Lab Assignment-4

1. Process Management System Calls

Process management system calls help create, execute, and manage processes in Linux.

a) fork()

The fork() system call creates a new child process, which is an exact copy of the parent process.

```
#include <stdio.h>
#include <unistd.h>
int main() {
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child process\n");
    } else {
        printf("Parent process\n");
    }
    return 0;
}
```


b) exec()

The exec() system call replaces the current process image with a new process image.

```
#include <stdio.h>
#include <unistd.h>
int main() {
    char *args[] = {"/bin/ls", NULL};
    execvp(args[0], args);
    return 0;
}
```

c) wait()

The wait() system call makes a parent process wait until a child process terminates.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
```

```
#include <unistd.h>
int main() {
    pid_t pid = fork();
    if (pid > 0) {
        wait(NULL);
        printf("Child process finished\n");
    }
    return 0;
}
```

d) exit()

The exit() system call terminates a process and releases resources.

```
#include <stdlib.h>
int main() {
    exit(0);
}
```


2. File Management System Calls

These system calls handle file operations like opening, reading, writing, and closing files.

a) open()

```
#include <fcntl.h>
#include <stdio.h>
int main() {
    int fd = open("file.txt", O_CREAT | O_WRONLY, 0644);
    return 0;
}
```

b) read()

```
#include <unistd.h>
int main() {
    char buffer[100];
    read(0, buffer, 100);
```

```
    return 0;
}
```

c) write()

```
#include <unistd.h>
int main() {
    write(1, "Hello, world!", 13);
    return 0;
}
```

d) close()

```
#include <unistd.h>
int main() {
    int fd = open("file.txt", O_RDONLY);
    close(fd);
    return 0;
}
```


3. Device Management System Calls

These system calls interact with hardware devices.

a) read() & write() (Device-specific)

Used to read from and write to devices.

b) ioctl()

Used to control devices.

```
#include <sys/ioctl.h>
#include <fcntl.h>
int main() {
    int fd = open("/dev/tty", O_RDONLY);
    ioctl(fd, 0, NULL);
    return 0;
}
```

c) select()

Monitors multiple file descriptors.

```
#include <sys/select.h>
int main() {
    fd_set set;
    FD_ZERO(&set);
    FD_SET(0, &set);
    select(1, &set, NULL, NULL, NULL);
    return 0;
}
```

---

4. Network Management System Calls

These system calls handle network connections and communication.

a) socket()

Creates a socket.

```
#include <sys/socket.h>
int main() {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    return 0;
}
```


b) connect()

Connects to a remote server.

```
#include <sys/socket.h>
#include <arpa/inet.h>
int main() {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server;
    server.sin_family = AF_INET;
    server.sin_port = htons(8080);
    connect(sock, (struct sockaddr *)&server, sizeof(server));
    return 0;
```

```
}
```

c) send() & recv()

Used for sending and receiving data over a network.

```
#include <sys/socket.h>
int main() {
    char buffer[1024];
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    send(sock, "Hello", 5, 0);
    recv(sock, buffer, 1024, 0);
    return 0;
}
```

---

5. System Information Management System Calls

These system calls retrieve system-related information.

a) getpid()

Returns the process ID.

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("PID: %d\n", getpid());
    return 0;
}
```

b) getuid()

Returns user ID.

```
#include <unistd.h>
int main() {
    printf("UID: %d\n", getuid());
    return 0;
}
```

c) gethostname()

Gets the hostname.

```
#include <unistd.h>
int main() {
    char hostname[100];
    gethostname(hostname, 100);
    printf("Hostname: %s\n", hostname);
    return 0;
}
```

d) sysinfo()

Retrieves system information.

```
#include <sys/sysinfo.h>
#include <stdio.h>
int main() {
    struct sysinfo info;
    sysinfo(&info);
    printf("Uptime: %ld seconds\n", info.uptime);
    return 0;
}
```