# MODUL SISTEM OPERASI

# (CCI210)

## MODUL SESI 4

## SYSTEM CALL

**DISUSUN OLEH**

**ADI WIDIANTONO, SKOM, MKOM.**

**UNIVERSITAS ESA UNGGUL**

**2020**

# SYSTEM CALL

## A. Kemampuan Akhir Yang Diharapkan

Setelah mempelajari modul ini, diharapkan mahasiswa mampu :

1. Memahami konsep dasar dari system call pada system operasi.
2. Memahami cara kerja system call.
3. Mamahami dan mengerti menerapkan system call pada program.

## B. Uraian dan Contoh

## 1. System Call

Menurut Onno Purbo, system call (biasa di kenal sebagai "syscall") adalah sebuah instruksi, mirip dengan instruksi "add" atau "jump". Pada tingkat tinggi, sebuah system call adalah cara sebuah program pada level user untuk meminta pada sistem operasi untuk menjalankan sesuatu untuknya. Jika kita seorang programmer, dan kita membutuhkkan untuk membaca dari sebuah file, kita akan menggunakan system call untuk meminta sistem operasi untuk membaca file tersebut untuk kita.

Kita telah melihat bahwa sistem operasi memiliki dua fungsi utama: menyediakan abstraksi untuk program pengguna dan mengelola sumber daya komputer. Untuk sebagian besar bagian, interaksi antara program pengguna dan sistem operasi berhubungan dengan bekas; misalnya membuat, menulis, membaca, dan menghapus file. Bagian manajemen sumber daya sebagian besar transparan bagi pengguna dan dilakukan secara otomatis.

Jadi, antarmuka antara program pengguna dan sistem operasi pada dasarnya tentang berurusan dengan abstraksi. Untuk benar-benar memahami sistem operasi apa lakukan, kita harus memeriksa antarmuka ini dengan cermat. Panggilan sistem (system call) yang tersedia di antarmuka bervariasi dari satu sistem operasi ke sistem operasi lainnya (meskipun konsep dasarnya cenderung mirip).

Komputer CPU tunggal mana pun hanya dapat menjalankan satu instruksi pada satu waktu. Jika suatu proses menjalankan program pengguna dan membutuhkan layanan sistem, seperti membaca data dari file, harus dijalankan melalui instruksi perangkap untuk mentransfer kontrol ke sistem operasi. Sistem operasi kemudian mencari tahu apa yang diinginkan proses pemanggilan dengan memeriksa parameter. Kemudian melakukan ***system call*** dan mengembalikan kontrol ke instruksi yang mengikuti system call. Dalam arti tertentu, membuat system call seperti membuat jenis panggilan prosedur (procedure call) khusus, hanya system call yang masuk ke kernel dan procedure call tidak.

System Call biasanya tersedia sebagai instruksi bahasa assembly. Beberapa sistem mengizinkan system calls dibuat langsung dari program bahasa tingkat tinggi. Beberapa bahasa pemrograman (contoh: C, C++) telah didefenisikan untuk menggantikan bahasa assembly untuk sistem pemrograman.

Tiga metoda umum yang digunakan dalam memberikan parameter kepada sistem operasi:

- Melalui register.
- Menyimpan parameter dalam block atau tabel pada memori dan alamat block tersebut diberikan sebagai parameter dalam register.
- Menyimpan parameter (push) ke dalam stack oleh program, dan melakukan pop off pada stack oleh sistem operasi.

**2. Cara kerja system call**

Cara system call bekerja adalah sebagai berikut. Pertama-tama, user program akan menyiapkan (set-up) argument/parameter untuk system call. Salah satu argumen adalah nomor system call. Perlu di catat bahwa semua ini dilakukan secara automatis oleh fungsi library kecuali jika kita menulis menggunakan bahasa assembler. Sesudah semua argumen di setup, program akan menjalankan instruksi "system call". Instruksi ini akan menyebabkan exception: event yang akan menyebabkan processor untuk jump ke satu address dan mulai menjalankan program / code di address tersebut.

Instruksi di alamat yang baru akan menyimpan state user program, menentukan sistem call apa yang kita inginkan, kemudian call fuction tersebut di kernel yang mengimplementasikan system call, setelah selesai maka mengembalikan program state, dan kembali ke user program. Sebuah system call adalah salah satu cara agar function yang di definisikan dalam device driver untuk bisa di panggil.

