**4ITRC2 Operating System Lab**

**Lab Assignment 4**

**Aim:** To study and learn about various system calls

**To perform:** Comprehensive study of different categories of Linux system calls, categorized as

**1. Process Management System calls**

These system calls are responsible for creating, managing, and terminating processes.

**fork()**

It creates a new child process by duplicating the current (parent) process. Both processes continue execution independently.

* Purpose: Creates a new process by duplicating the calling process.
* Details: Returns the PID of the child to the parent and 0 to the child process. On failure, returns -1.
* Use Case: Creating a child process to perform a different task concurrently.

pid\_t pid = fork();

if (pid == 0) {

// Child process

} else if (pid > 0) {

// Parent process

}

**exec()**

exec() replaces the current process image with a new program.

* Purpose: Replaces the current process image with a new process image.
* Details: Comes in several forms like execl(), execp(), execv() etc.
* Use Case: Used after fork() in the child process to run a new program.

execl("/bin/ls", "ls", "-l", NULL);

**wait()**

It makes a parent process wait for its child process to finish.

* Purpose: Waits for the child process to terminate.
* Details: It blocks the calling process until one of its child processes exits or a signal is received.
* Use Case: Used in parent process to synchronize with the child.

int status;

wait(&status);

**exit()**

It is used to terminate a process and return a status code.

* Purpose: Terminates the calling process.
* Details: Takes an integer status value to be returned to the parent via wait().

exit(0);

**2. File Management System calls**

These are used to handle files like opening, reading, writing, and closing.

**open()**

* Purpose: Opens a file and returns a file descriptor.
* Syntax: int fd = open("file.txt", O\_RDONLY);

**read()**

* Purpose: Reads data from a file descriptor into a buffer.
* Syntax: read(fd, buffer, size);

**write()**

* Purpose: Writes data to a file descriptor.
* Syntax: write(fd, buffer, size);

**close()**

* Purpose: Closes the file descriptor.
* Syntax: close(fd);

**3. Device Management System calls**

These allow direct interaction with hardware devices via device files.

**read() and write()**

* Purpose: Similar to file operations but applied on device files like /dev/tty, /dev/sda, etc.

**ioctl()**

* Purpose: Performs device-specific input/output operations.
* Syntax: ioctl(fd, request, argument);
* Use Case: Used to configure device settings like serial port baud rate.

**select()**

* Purpose: Monitors multiple file descriptors to see if I/O is possible.
* Use Case: Useful in I/O multiplexing.

fd\_set readfds;

FD\_ZERO(&readfds);

FD\_SET(fd, &readfds);

select(fd+1, &readfds, NULL, NULL, NULL);

**4. Network Management System calls**

These enable socket-based communication for networking.

**socket()**

* Purpose: Creates a socket endpoint.
* Syntax: int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

connect()

* Purpose: Initiates a connection on a socket to a remote server.

**send() and recv()**

* Purpose: Send and receive data over the network.
* Syntax:

send(sockfd, msg, strlen(msg), 0);

recv(sockfd, buffer, sizeof(buffer), 0);

5. System Information Management System calls

These system calls fetch system-related info like process/user IDs, system configuration, etc.

**getpid()**

* Purpose: Returns the process ID of the calling process.

**getuid()**

* Purpose: Returns the user ID of the calling process.

**gethostname()**

* Purpose: Gets the standard host name of the machine.

char hostname[1024];

gethostname(hostname, sizeof(hostname));

**sysinfo()**

* Purpose: Provides system statistics (RAM, uptime, load, etc.)

struct sysinfo info;

sysinfo(&info);