

Islamic University – Gaza
Faculty of information technology




Operating Systems Lap

Tamer Alnuaizi
September 2021

What is an OS?

An Operating System (OS) is a software that acts as an interface between the end-user and computer hardware. Every computer must have at least one OS to run other software programs and application like MS Word, Chrome, Games, etc.

Responsibility of OS

- 
- Program execution.
 - Input/Output processing.
 - File systems management.
 - Error detection.
 - Security system.

Program execution

Program load scheduling Program execution from beginning to end
Normal termination and abnormal termination Protecting ripple
effects to another program loading into memory or another resource
such as files, hardware, network, etc.

Input/Output processing

Provide facility for receiving and sending data from and to external system
Communicate with Input/Output devices through drivers
Protecting and ensuring correctness and integrity of data at all time of program execution.

File systems management

Permanent store of data File creating, storing, updating, and deleting are normally accomplished by user software Users don't know where in physical memory their files are stored.

Error detection

OS detects errors during program execution at all times to maintain data integrity. Errors may be caused by the program, computer architecture physical limitations, power supply system, attackers.

Examples of errors:

- 1) Trying to write data in the write-protect section of memory.
- 2) Divide by zero.
- 3) Overflow in the control register, etc.

Security system

Security is important, especially, in multiuser system Access rights or privileges for using resources, utility programs, and communication through computer networks must be controlled Examples of computer and data security: firewall installation, data encryption, etc.

Islamic University – Gaza
Faculty of information technology



Operating Systems Lap

Tamer Alnuaizi
September 2021

Windows Commands

- **cd:** change directory - move to a specific Folder.
- **mkdir:** create a new directory.
- **dir:** display a list of files and folders.
- **cls:** clear the screen.
- **move:** move files from one folder to another.
- **rmdir:** delete directory.

Windows Commands

- **rename:** rename files.
- **del:** delete file.
- **copy:** copy one or more files to another location.
- **xcopy:** copy files and folders
- **echo:** display message on screen.
- **type:** display the contents of a text file.
- **findstr:** search for strings in files.

The most important Linux commands

cd	used to move between folders in the operating system
pwd	stands for Print Working Directory. It prints the path of the working directory, starting from the root.
mkdir	allows users to create or make new directories.
touch	used to create of a file
ls ls -lh ls -l	This command is used to display folders and files within a specific folder
rm -i	for removing files and directories.
clear	used to clear the terminal screen.
rm -f	for removing files
rm -r	for removing directories
cp	used to copy files
mv	moves files or directories from one place to another
find	use to search for a specific file or folder
chmod	used to change the access mode of a file

In Linux there are three types of rights - read, write, and execute. Each of them has its own character designation in chmod, which should be used when working with a command.

r: reading;

w: writing;

x: execute;

It is noticeable that in the properties of the setting item, each group of users is divided. There are also three of them, and they are defined in chmod as follows:

u - the owner of the object;

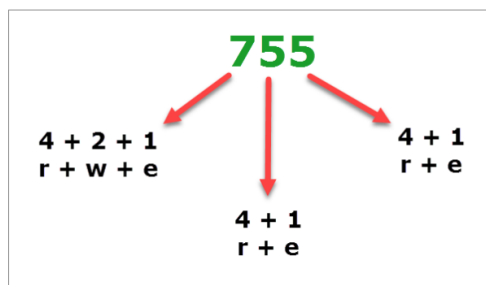
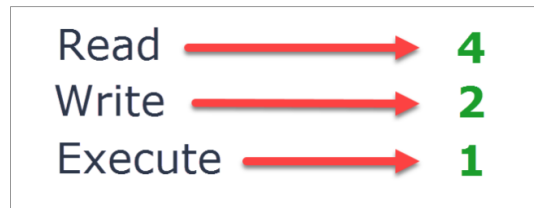
g is a group;

o - other users;

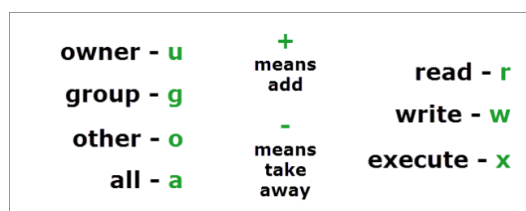
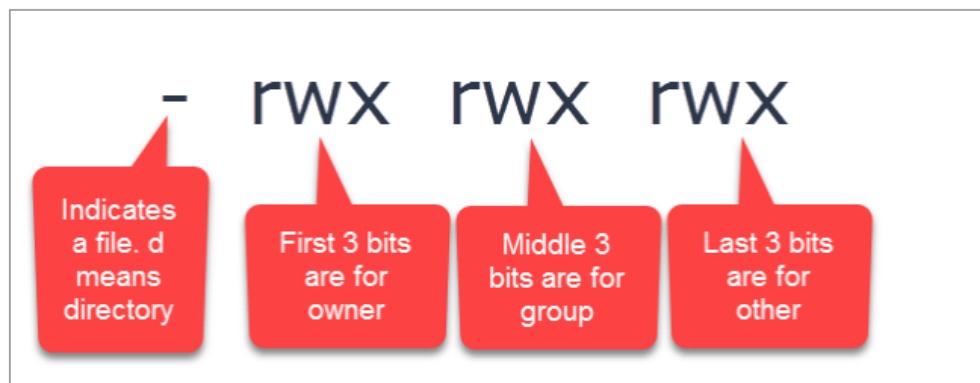
a - All users mentioned above.