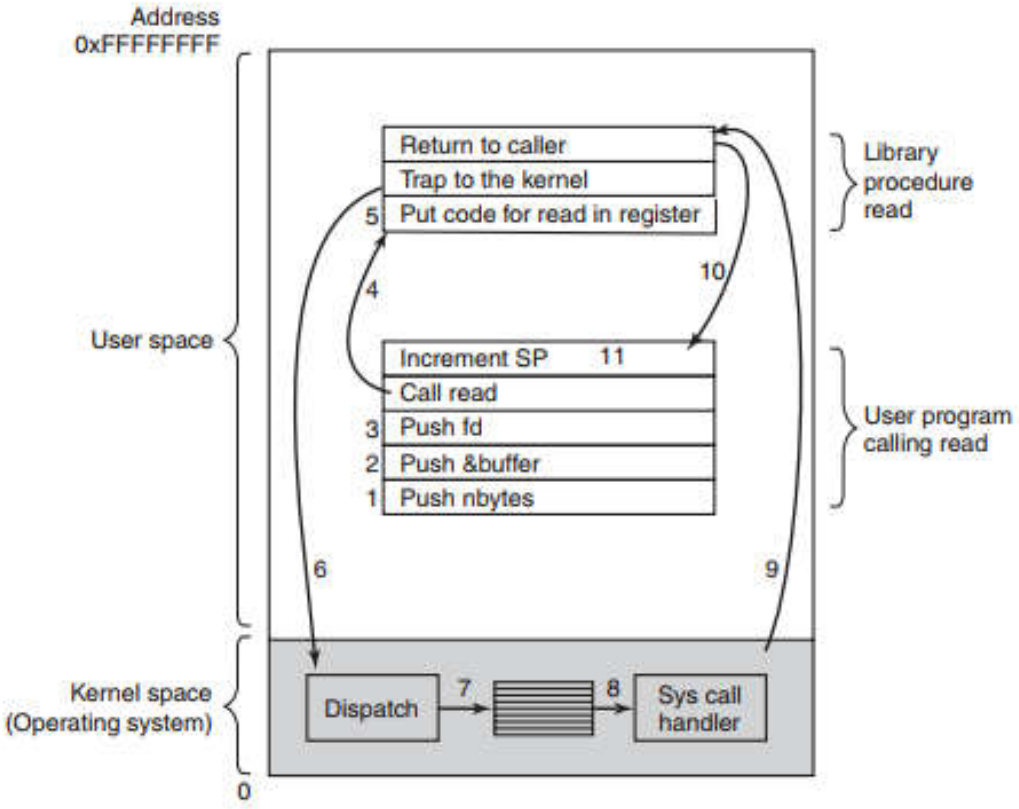
Untuk membuat mekanisme panggilan sistem lebih jelas, coba lihat sekilas tentang *read system call*. Seperti disebutkan di atas, ini memiliki tiga parameter/argumen: yang pertama menentukan file, yang kedua menunjuk ke buffer, dan yang ketiga memberikan nomor byte untuk dibaca. Seperti hampir semua system call, dipanggil dari program C dengan memanggil prosedur dari *library* dengan nama yang sama dengan *system call: read*. Panggilan dari program C mungkin terlihat seperti ini:

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

Sistem Call (dan *library procedure*) mengembalikan jumlah byte sebenarnya yang terbaca kedalam *count*. Nilai ini biasanya sama dengan nbytes, tetapi mungkin lebih kecil, misalnya, akhir file ditemukan saat membaca.

Jika panggilan sistem tidak dapat dilakukan karena parameter atau disk tidak valid/error, count diisi −1, dan nomor error dimasukkan ke dalam variabel global, errno. Program harus selalu memeriksa hasil panggilan sistem untuk melihat apakah terjadi kesalahan.

Langkah yang dilakukan dapat dilihat pada gambar berikut:




Gambar 1.1 . The 11 steps in making the system call read(fd, buffer, nbytes).

### 3. Jenis System Call

System calls yang berhubungan dengan kontrol proses antara lain ketika penghentian pengeksekusian program. Baik secara normal (*end*) maupun tidak normal (*abort*). Selama proses dieksekusi kadang kala diperlukan untuk me-load atau mengeksekusi program lain, disini diperlukan lagi suatu system calls. Juga ketika membuat suatu proses baru dan menghentikan sebuah proses.

Ada juga system calls yang dipanggil ketika kita ingin meminta dan merubah atribut dari suatu proses. MS-DOS adalah contoh dari sistem *single-tasking*. MS-DOS menggunakan metoda yang sederhana dalam menjalankan program aan tidak menciptakan proses baru. Program di-*load* ke dalam memori, kemudian program dijalankan. Berkeley Unix adalah contoh dari sistem *multi-tasking. Command Interpereter* masih tetap bisa dijalankan ketika program lain dieksekusi.

