ASSINGMENT : - 1

NAME:-SOURABH PATEL

ADDMISSION NO:-U19CS082

Q:-1 To study the basics of system call and system library.

System Call:-

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Types of System Calls:-

1.Process Control

- Create Process (for example,fork on unix-like systems,or NtCreateProcess in the Windows NT Native API)
- terminate Process
- load,execute
- get/set process attributes
- wait for time,wait event,signal event
- allocate and free memory

2.File Management

- create file,delete file
- open,close
- read,write,reposition
- get/set file attributes

3.Device Management

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

4.Information Maintenance

- get/set total system information (including time,date,computer name,enterprise etc.)
- get/set process,file,or device metadata(including author,opener,creation time and data, etc.)

5.Communication

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

6.Protection

- get/set file permissions

Type of system call	Examples
Process Control	fork(),exit(),wait()
File Management	Open(),read(),write(),close()
Device Management	Ioctl(),read(),write()
Information Maintenance	Getpid(),alarm(),sleep()
Communication	Pipe(),mmap()

System Library:-

An organized collection of computer programs that is maintained online with a comp uter system by being held on a secondary storage device and is managed by the op erating system.

Q:-2 Study the following system calls :

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). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.It takes no parameters and returns an integer value.

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.

exec:-

In computing, exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable. This act is also referred to as an overlay. It is especially important in Unixlike systems, although it exists elsewhere. As no new process is created, the process identifier (PID) does not change, but the machine code, data, heap, and stack of the process are replaced by those of the new program.

The *exec* call is available for many programming languages including compilable languages and some scripting languages. In OS command interpreters, the exec built-in command replaces the shell process with the specified program.

getpid:-

returns the process ID of the calling process. This is often used by routines that generate unique temporary filenames.

Syntax: pid_t getpid(void);

Return type: getpid() returns the process ID of the current process. It never throws any error therefore is always successful.

exit:-

The exit function, declared in <stdlib.h>, terminates a C++ program. The value supplied as an argument to exit is returned to the operating system as the program's

return code or exit code. By convention, a return code of zero means that the program completed successfully. You can use the constants EXIT_FAILURE and EXIT_SUCCESS, also defined in <stdlib.h>, to indicate success or failure of your program.

Issuing a return statement from the main function is equivalent to calling the exit function with the return value as its argument.

wait:-

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent *continues* its execution after wait system call instruction.
Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.

stat:-

stat () is a Unix system call that returns file attributes about an inode. The semantics of stat () vary between operating systems. As an example, Unix command ls uses this system call to retrieve information on files that includes: stat appeared in Version 1 Unix.

opendir:-

The opendir () function opens a directory handle. Required. Specifies the directory path to be opened Optional. Specifies the context of the directory handle. Context is a set of options that can modify the behavior of a stream Returns the directory handle resource on success. FALSE on failure.

readdir:-

The readdir () system call function is used to read into a directory**.** The function returns a pointer to a dirent structure. This structure contains five fields but only two are POSIX standard, this is the d_name and the d_ino member. That's the first we will use in our readdir () example.

chdir:-

The chdir command is a system function (system call) which is used to change the current working directory. On some systems, this command is used as an alias for the shell command cd. chdir changes the current working directory of the calling process to the directory specified in path.

int chdir(const char *path);

Parameter: Here, the *path* is the Directory path which the user want to make the current working directory.
Return Value: This command returns zero (0) on success. -1 is returned on an error and errno is set appropriately.

chmod:-

In Unix and Unix-like operating systems, chmod is the command and system call used to change the access permissions of file system objects sometimes known as modes. It is also used to change special mode flags such as setuid and setgid flags and a 'sticky' bit. The request is filtered by the umask. The name is an abbreviation of change mode. They are shown when listing files in long format.

kill:-

To send a signal to another process, we need to use the Unix system kill(). The following is the prototype of kill():

int kill(pid_t pid, int sig)

- System call kill() takes two arguments. The first, pid, is the process ID you want to send a signal to, and the second, sig, is the signal you want to send. Therefore, you have to find some way to know the process ID of the other party.
- If the call to kill() is successful, it returns 0; otherwise, the returned value is negative.
- Because of this capability, kill() can also be considered as a communication mechanism among processes with signals SIGUSR1 and SIGUSR2.
- The pid argument can also be zero or negative to indicate that the signal should be sent to a group of processes. But, for simplicity, we will not discuss this case.

read:-

A system call is a method for software to communicate with the operating system.

When software performs a system call, it sends the request to the kernel of the operating system. To read by a file descriptor, you can use the read () system function. Each process has its personal file descriptors table in the operating system. The sole difference between read () and write () is that read () reads data from the file referred to by the file descriptor. The reading time for the file is updated after a successful read ().

write:-

The write is one of the most basic routines provided by a Unix-like operating system kernel. It writes data from a buffer declared by the user to a given device, such as a file. This is the primary way to output data from a program by directly using a system call. The destination is identified by a numeric code. The data to be written, for instance a piece of text, is defined by a pointer and a size, given in number of bytes.

`write` thus takes three arguments:

1. The file code (file descriptor or fd).
2. The pointer to a buffer where the data is stored (buf).
3. The number of bytes to write from the buffer (nbytes).

open:-

The open() system call is used to provide access to a file in a file system. This system call allocates resources to the file and provides a handle that the process uses to refer to the file. A file can be opened by multiple processes at the same time or be restricted to one process. It all depends on the file organisation and file system.

close:-

The close() system call is used to terminate access to a file system. Using this system call means that the file is no longer required by the program and so the buffers are flushed, the file metadata is updated and the file resources are deallocated.

lseek:- lseek is a system call that is used to change the location of the read/write pointer of a file descriptor**.** The location can be set either in absolute or relative terms. Function Definition. off_t lseek(int fildes, off_t offset, int whence); Field Description int fildes : The file descriptor of the pointer that is going to be moved

time:-

The time() function is defined in time.h (ctime in C++) header file. This function returns the time since 00:00:00 UTC, January 1, 1970 (Unix timestamp) in seconds.

If second is not a null pointer, the returned value is also stored in the object pointed to by second.

Syntax: time_t time(time_t
*second)

mount:-

Mount system call makes a directory accessible by attaching a root directory of one file system to another directory**.** In UNIX directories are represented by a tree structure, and hence mounting would mean attaching them to the branches. This means the file system found on one device can be attached to the tree.

chown:- chown command is used to change the file Owner or group. Whenever you want to change ownership you can use chown command. Syntax:

chown [OPTION]… [OWNER][:[GROUP]] FILE… chown

[OPTION]… –reference=RFILE FILE…