**LAB ASSIGNMENT-4**

**Process Management System Calls**

**1. fork()**  
Creates a new child process by duplicating the parent process. Returns 0 to child and PID to parent.

c

pid\_t pid = fork();

if (pid == 0) {

*/\* Child process \*/*

} else {

*/\* Parent process \*/*

}

Key features:

* Creates separate address spaces
* Essential for process hierarchy

**2. exec()**  
Replaces current process image with new program. Multiple variants:

c

execl("/bin/ls", "ls", "-l", NULL); *// List format arguments*

execv("/bin/ls", args); *// Array format arguments*

* Never returns on success
* Preserves PID but replaces memory

**3. wait()**  
Makes parent process wait for child termination:

c

int status;

pid\_t child\_pid = wait(&status);

* Retrieves child exit status via WEXITSTATUS
* Prevents zombie processes

**4. exit()**  
Terminates process with status code:

c

exit(EXIT\_SUCCESS); *// 0 indicates success*

* Closes all file descriptors
* Sends SIGCHLD to parent

**File Management System Calls**

**1. open()**  
Opens/Creates files with specified flags:

c

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

* Returns lowest available file descriptor
* Flags control access mode (O\_RDONLY/O\_WRONLY)

**2. read()/write()**  
Handles data transfer:

c

char buffer[1024];

ssize\_t bytes\_read = read(fd, buffer, sizeof(buffer));

ssize\_t bytes\_written = write(fd, data, data\_len);

* Returns bytes transferred, 0 on EOF
* Uses file offset pointer

**3. close()**  
Releases file descriptor:

c

close(fd);

* Essential for resource management
* Automatic on process termination

**Device Management System Calls**

**1. ioctl()**  
Device-specific operations:

c

ioctl(fd, TIOCGWINSZ, &winsize); *// Get terminal size*

* Generic interface for device configuration
* Requires device-specific knowledge

**2. select()**  
Multiplexed I/O monitoring:

c

fd\_set read\_fds;

select(max\_fd+1, &read\_fds, NULL, NULL, &timeout);

* Handles multiple descriptors simultaneously
* Limited to FD\_SETSIZE (1024) descriptors

**Network Management System Calls**

**1. socket()**  
Creates communication endpoint:

c

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

* AF\_INET for IPv4, SOCK\_STREAM for TCP
* Returns file descriptor for network operations

**2. connect()**  
Initiates connection:

c

struct sockaddr\_in addr;

connect(sock, (struct sockaddr\*)&addr, sizeof(addr));

* Blocks until connection established
* Essential for TCP clients

**3. send()/recv()**  
Data transmission:

c

send(sock, buffer, len, 0);

recv(sock, buffer, len, 0);

* Flags control transmission behavior
* Connection-oriented (TCP) vs connectionless (UDP)

**System Information Management**

**1. getpid()/getppid()**

c

pid\_t current = getpid(); *// Current PID*

pid\_t parent = getppid(); *// Parent PID*

* Unique process identification

**2. getuid()**

c

uid\_t user\_id = getuid(); *// Real user ID*

* User privilege management

**3. gethostname()**

c

char name[256];

gethostname(name, sizeof(name));

* Retrieves system hostname

**4. sysinfo()**

c

struct sysinfo info;

sysinfo(&info); *// Contains uptime, load averages, etc.*

* Returns system-wide statistics