## 4. Kelompok System Call

System Call dapat dikelompokan berdasarkan fungsi sbb:

**System Call Process Control/Management Process**

System Call untuk managemen proses diperlukan untuk mengatur proses-proses yang sedang berjalan, contoh dari system call untuk fungsi ini:

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

**System Call untuk File Management**

System calls yang berhubungan dengan berkas sangat diperlukan. Seperti ketika kita ingin membuat atau menghapus suatu berkas. Atau ketika ingin membuka atau menutup suatu berkas yang telah ada, membaca berkas tersebut, dan menulis berkas itu. System calls juga diperlukan ketika kita ingin mengetahui atribut dari suatu berkas atau ketika kita juga ingin merubah atribut tersebut. Yang termasuk atribut berkas adalah nama berkas, jenis berkas, dan lain-lain. Ada juga system calls yang menyediakan mekanisme lain yang berhubungan dengan direktori atau sistim berkas secara keseluruhan. Jadi bukan hanya berhubungan dengan satu spesifik berkas. create file, delete file.

Contoh system call untuk File Management:

- open, close
- read, write, reposition
- get file attributes, set file attributes

**System call untuk Device Management**

Program yang sedang dijalankan kadang kala memerlukan tambahan sumber daya. Jika banyak pengguna yang menggunakan sistem, maka jika memerlukan tambahan sumber daya maka harus meminta peranti/device terlebih dahulu. Dan setelah selesai penggunakannnya harus dilepaskan kembali. Ketika sebuah peranti telah diminta dan dialokasikan maka peranti tersebut bisa dibaca, ditulis, atau direposisi.

Contoh system call:

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

**System Call untuk Information Maintenance**

Beberapa system calls disediakan untuk membantu pertukaran informasi antara pengguna dan sistem operasi. Contohnya system calls untuk meminta dan mengatur waktu dan tanggal. Atau meminta informasi tentang sistem itu sendiri, seperti jumlah pengguna, jumlah memori dan disk yang masih bisa digunakan, dan lain-lain. Ada juga system calls untuk meminta informasi tentang proses yang disimpan oleh sistem dan system calls untuk merubah (reset) informasi tersebut.

Contoh system call:

- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

**System Call untuk Communications**

Dua model komunikasi:

- Message-passing Pertukaran informasi dilakukan melalui fasilitas komunikasi antar proses yang disediakan oleh sistem operasi.

- Shared-memory Proses menggunakan memori yang bisa digunakan oleh berbagai proses untuk pertukaran informasi dengan membaca dan menulis data pada memori tersebut.

Dalam *message-passing*, sebelum komunikasi dapat dilakukan harus dibangun dulu sebuah koneksi. Untuk itu diperlukan suatu system calls dalam pengaturan koneksi tersebut, baik dalam menghubungkan koneksi tersebut maupun dalam memutuskan koneksi tersebut ketika komunikasi sudah selesai dilakukan. Juga diperlukan suatu system calls untuk membaca dan menulis pesan (message) agar pertukaran informasi dapat dilakukan

Contoh system call:

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

## 5. Contoh system Call

Saat kita menjalankan system call, processor akan jump ke address 0xc00. Code pada lokasi ini di definisikan pada file:

arch/ppc/kernel/head.S

Tertampil :

```
/* System call */
      . = 0xc00
SystemCall:
      EXCEPTION_PROLOG
      EXC_XFER_EE_LITE(0xc00, DoSyscall)

/* Single step - not used on 601 */
     EXCEPTION(0xd00, SingleStep, SingleStepException, EXC_XFER_STD)
      EXCEPTION(0xe00, Trap_0e, UnknownException, EXC_XFER_EE)
```

Yang akan dilakukan oleh code ini adalah menyimpan state program, dan call function DoSyscall.

Berikut adalah penjelasan lebih detail:

- EXCEPTION_PROLOG adalah sebuah macro yang akan menangani switch dari user ke kernel space, yang akan melakukan hal seperti menyimpan kondisi register dari proses user.
- EXC_XFER_EE_LITE akan dipanggil menggunakan address dari route tersebut, dan address dari function DoSyscall. Pada suatu saat, register akan di simpan dan DoSyscall akan di panggil. Dua kalimat selanjutnya adalah exception vector pada address 0xd00 dan 0xe00.

