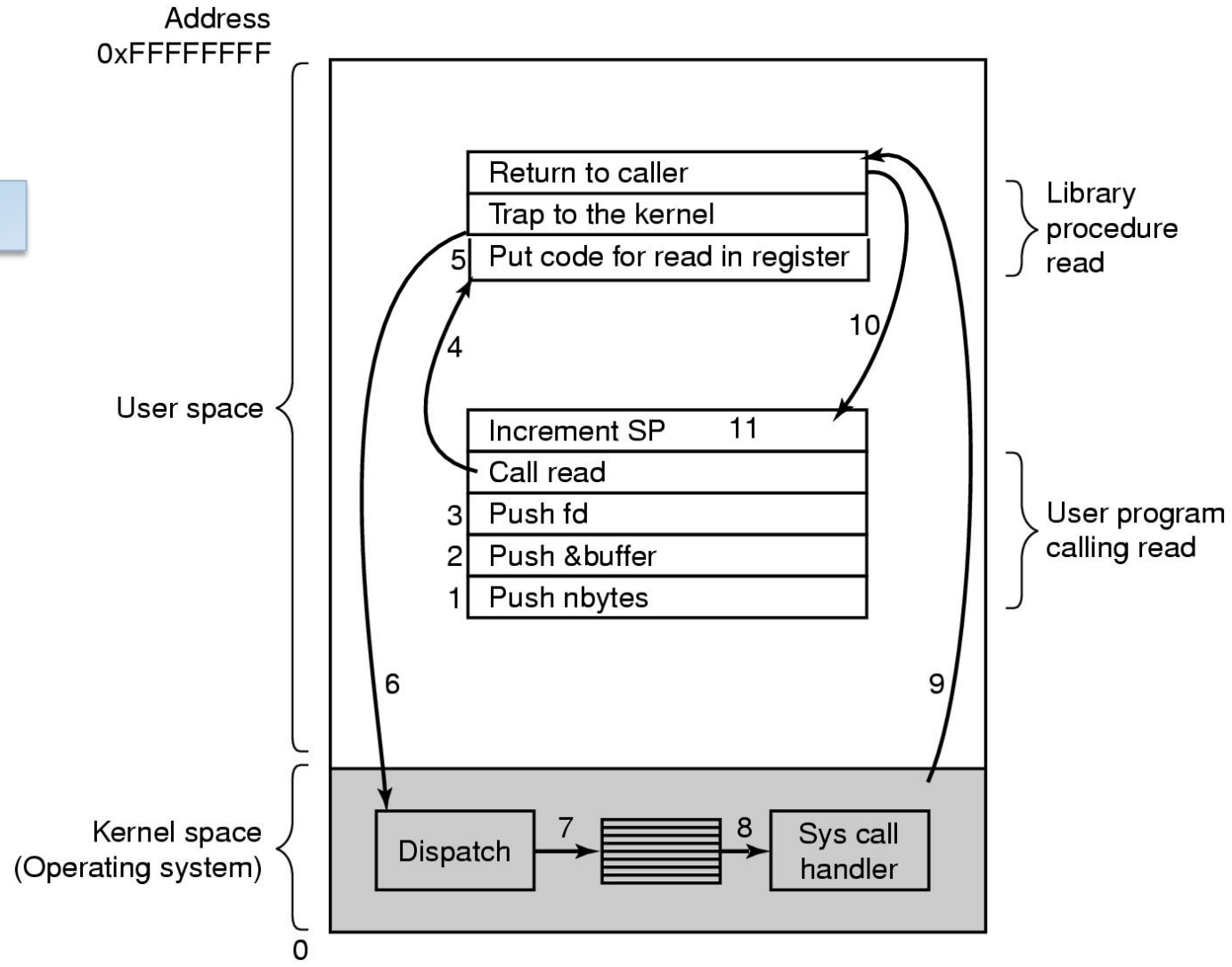
syscall instruction (x86_64)

ecall instruction (RISC-V)

