

# 1 Limited Direct Execution

## Vocabulary

### 1. Time Sharing

- Is a mechanism used by an OS to share a resource
- Allows an entity to use the resource for a little while, and then a little while by another, and so forth

### Example

CPU

### 2. Limited Direct Execution

- Is synonymous to **baby proofing**
- **Limited** Means there will be a limit to what a processor can and cannot do
- **Direct Execution** Means that the processor will run directly on the CPU

### 3. User Mode

- Is a processor mode where code that runs is restricted in what it can do

### 4. Kernel Mode

- Is a processor mode where code that runs can do what it likes, including privileged operations

### Example

Privileged operations include

1. I/O requests
2. Executing all types of restricted instructions

### 5. System Call

- Is a programmatic way in which a computer program requests a privileged service from the kernel of the operating system

### 6. Trap

- Is a type of synchronous interrupt caused by an exceptional condition that
- Exceptional condition include:

- Breakpoint
- Division by zero
- Invalid memory access
- System Call
- Usually results in a processor switching to kernel mode

## 7. Return-from-Trap

- Is an instruction that
  - Restores saved registers from kernel stack
  - Switches the processor back to **user mode**

## 8. Trap Table

**Question** What is the exact definition of a trap table? OSTEP glosses over it :(

- Is synonymous to 대응 메뉴얼
- Is a list of trap handlers where each is associated with a specific trap

## 9. Trap Handlers

- Is the code that will run when the trap is triggered.

## 10. System-call Number

- Is an ID assigned to each system call

### Linux System Call Table

The following table lists the system calls for the Linux 2.2 kernel. It could also be thought of as an API for the interface between user space and kernel space. My motivation for making this table was to make programming in assembly language easier when using only system calls and not the C library (for more information on this topic, go to <http://www.linuxassembly.org>). On the left are the numbers of the system calls. This number will be put in register %eax. On the right of the table are the types of values to be put into the remaining registers before calling the software interrupt 'int 0x80'. After each syscall, an integer is returned in %eax.

For convenience, the kernel source file where each system call is located is linked to in the column labelled "Source". In order to use the hyperlinks, you must first copy this page to your own machine because the links take you directly to the source code on your system. You must have the kernel source installed (or linked from) under '/usr/src/linux' for this to work.

System Call  
Number

| %eax | Name        | Source                                     | %ebx           | %ecx                              | %edx                   | %esi | %edi |
|------|-------------|--------------------------------------------|----------------|-----------------------------------|------------------------|------|------|
| 1    | sys_exit    | <a href="#">kernel/exit.c</a>              | int            | -                                 | -                      | -    | -    |
| 2    | sys_fork    | <a href="#">arch/i386/kernel/process.c</a> | struct pt_regs | -                                 | -                      | -    | -    |
| 3    | sys_read    | <a href="#">fs/read_write.c</a>            | unsigned int   | char *                            | <a href="#">size_t</a> | -    | -    |
| 4    | sys_write   | <a href="#">fs/read_write.c</a>            | unsigned int   | const char *                      | <a href="#">size_t</a> | -    | -    |
| 5    | sys_open    | <a href="#">fs/open.c</a>                  | const char *   | int                               | -                      | -    | -    |
| 6    | sys_close   | <a href="#">fs/open.c</a>                  | unsigned int   | -                                 | -                      | -    | -    |
| 7    | sys_waitpid | <a href="#">kernel/exit.c</a>              | pid_t          | unsigned int *                    | int                    | -    | -    |
| 8    | sys_creat   | <a href="#">fs/open.c</a>                  | const char *   | int                               | -                      | -    | -    |
| 9    | sys_link    | <a href="#">fs/namei.c</a>                 | const char *   | const char *                      | -                      | -    | -    |
| 10   | sys_unlink  | <a href="#">fs/namei.c</a>                 | const char *   | -                                 | -                      | -    | -    |
| 11   | sys_execve  | <a href="#">arch/i386/kernel/process.c</a> | struct pt_regs | -                                 | -                      | -    | -    |
| 12   | sys_chdir   | <a href="#">fs/open.c</a>                  | const char *   | -                                 | -                      | -    | -    |
| 13   | sys_time    | <a href="#">kernel/time.c</a>              | int *          | -                                 | -                      | -    | -    |
| 14   | sys_mknod   | <a href="#">fs/namei.c</a>                 | const char *   | int                               | <a href="#">dev_t</a>  | -    | -    |
| 15   | sys_chmod   | <a href="#">fs/open.c</a>                  | const char *   | <a href="#">mode_t</a>            | -                      | -    | -    |
| 16   | sys_lchown  | <a href="#">fs/open.c</a>                  | const char *   | <a href="#">uid_t</a>             | <a href="#">gid_t</a>  | -    | -    |
| 18   | sys_stat    | <a href="#">fs/stat.c</a>                  | char *         | <a href="#">old_kernel_stat *</a> | -                      | -    | -    |

## 11. Timer Interrupt

- Is a type of interrupt generated by an internal clock instead of an external event (e.g I/O or system call)

## 12. Interrupt Handler

- Is a special block of code associated with a specific interrupt condition

### 1.1 Direct Execution

- Means just run program directly without limits
- Has advantage of being fast

| OS                            | Program                         |
|-------------------------------|---------------------------------|
| Create entry for process list |                                 |
| Allocate memory for program   |                                 |
| Load program into memory      |                                 |
| Set up stack with argc/argv   |                                 |
| Clear registers               |                                 |
| Execute <b>call</b> main()    |                                 |
|                               | Run main()                      |
|                               | Execute <b>return</b> from main |
| Free memory of process        |                                 |
| Remove from process list      |                                 |

### 1.2 Problem #1: Restricted Operations

- **Question:** How can the OS make sure a program doesn't do anything that we don't want to do while running it efficiently?

### 1.3 Problem #2: Switching Between Processes

- **Question:** When we are running a process, how does the operating system stop it from running and switch to another process, thus implementing **time sharing** mechanism to virtualize CPU?

### 1.4 Concurrency