

System Calls

Lab 4

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 .

How we'll use system calls

- `system_call(arg1,arg2,arg3,arg4).`
- Where the first arg goes to EAX
the second to EBX
the third to ECX
and the fourth to EDX
- This function is made for to at start.c

system_call function

- system_call:
- push ebp ; Save caller state
- mov ebp, esp
- sub esp, 4 ; Leave space for local var on stack
- pushad ; Save some more caller state
- mov eax, [ebp+8] ; Copy function args to registers: leftmost...
- mov ebx, [ebp+12] ; Next argument...
- mov ecx, [ebp+16] ; Next argument...
- mov edx, [ebp+20] ; Next argument...
- int 0x80 ; Transfer control to operating system
- mov [ebp-4], eax ; Save returned value...
- popad ; Restore caller state (registers)
- mov eax, [ebp-4] ; place returned value where caller can see it
- add esp, 4 ; Restore caller state
- pop ebp ; Restore caller state
- ret ; Back to caller

- 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 (arg1): 5
- arguments:
 - arg2: The pathname of the file to open/create
 - arg3: set file access bits (can be OR'd together):
 - 0 = O_RDONLY open for reading only
 - 1 = O_WRONLY open for writing only
 - 2 = O_RDWR open for both reading and writing
 - 1024 = O_APPEND open for appending to the end of file
 - 512 = O_TRUNC truncate to 0 length if file exists
 - 64 = O_CREAT create the file if it doesn't exist
 - arg4: set file permissions.
- Returns : file descriptor.
- On errors: -1.

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

- system call number (arg1): 6
- arguments:
 - arg2: file descriptor.
- Errors: -1.

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

- system call number (arg1): 3
- arguments:
 - arg2: file descriptor.
 - arg3: pointer to input buffer.
 - arg4: buffer size, max. count of bytes to receive.
- Returns : number of bytes received.
- On Errors: -1 or 0 (no bits read).

4. **Sys write** - write (up to) count bytes of data from buffer to file descriptor reference.

- system call number (arg1): 4
- arguments:
 - arg2: file descriptor.
 - arg3: pointer to output buffer.
 - arg4: count of bytes to send.
- Returns : number of bytes send.
- On Errors: -1 or 0 (no bits written).

5. Sys_lseek - change file pointer.

- system call number (arg1): 19
- arguments:
 - arg2: file descriptor.
 - arg3: offset, given in number from the following parameter.
 - arg4: either one of
 - SEEK_SET 0 - beginning of file.
 - SEEK_CUR 1 - current position.
 - SEEK_END 2 - end of file.
- Returns : 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.