**EXC_XFER_EE_LITE** akan tampak sebagai berikut:

```
#define EXC_XFER_EE_LITE(n, hdlr)       \
        EXC_XFER_TEMPLATE(n, hdlr, n+1, COPY_EE, transfer_to_handler, \
                        ret_from_except)
```

**EXC_XFER_TEMPLATE** adalah macro dan code akan tampak sebagai berikut:

```
#define EXC_XFER_TEMPLATE(n, hdlr, trap, copyee, tfer, ret)     \
        li      r10,trap;                                       \
        stw     r10,TRAP(r11);                                  \
        li      r10,MSR_KERNEL;                                 \
        copyee(r10, r9);                                        \
        bl      tfer;                                           \
i##n:                                                           \
        .long   hdlr;                                           \
        .long   ret
```

Penjelasan

li singkatan dari "load immediate", yang berarti nilai kontanta yang diketahui saat waktu compile di simpan di register. Pertama kali, trap di load ke register r10. Di

kalimat selanjutnya, nilai tersebut di simpan ke address yang diberikan oleh TRAP(r11).

TRAP(r11) dan dua kalimat selanjutnya melakukan manipulasi bit spesifik, yang pada dasarnya melakukan housekeeping, dan kemudian men-transfer kontrol ke register, oleh karenanya kita akan melihat .long DoSyscall bukan bl DoSyscall.

## 6. System Call Windows (Win32 API)

Program Windows biasanya digerakkan oleh *event*. Program utama menunggu beberapa *event* terjadi, lalu memanggil prosedur untuk menanganinya. *Event* itu seperti tombol keyboard ditekan, mouse digerakkan, tombol mouse ditekan, atau drive USB dimasukkan. Prosedur penangan kemudian dipanggil untuk memproses *event* tersebut, memperbarui layar dan memperbarui status program internal. Dan tentunya Windows memiliki system call. Pertama-tama, libray call dan system call yang sebenarnya sangat dipisahkan. Microsoft telah menetapkan satu set prosedur yang disebut Win32 API (Application Programming Interface), dimana pemrogram diharapkan menggunakan untuk mendapatkan layanan sistem operasi. Antarmuka ini (sebagian) didukung pada semua versi Windows sejak Windows 95. Dengan memisahkan antarmuka API dari panggilan sistem yang sebenarnya, Microsoft mempertahankan kemampuan untuk mengubah *system call* aktual dalam waktu (bahkan dari rilis ke rilis) tanpa membatalkan program yang ada. Apa yang sebenarnya merupakan Win32 juga sedikit ambigu karena versi Windows terbaru memiliki banyak panggilan baru yang sebelumnya tidak tersedia. Di bagian ini, Win32 berarti antarmuka yang didukung oleh semua versi Windows. Win32 menyediakan kompatibilitas di antara versi Windows.

Jumlah API Win32 call sangat besar, berjumlah ribuan. Selain itu, sementara banyak dari mereka melakukan system call, sejumlah besar dilakukan seluruhnya di ruang pengguna. Akibatnya, dengan Windows itu tidak mungkin untuk melihat apakah itu system call (yaitu, dilakukan oleh kernel) atau hanya library call di *user space*. Sebenarnya, system call dalam satu versi Windows dapat dilakukan di user space dan sebaliknya dalam versi yang berbeda.

Win32 API memiliki sejumlah besar panggilan untuk mengelola jendela/window, gambar geometris, teks, font, scrollbar, kotak dialog, menu, dan fitur GUI lainnya. Sejauh subsistem grafis berjalan di kernel (pada beberapa versi Windows tetapi tidak pada semua), ini adalah system call; kalau tidak, mereka hanya library call.

Perbandingan Sistem Call Windows dan Unix

| UNIX | Win32 | Description |
|---|---|---|
| fork | CreateProcess | Create a new process |
| waitpid | WaitForSingleObject | Can wait for a process to exit |
| execve | (none) | CreateProcess = fork + execve |
| exit | ExitProcess | Terminate execution |
| 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 | GetFileAttributesEx | Get various file attributes |
| mkdir | CreateDirectory | Create a new directory |
| rmdir | RemoveDirectory | Remove an empty directory |
| link | (none) | Win32 does not support links |
| unlink | DeleteFile | Destroy an existing file |
| mount | (none) | Win32 does not support mount |
| umount | (none) | Win32 does not support mount, so no umount |
| chdir | SetCurrentDirectory | Change the current working directory |
| chmod | (none) | Win32 does not support security (although NT does) |
| kill | (none) | Win32 does not support signals |
| time | GetLocalTime | Get the current time |

C. **Daftar Pustaka**


1. Modern Operating System 4th Ed. Andrew S Tanembaun 2009
2. Pengantar Sistem Operasi Komputer: Plus Ilustrasi Kernel Linux, Masyarakat Digital Gotong Royong (MDGR), 2005