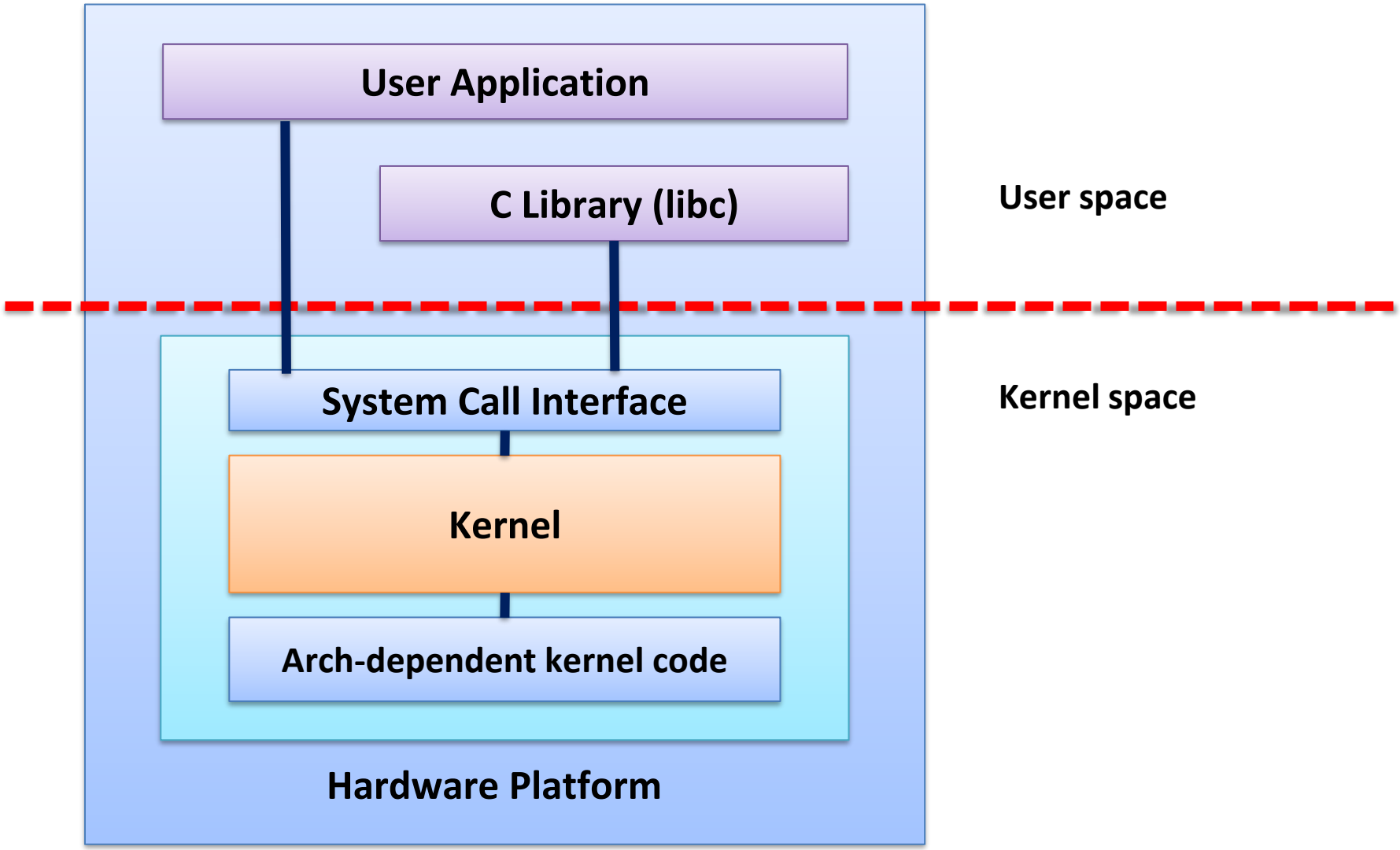


# Implementing System Calls

```
count = read (fd, buffer, nbytes);
```



# Typical OS Structure



# Trap Instructions in x86

- `int 0x80` (+ `iret`)
  - "Software interrupt"
  - A legacy way to invoke a system call (used for 32-bit mode)
  - Slow: use the same mechanism as interrupts
- `sysenter` (+ `sysexit`)
  - A new, fast instruction to invoke a system call in 32-bit mode
  - Introduced by Intel (not available in 64-bit mode on AMD CPUs)
- `syscall` (+ `sysret`)
  - Similar to **`sysenter`**, but used in 64-bit mode
  - Introduced by AMD (not available in 32-bit mode on Intel CPUs)

# syscall

## SYSCALL—Fast System Call

| Opcode | Instruction | Op/ En | 64-Bit Mode | Compat/ Leg Mode | Description |
|---|---|---|---|---|---|
| 0F 05 | SYSCALL | ZO | Valid | Invalid | Fast call to privilege level 0 system procedures. |

