**PRACTICAL-12**

**STUDY OF SYSTEM CALL AND VARIOUS OPERATING SYSTEM MANAGEMENT SERVICES IN UNIX/LINUX O.S AND THEIR IMPLEMENTATION.**

**System services :-**

We consider these under two headings, namely:

. interactive facilities available to the on-line user .

. facilities available to the programmer .

On-line facilities

Most operating systems provide interactive facilities to enable the on-line user to work with files in various ways, usually for the purposes of 'house-keeping'. Some of these facilities are 'built-in' commands of the system, while others are provided by separate utility programs. However, the effect and appearance to the user is very similar .

It would be true to say that many of the powers given to the user in this respect can be somewhat dangreous - it is possible for the user to cause substantial damage to the stored data with carelessly entered commands. This is especially true in more basic operating systems such as MS-DOS which has limited security provisions. Classic examples are the DEL. \*.\* command which can erase all the files in the current directory and, even more devastating, the FORMAT C: command which can erase a whole disk.

**A list of typical facilities are given below:**

. create a file . delete a file . copy a file

. rename a file

. display,) file

. create a directory

. remove an empty directory . list the Cl1ntents of a directory

As mentioned above, these commands are potentially dangerous in the wrong hands, largely because they can be executed with little of the validation which one would expect in an applicatiion program. In general, they are used by technical support staff or 'educated' users.

**Programming services**

As we mentioned earlier, systems vary greatly with respect to the complexity of the file services offered. However, all systems will provide basic operations on which more complex services can be built. A typical set of operations is given below:

. open: make a'file ready for processing.

. close: make a file unavailable for processing

. read: input.data from the file

. write: output data to file

. seek: select a position in the file for subsequent data tran

If you are using a high level language such BASIC, Pascal or C, these operations will be employed in language verbs (e.g. INPUT in BASIC) or built-in procedures (READ in Pascal) or within supplied library routines (fread in C). The actual implementation for the language will involve additional layers of code, supplied by the compiler, to meet the intent of the command. For example, the Pascal READ can input a data item of arbitrary type, hence the implementatiun must take account of the number of bytes in the data being read.

If the. operating system supported more complex file operations, then we would expect to find a much wider range of facilities, such.as reading / writing records, locating a record on the basis of a key value etc. .

At assembler level, one would typically use software interrupts to activate operating system functions. In MS-DOS, interrupt 21 (hex) is a function call request which can perform a wide variety of tasks for the programmer, including opening, reading and writing a file . In addition to file functions, the operating system must provide support for operations on directories . This is not something which one would be implemented in high level languages ,you are unlikely to find a high level language with a ‘change directory’ command . In particular UNIX and LINUX uses C as a system programming language , so that all system calls are implemented as C functions

* Create or remove directory .
* Change directory .
* Read a directory entry .

**Change a directory entry**

**PROGRAM :** Study of system calls and various OS management services in Unix/Linux OS and their implementation .

**SYSTEM CALLS :**

**General :**

To a programmer, invoking a system call is very similar in nature to calling any other Procedure . The essential difference, however, is that in the case of a conventional subroutine, the object code is part of the calling program, while the system call code is within the operating system .

System calls provide an interface between a process and the operating system . These are generally available as assembly-language instructions, and they are usually listed in the various manuals used by assembly-language .

An operating system will only have a limited number of available system calls and in general these will be supplemented by standard subroutine libraries. Many of these standard subroutines, which provide additional and / or higher level facilities for the programmer, will themselves use system calls. These higher level subroutines are generally organized into Application Programming Interfaces or API’s .

These are the entries to the kernel . These are the facilties that the operating system provides . To give a flavour of the tasks performed by system calls, we will describe briefly some of the UNIX and LINUX facilities

**UNIX and LINUX**

System V UNIX provides about 64 system calls moslly concerned with input/output, files or processes, A representative sample is shown below. In UNIX, the C language is used for system progrmnming and hence the system calls are defined as C functions. For our purposes, we show the appropriate parameter list but omit the Ctype identifications .

1. **File Facilities**

|  |  |
| --- | --- |
| open(name,oflag,amode) | Opens file name ;oflag specifies opening for reading ,writing ,append etc.;amode returns a file descriptor fd for use in other calls |
| read(fd,buf,size) | Reads up to size bytes from the file specified by fd into user buffer buf |
| write(fd,buf,count) | Writes count bytes from the user buffer buf to the file specified by fd . |
| fcntl(fd,cmd,arg) | Performs a range of commands, specified by cmd , on the file fd . The parameter arg is a command-dependent argument value . |
| close(fd) | Closes the file fd ;i.e. the file becomes unavailable to the process . |

|  |  |
| --- | --- |
| lseek(fd,offset,origin) | Changes the position of the read / write pointer for the file fd . The position is expressed as offset bytes relative to an origin which can be start or end of file , or the current pointer position . |
| link(f1,f2) | Creates another name , f2 , for the file f1 . |
| unlink(pathname) | It removes a directory from a file where pathname identifies the name of the file to be unlinked . |

1. **Process facilities**

|  |  |
| --- | --- |
| fork() | Cause creation of a new process which is an exact copy of the calling process . |
| exec(file,arglist) | Causes a new process to be created by execution of object file file , overlaying the calling process . Arglist is an array of parameters passed to the new process . |

|  |  |
| --- | --- |
| exit(status) | Exit causes the calling process to terminate ; the vale of status is returned to the process’s parent , i.e. the process which invoked the terminating process . |
| kill(pid,sig) | Sends a software signal identified by sig to the process indentified by pid . signals convey the occurrence of error events to the process’s parent ;the unusal effect that the process is aborted . |
| pause() | Pause suspends the execution of a calling process until it receives a signal . |
| signal(sig,func) | Allows the calling process to control the handling ;of signals;a user-suspended function identified by func can be activated on receipt of the signal sig . |

1. **Miscellaneous**

|  |  |
| --- | --- |
| mount(filesys,dir,rflag) | Cause a new file system (e.g. Floppy drive volume) specified by filesys to be connected into the current file tree at the directory dir . If rflag=1 , the new file system is moouned read-only . |
| time(secs) | Returns ,and sets the parameter secs to, the number of seconds which have elapsed since the beginning of January 1st . 1970 GMT . |
| stime(secs) | Sets the system time and date , expressed by the number of seconds. |