**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, categorizedas

**1. Process Management System calls**

fork(), exec(), wait(), exit().

**2. File Management System calls**

open(), read(), write(), close().

**3. Device Management System calls**

read(), write(), ioctl(), select().

**4. Network Management System calls**

socket(), connect(), send(), recv().

**5. System Information Management System calls**

getpid(), getuid(), gethostname(), sysinfo().

**To Submit:** Write up for the exhaustive study of the above mentioned system call categories with

their examples.

**1. Process Management System Calls**

Process management involves creating, executing, suspending, resuming, and terminating processes. These system calls provide essential control over the lifecycle of processes.

**fork()**

* fork() is used to create a new process by duplicating the calling process. The new process is referred to as the child process.
* The child process has a unique process ID but initially shares the code segment with the parent.
* Return Values:
  + 0: Returned to the child process
  + 0: Process ID of the child, returned to the parent
  + -1: Error occurred

**Use Cases**:

* Implementing multiprocessing
* Spawning background tasks

**EXAMPLE-**

pid\_t pid = fork();

if (pid == 0)

printf("Child process with PID: %d\n", getpid());

else if (pid > 0)

printf("Parent process with PID: %d\n", getpid());

else

perror("Fork failed");

**exec() Family**

* Replaces the current process image with a new process image.
* Variants include: execl, execv, execlp, execvp, execle, execvpe.
* After a successful exec() call, the new program starts and the original process code is no longer executed.

**Use Cases**:

* Used after fork() to run different programs

**EXAMPLE-**

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

**wait() and waitpid()**

* wait() makes the parent wait until one of its child processes exits.
* waitpid() allows for more fine-grained control (waiting for a specific process).
* Returns the process ID of the terminated child.

**EXAMPLE-**

int status;

pid\_t pid = wait(&status);

**exit()**

* Terminates the calling process.
* The exit status is returned to the parent process (can be accessed using wait() or waitpid()).

**EXAMPLE-**

exit(0);

(0 indicates successful termination)

**2. File Management System Calls**

File system calls allow user programs to perform operations on files such as creation, reading, writing, and deletion.

**open()**

* Opens a file and returns a file descriptor (non-negative integer).
* Flags: O\_RDONLY, O\_WRONLY, O\_RDWR, O\_CREAT, O\_APPEND, O\_TRUNC
* Mode specifies file permissions (e.g., 0644 for read/write).

**EXAMPLE-**

int fd = open("data.txt", O\_CREAT | O\_WRONLY, 0644);

**read()**

* Reads data from an open file into a buffer.

**EXAMPLE-**

char buffer[100];

int bytes = read(fd, buffer, sizeof(buffer));

**write()**

* Writes the contents of a buffer into an open file.

**EXAMPLE-**

write(fd, "Hello, World!\n", 14);

**close()**

* Closes an open file descriptor to free system resources.

**EXAMPLE-**

close(fd);

**3. Device Management System Calls**

Device files represent hardware devices in Unix-like systems. These system calls allow low-level interaction with devices.

**read() and write()**

* These calls also work with device files (e.g., /dev/tty, /dev/sda).

**ioctl()**

* Used to control device parameters.
* Defined using macros in <sys/ioctl.h>.

**EXAMPLE-**

ioctl(fd, COMMAND, &argument);

**Use Cases**:

* Configuring serial ports
* Querying disk device properties

**select()**

* Allows a process to monitor multiple file descriptors to see if they are ready for I/O.
* Avoids blocking on I/O calls.

**EXAMPLE-**

fd\_set rfds;

FD\_ZERO(&rfds);

FD\_SET(0, &rfds); // monitor stdin

select(1, &rfds, NULL, NULL, NULL);

**4. Network Management System Calls**

These system calls enable inter-process communication (IPC) over networks using sockets. Widely used in client-server applications.

**socket()**

* Creates a socket.
* Types: SOCK\_STREAM (TCP), SOCK\_DGRAM (UDP)

**EXAMPLE-**

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

**bind()**

* Associates an address (IP + port) with a socket.

**EXAMPLE-**

bind(sockfd, (struct sockaddr\*)&addr, sizeof(addr));

**listen() and accept()**

* listen() marks a socket as passive (used to accept connections).
* accept() extracts the first connection request.

**EXAMPLE-**

listen(sockfd, 5);

int new\_sock = accept(sockfd, (struct sockaddr\*)&client, &len);

**connect()**

* Used by clients to connect to a server socket.

**EXAMPLE-**

connect(sockfd, (struct sockaddr\*)&server, sizeof(server));

**send() and recv()**

* send() transmits data, recv() receives it.

**EXAMPLE-**

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

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

**shutdown() and close()**

* shutdown() disables sends/receives.
* close() releases the socket descriptor.

**5. System Information Management System Calls**

These calls provide critical information about system and process metadata.

**getpid()**

* Returns the current process ID.

**EXAMPLE-**

pid\_t pid = getpid();

**getppid()**

* Returns parent process ID.

**EXAMPLE-**

pid\_t ppid = getppid();

**getuid() and geteuid()**

* Return real and effective user IDs.
* Used in permission and access control.

**getgid() and getegid()**

* Return real and effective group IDs.

**gethostname()**

* Retrieves system hostname.

**EXAMPLE-**

char hostname[1024];

gethostname(hostname, sizeof(hostname));

**sysinfo()**

* Provides memory, uptime, and load statistics.

**EXAMPLE-**

#include <sys/sysinfo.h>

struct sysinfo info;

sysinfo(&info);

printf("Uptime: %ld seconds\n", info.uptime);