- RCX ← RIP
- RIP ← MSR_LSTAR
- R11 ← RFLAGS
- …
- Initialize CS and SS from MSR_STAR
- Set CPL(Current Privilege Level) to 0

# Using a System Call

- Example: `getpid()`

```c
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
  return getpid();
}
```

# Invoking a System Call

- System call number for `getpid()`: `0x27` (= 39)

```
00000000004005a0 <main>:
  4005a0: e9 fb 84 04 00        jmpq   448aa0 <__getpid>
  4005a5: 66 2e 0f 1f 84 00 00  nopw   %cs:0x0(%rax,%rax,1)
  4005ac: 00 00 00
  4005af: 90                    nop
```

```
0000000000448aa0 <__getpid>:
  448aa0: b8 27 00 00 00        mov    $0x27,%eax
  448aa5: 0f 05                 syscall
  448aa7: c3                    retq
  448aa8: 0f 1f 84 00 00 00 00  nopl   0x0(%rax,%rax,1)
  448aaf: 00
```

**In C library**

# Numbering System Calls

- @ arch/x86/entry/syscalls/syscall_64.tbl

```
#
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
#
# The abi is "common", "64" or "x32" for this file.
#
0 common  read      __x64_sys_read
1 common  write     __x64_sys_write
2 common  open      __x64_sys_open
3 common  close     __x64_sys_close
4 common  stat      __x64_sys_newstat

...

37  common  alarm     __x64_sys_alarm
38  common  setitimer   __x64_sys_setitimer
39  common  getpid      __x64_sys_getpid
40  common  sendfile    __x64_sys_sendfile64
```

# Setting up the System Call Entry

- @ arch/x86/kernel/cpu/common.c

```
void syscall_init(void)
{
  wrmsr(MSR_STAR, 0, (__USER32_CS << 16) | __KERNEL_CS);
  wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);
```

# Entering the Kernel

- @ arch/x86/entry/entry_64.S

```
SYM_CODE_START(entry_SYSCALL_64)
  UNWIND_HINT_EMPTY

  swapgs
  /* tss.sp2 is scratch space. */
  movq  %rsp, PER_CPU_VAR(cpu_tss_rw + TSS_sp2)
  SWITCH_TO_KERNEL_CR3 scratch_reg=%rsp
  movq  PER_CPU_VAR(cpu_current_top_of_stack), %rsp

  /* Construct struct pt_regs on stack */
  pushq $__USER_DS        /* pt_regs->ss */
  pushq PER_CPU_VAR(cpu_tss_rw + TSS_sp2) /* pt_regs->sp */
  pushq %r11          /* pt_regs->flags */
  pushq $__USER_CS        /* pt_regs->cs */
  pushq %rcx          /* pt_regs->ip */
SYM_INNER_LABEL(entry_SYSCALL_64_after_hwframe, SYM_L_GLOBAL)
  pushq %rax          /* pt_regs->orig_ax */

  PUSH_AND_CLEAR_REGS rax=$-ENOSYS

  /* IRQs are off. */
  movq  %rax, %rdi
  movq  %rsp, %rsi
  call  do_syscall_64   /* returns with IRQs disabled */
```

# Jumping to the System Call Handler

- @ `arch/x86/entry/common.c`

```
__visible void do_syscall_64(unsigned long nr, struct pt_regs *regs)
{
  struct thread_info *ti;

  enter_from_user_mode();
  local_irq_enable();
  ti = current_thread_info();
  if (READ_ONCE(ti->flags) & _TIF_WORK_SYSCALL_ENTRY)
    nr = syscall_trace_enter(regs);

  if (likely(nr < NR_syscalls)) {
    nr = array_index_nospec(nr, NR_syscalls);
    regs->ax = sys_call_table[nr](regs);
```

# System Call Table

- @ arch/x86/entry/syscall_64.c

```
#define __SYSCALL_64(nr, sym) extern long __x64_##sym(const struct pt_regs *);
#include <asm/syscalls_64.h>
#undef __SYSCALL_64

#define __SYSCALL_64(nr, sym) [nr] = __x64_##sym,

asmlinkage const sys_call_ptr_t sys_call_table[__NR_syscall_max+1] = {
  /*
   * Smells like a compiler bug -- it doesn't work
   * when the & below is removed.
   */
  [0 ... __NR_syscall_max] = &__x64_sys_ni_syscall,
#include <asm/syscalls_64.h>
};
```

