

SYSTEM CALL

System Call adalah penyedia antarmuka dari pelayanan-pelayanan yang tersedia dengan **Sistem Operasi**.

Dalam pemrograman terdapat beberapa proses ketika suatu file sumber sampai ke file tujuan. Hal ini bisa dilihat pada gambar dibawah ini.

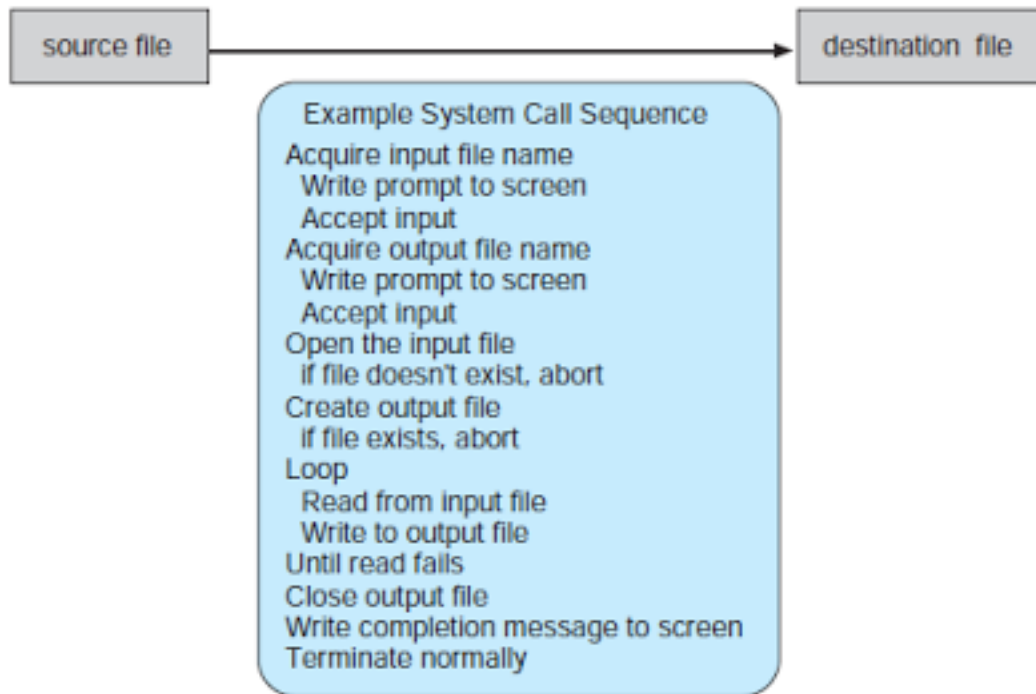
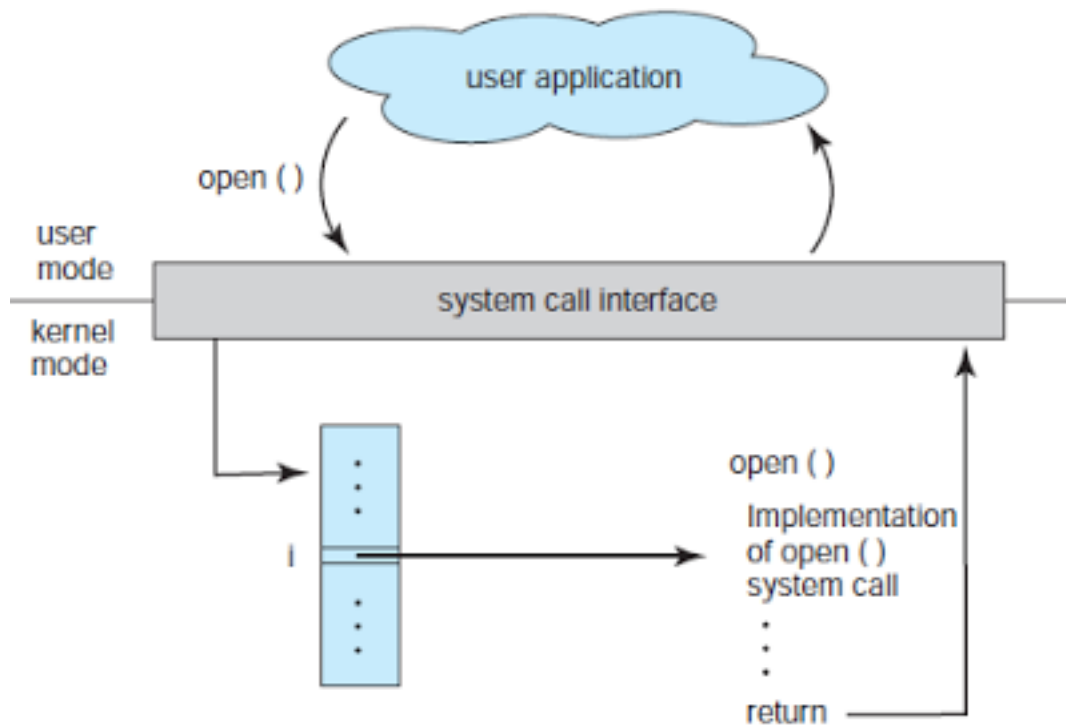


Figure 2.1 Example of how system calls are used.

