System Calls

Lab 5

Linux System Calls

- System calls are low level functions the operating system makes available to applications via a defined API (Application Programming Interface)
- System calls represent the *interface* the kernel presents to user applications.
- In Linux all low-level I/O is done by reading and writing file handles, regardless of what particular peripheral device is being accessed—a tape, a socket, even your terminal, they are all *files.*
- Low level I/O is performed by making *system calls.*

Anatomy of a System Call

- A System Call is an explicit request to the kernel made via a software interrupt.
- The interrupt call ‘0x80’ call to a system call handler (sometimes called the “call gate”).
- The system call handler in turns calls the system call interrupt service routine (ISR).
- To perform Linux system calls we have to do following:
  - Put the system call number in `EAX` register.
  - Set up the arguments to the system call in `EBX,ECX, etc.`
  - call the relevant interrupt (for DOS, `21h; for Linux, 80h).`
  - The result is usually returned in `EAX.`

- There are six registers that are used for the arguments that the system call takes. The first argument goes in EBX, the second in ECX, then EDX, ESI, EDI, and finally EBP. If more then 6 arguments needed (not likely), the EBX register must contain the memory location where the list of arguments is stored.
- We will learn 5 basic system calls:
  - sys_open
  - sys_close
  - sys_read
  - sys_write
  - sys_lseek.
- Files (in Linux everything is a file) are referenced by an integer file descriptor.

1. **Sys_open** - open a file

- system call number (in EAX): 5
- arguments:
  - EBX: The pathname of the file to open/create
  - ECX: set file access bits (can be OR’d togather):
    - O_RDONLY open for reading only
    - O_WRONLY open for writing only
    - O_RDRW open for both reading and writing
    - O_APPEND open for appending to the end of file
    - O_TRUNC truncate to 0 length if file exists
    - O_CREAT create the file if it doesn’t exist
  - EDX: set file permissions.
- Returns in EAX: file descriptor.
- On errors: -1.

2. **Sys_close** - close a file by file descriptor reference

- system call number (in EAX): 6
- arguments:
  - EBX: file descriptor.
- Returns in EAX: -
- Errors: -1.

3. **Sys_read** - read up to count bytes from file descriptor into buffer

- system call number (in EAX): 3
- arguments:
  - EBX: file descriptor.
  - ECX: pointer to input buffer.
  - EDX: buffer size, max. count of bytes to receive.
- Returns in EAX: number of bytes received.
- On Errors: -1 or 0 (no bits read).

4. **<u>Sys_write</u>** - write (up to) count bytes of data from buffer to file descriptor reference.

- system call number (in EAX): 4
- arguments:
  - EBX: file descriptor.
  - ECX: pointer to output buffer.
  - EDX: count of bytes to send.
- Returns in EAX: number of bytes send.
- On Errors: -1 or 0 (no bits written).

5. **<u>Sys_lseek</u>** - change file pointer.

- system call number (in EAX): 19
- arguments:
  - EBX: file descriptor.
  - ECX: offset, given in number from the following parameter.
  - EDX: either one of
    - SEEK_SET 0 - beginning of file.
    - SEEK_CUR 1 - current position.
    - SEEK_END 2 - end of file.
- Returns in EAX: current file pointer position.
- On Errors: beginning of file position.

Error handling

- System calls set a *global* integer called errno on error.
- The constants that errno may be set to are (partial list):
  - EPERM operation not permitted.
  - ENOENT no such file or directory (not there).
  - EIO I/O error – EEXIST file already exists.
  - ENODEV no such device exists.
  - EINVAL invalid argument passed.