# System Call Handlers

- @ arch/x86/include/generated/asm/syscalls_64.h

```
__SYSCALL_COMMON(0, sys_read)
__SYSCALL_COMMON(1, sys_write)
__SYSCALL_COMMON(2, sys_open)
__SYSCALL_COMMON(3, sys_close)
__SYSCALL_COMMON(4, sys_newstat)
__SYSCALL_COMMON(5, sys_newfstat)
__SYSCALL_COMMON(6, sys_newlstat)
__SYSCALL_COMMON(7, sys_poll)
__SYSCALL_COMMON(8, sys_lseek)
__SYSCALL_COMMON(9, sys_mmap)
__SYSCALL_COMMON(10, sys_mprotect)
__SYSCALL_COMMON(11, sys_munmap)
__SYSCALL_COMMON(12, sys_brk)
__SYSCALL_64(13, sys_rt_sigaction)
__SYSCALL_COMMON(14, sys_rt_sigprocmask)
__SYSCALL_64(15, sys_rt_sigreturn)
__SYSCALL_64(16, sys_ioctl)
```

# vDSO

- Virtual Dynamic Shared Object (@ `arch/x86/entry/vdso/`)
  - A small shared library exported by the kernel that is mapped into the address space of all user-space applications
  - Mapped to a different location every time (for security)
  - Used to accelerate the execution of certain read-only system calls ("virtual system calls") without entering the kernel
    - `clock_gettime()`
    - `gettimeofday()`
    - `getcpu()`
    - `time()`
    - `clock_getres()`
  - @ `arch/x86/entry/vdso`
  - `$ man vdso`

# vDSO Layout

```
$ cat /proc/self/maps
5611a0174000-5611a0176000 r--p 00000000 08:05 265118                     /usr/bin/cat
5611a0176000-5611a017b000 r-xp 00002000 08:05 265118                     /usr/bin/cat
5611a017b000-5611a017e000 r--p 00007000 08:05 265118                     /usr/bin/cat
5611a017e000-5611a017f000 r--p 00009000 08:05 265118                     /usr/bin/cat
5611a017f000-5611a0180000 rw-p 0000a000 08:05 265118                     /usr/bin/cat
5611a1ebb000-5611a1edc000 rw-p 00000000 00:00 0                          [heap]
7f9d19bdc000-7f9d19bfe000 rw-p 00000000 00:00 0
7f9d19bfe000-7f9d1ab21000 r--p 00000000 08:05 269248                     /usr/lib/locale/locale-archive
7f9d1ab21000-7f9d1ab46000 r--p 00000000 08:05 274111                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9d1ab46000-7f9d1acbe000 r-xp 00025000 08:05 274111                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9d1acbe000-7f9d1ad08000 r--p 0019d000 08:05 274111                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9d1ad08000-7f9d1ad09000 ---p 001e7000 08:05 274111                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9d1ad09000-7f9d1ad0c000 r--p 001e7000 08:05 274111                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9d1ad0c000-7f9d1ad0f000 rw-p 001ea000 08:05 274111                     /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f9d1ad0f000-7f9d1ad15000 rw-p 00000000 00:00 0
7f9d1ad24000-7f9d1ad25000 r--p 00000000 08:05 273898                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9d1ad25000-7f9d1ad48000 r-xp 00001000 08:05 273898                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9d1ad48000-7f9d1ad50000 r--p 00024000 08:05 273898                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9d1ad51000-7f9d1ad52000 r--p 0002c000 08:05 273898                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9d1ad52000-7f9d1ad53000 rw-p 0002d000 08:05 273898                     /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f9d1ad53000-7f9d1ad54000 rw-p 00000000 00:00 0
7ffe8b35b000-7ffe8b37c000 rw-p 00000000 00:00 0                          [stack]
7ffe8b3f3000-7ffe8b3f6000 r--p 00000000 00:00 0                          [vvar]
7ffe8b3f6000-7ffe8b3f7000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

# vvar and vsyscall

- vvar
  - Mapped just before the vdso page
  - Contains data accessed by virtual system calls
  - Kernel periodically updates the values (if necessary)
  - User-space application can only read the values

- vsyscall
  - A legacy ABI for virtual system calls
  - Mapped to the fixed user-space address
  - Not recommended to use

# syscall()

- A generic library function that performs the specified system call
  - Symbolic constants for system call numbers are specified in `<sys/syscall.h>`
  - Useful when you add a new system call that has no wrapper function in the C library

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

int main(void)
{
  printf("%ld\n", syscall(__NR_getpid));
  printf("%ld\n", syscall(SYS_getpid));
}
```