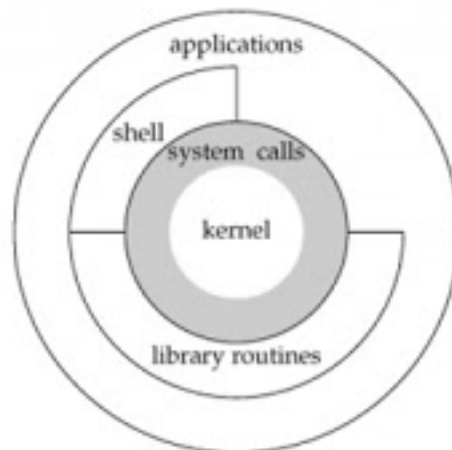
Misalkan pembacaan suatu file, maka dalam program akan membutuhkan nama dari 2 file input dan output. Dengan proses sbb : memasukkan nama file input dan menampilkannya pada layar, menerima masukan seperti inputan data dari keyboard yang diketik, dan nama file output hasil dari ketikkan kita. Setelah dua nama file telah diperoleh program harus membuka file input dan membuat file output.

Masing-masing membutuhkan system call. Terdapat command interpreter ([baca mengenai command interpreter](#)) yang terdapat di OS kita dan membukakan file tersebut. Programming biasanya menyediakan library untuk masing-masing system call yang dibutuhkan. Misal akses file terdapat library `open()`, `read()` dan `write()`. Gambar dibawah ini adalah contoh proses akses file.



Jenis-jenis System Call:

- Process control: mengontrol proses yang berjalan
- File management: manage file-file yang berjalan pada program
- Device management: manage device apa saja yang digunakan pada program
- Information Maintenance: sebagai penghubung antara user dengan sistem operasi dari berbagai informasi.
- Communication: pertukaran informasi dari proses yang berjalan dengan sistem operasi.



Gambar diatas adalah gambaran posisi system call yang terdapat dalam sistem operasi UNIX.

Percobaan kali ini kita akan mencoba beberapa syscall yang ada di linux, yaitu :
eksekusi file : `exec()` dan Variannya yaitu `execl()`, `execvp()`, `execle()`, `execv()`, `execvp()`, dan `execvpe()`.

Ketikkan kembali program berikut :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    printf ("Parent: PID proses parent = %d \n",getpid());
    printf("Parent: Running ps menggunakan execlp \n");
    execlp("ps","ps","ax",NULL);
    sleep(10);
    exit(0);
}
```

Amati hasilnya dan buat analisa apa maksudnya perintah `execlp` diatas. Dan bagaimana cara kerjanya ?

Ketikkan program dibawah ini, buat analisa apa maksudnya perintah `execl` dan bagaimana cara kerjanya ?

```
#include <stdio.h>
#include <unistd.h>
int main () {
    execl ("/bin/ls", /* path program */
          "ls", /* nama program yang dikirim ke argument string argv[0] */
          "-l", /* parameter pertama (argv[1])*/
          "-a", /* parameter kedua (argv[2]) */
          NULL) ; /* terminate arg list */
    printf ("EXEC Failed\n") ;
}
```

Ketikkan juga program berikut , analisa hasilnya apa beda antara `execl` dan `execlp` ?

```

#include <stdio.h>
#include <unistd.h>
int main () {
    execlp ("ls", /* nama program yang dikirim ke argument string argv[0] */
           "ls", /* parameter pertama (argv[1])*/
           "-l",
           "-a", /* parameter kedua (argv[2]) */
           NULL) ; /* terminate arg list */
    printf ("EXEC Failed\n") ;
}

```

Ketikkan juga program beirkut ini, analisa hasilnya .

```

#include <stdio.h>
#include <unistd.h>
int main (argc,argv)
int argc;
char *argv[];
{
    execl ("/bin/echo",
           &argv[0]);
    printf ("EXEC Failed\n") ;
}

```

Ketikkan pula program dibawah ini dan analisa hasilnya :

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main(){
    pid_t hasil;
    printf ("Parent[%d]: PID saya = %d\n",getpid(),getpid());
    printf ("Parent[%d]: Saya akan melakukan forking... \n",getpid());
    hasil = fork();
    if (hasil > 0) { /* ini proses parent */
        printf ("Parent[%d]: PID Child saya = %d\n",getpid(),hasil);
    }
    else {
        printf ("Child[%d]: PID saya = %d\n", getpid(),getpid());
        printf ("Child[%d]: Sekarang...Saya akan mengeksekusi: exec ls!\n\n\n",getpid());
        execlp ("ls", "ls", NULL);
        printf("\n\n");

        printf ("Child[%d]: Sorry....Eksekusi gagal !\n", getpid());
        exit (1) ;
    }
}

```

Berikut ini adalah program dimana Parent membuat proses child kemudian child akan mengeksekusi system call `exec()`

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main(){
    pid_t hasil;
    printf ("Parent[%d]: PID saya = %d\n",getpid(),getpid());
    printf ("Parent[%d]: Saya akan melakukan forking... \n",getpid());
    hasil = fork();
    if (hasil > 0) { /* ini proses parent */
        printf ("Parent[%d]: PID Child saya = %d\n",getpid(),hasil);
    }
    else {
        printf ("Child[%d]: PID saya = %d\n", getpid(),getpid());
        printf ("Child[%d]: Sekarang...Saya akan mengeksekusi: exec ls!\n\n\n",getpid());
        execlp ("ls", "ls", NULL);
        printf("\n\n");

        printf ("Child[%d]: Sorry....Eksekusi gagal !\n", getpid());
        exit (1) ;
    }
}
```

Ketikkan program diatas jalankan program beberapa kali, kenapa hasilnya bisa berbeda-beda ?