In addition, the concerned team accepts rights in the form of numbers. Numbers 0 to 7 denote a specific parameter: 0 - no rights;

- 1 - Execution exclusively;
- 2 - Writing only;
- 3 - Execution and write together;
- 4 - read-only;
- 5 - Read and execute;
- 6 - Reading and writing;
- 7 - All rights together....



When writing ls-lh



© 2015 by Red Hat, Inc. All rights reserved. This document is licensed under a Creative Commons License.

Islamic University – Gaza
Faculty of information technology



Operating Systems Lap

Tamer Alnuaizi
Octoer 2021

System Call

A **system call** is a procedure that provides the interface between a process and the operating system. It is the way by which a computer program requests a service from the kernel of the operating system.

File descriptor is integer that uniquely identifies an open file of the process.

File descriptor table is the collection of integer array indices that are file descriptors in which elements are pointers to file table entries.

Open

Syntax in C language :

```
int open (const char* Path, int flags [, int mode ]);
```

Parameter:

- **Path** : path to file which you want to use.
- **flags** : How you like to use
 - **O_RDONLY**: read only.
 - **O_WRONLY**: write only
 - **O_RDWR**: read and write.
 - **O_CREATE**: create file if it doesn't exist.

Returns:

- Return file descriptor used, -1 upon failure.

Close

Syntax in C language :

```
int close(int fd);
```

Parameter:

➤ **fd** : file descriptor.

Returns:

➤ 0 on success.

➤ -1 on error.

Write

Syntax in C language :

```
size_t write (int fd, void* buf, size_t cnt);
```

Parameter:

- **fd** : file descriptor.
- **buf**: buffer to read data from.
- **cnt**: length of buffer

Returns: How many bytes were actually written

- return number of bytes written on success
- return 0 on reaching end of file
- return -1 on error

Read

Syntax in C language :

```
size_t read (int fd, void* buf, size_t cnt);
```

Parameter:

- **fd** : file descriptor.
- **buf**: buffer to read data from.
- **cnt**: length of buffer

Returns: How many bytes were actually read

- return number of bytes read on success
- return 0 on reaching end of file
- return -1 on error

Fork

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). It takes no parameters and returns an integer value.

Returns:

- **Negative Value:** creation of a child process was unsuccessful.
- **Zero:** Returned to the newly created child process.
- **Positive value:** Returned to parent or caller. The value contains process ID of newly created child process.

Total Number of Processes = 2^n

Note: n = number of fork system calls

Steps

1) Open terminal and run these commands:

- `sudo apt-get update`
- `sudo apt-get upgrade`
- `sudo apt-get install gcc`

2) Now you can create c file by typing this command:

- `gedit example.c`

3) To run the c file, typing this command:

- `gcc example.c`

4) To print the output of the c program, typing this command:

- `./a.out`

Open

// open system call

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

```
#include<fcntl.h>
```

```
#include<errno.h>
```

```
int main(){
```

```
    // if file dose not have in directory
```

```
    // then file foo.txt is created
```

```
    int fd = open("foo.txt", O_RDONLY | O_CREAT);
```

```
    printf("fd = %d", fd);
```

```
    if(fd == -1){
```

```
        // print error
```

```
        printf("Error");
```

```
    }
```

```
    return 0;
```

```
}
```

Close

// c program to illustrate close system call

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

```
#include<fcntl.h>
```

```
#include<string.h>
```

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

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

```
int main(){
```

```
    int fd = open("foo.txt", O_RDONLY | O_CREAT);
```

```
    if (fd < 0 ){
```

```
        perror("Error");
```

```
        exit(1);
```

```
    }
```

```
    printf("opened the fd = %d\n", fd);
```

```
    // Using close system call
```

```
    if (close(fd) < 0 ){
```

```
        perror("c1");
```

```
        exit(1);
```

```
    }
```

```
    printf("closed the fd\n");
```

```
}
```

Write

```
// write system call
```

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

```
#include<fcntl.h>
```

```
#include<string.h>
```

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

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

```
int main(){
```

```
    int fd = open("foo.txt", O_WRONLY | O_CREAT, 0644);
```

```
    if (fd < 0){
```

```
        perror("error");
```

```
        exit(1);
```

```
    }
```

```
    int size = write(fd, "Hello", strlen("Hello"));
```

```
    if (size < 0){
```

```
        perror("error");
```

```
        exit(1);
```

```
    }
```

```
    if (size > 0){
```

```
        printf("write done");
```

```
    }
```

```
    close(fd);
```

```
}
```


Read

// read system call

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

```
#include<fcntl.h>
```

```
#include<string.h>
```

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

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

```
int main(){
```

```
    int fd, sz;
```

```
    char c[100];
```

```
    fd = open("foo.txt", O_RDONLY | O_CREAT);
```

```
    if(fd < 0 ){
```

```
        perror("r1");
```

```
        exit(1);
```

```
    }
```

```
    sz = read(fd, c , 10);
```

```
    printf("Output: %s", c);
```

```
}
```

Fork

// fork system call

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

```
#include <sys/types.h>
```

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

```
int main()
```

```
{
```

```
    int pid = fork();
```

```
    printf("hello\n");
```

```
    printf("pid = %d", pid);
```

```
    return 0;
```

```
}
```