System Call

What is a System Call?

A **system call** is a programmatic way for a user-space application to request services from the kernel (the core of the OS). These services include file manipulation, process control, and networking.

In Linux, system calls are accessed using special instructions (like `syscall` on x86_64), and user-space libraries (like **glibc**) wrap these system calls in C functions.

What is the `listen` System Call?

The `listen()` system call is used in **network programming**. After a socket is created and bound to an address, `listen()` is called to **mark the socket as passive**, i.e., it will be used to accept incoming connection requests.

Syntax (C):

```
int listen(int sockfd, int backlog);
```

Parameters:

- `sockfd`: The file descriptor of the socket.
- `backlog`: The number of incoming connections that can be queued before the kernel starts rejecting new connections.

Return Value:

- `0` on success.
- `-1` on error (with `errno` set to indicate the error).

Kernel Internals: What Happens in `listen()`?

When you call `listen()` in Manjaro (or any Linux distro), here's what happens under the hood:

1. User space to Kernel space transition

The `listen()` function in your C program calls into glibc, which internally issues the actual system call using the syscall interface.

```
syscall(SYS_listen, sockfd, backlog);
```

2. Kernel System Call Entry

In the kernel (say Linux 6.x used by Manjaro), the syscall is dispatched to the `sys_listen` implementation in the kernel source.

This typically resides in:

```
net/socket.c
```

3. sys_listen() Function

This function looks like:

```
SYSCALL_DEFINE2(listen, int, fd, int, backlog)
{
    return __sys_listen(fd, backlog);
}
```

It forwards the call to `__sys_listen`.

4. __sys_listen()

This function retrieves the socket from the file descriptor, ensures it's valid, and then calls the `listen()` function defined in the socket's operations table.

```
int __sys_listen(int fd, int backlog)
{
    struct socket *sock;
    int err;

    sock = sockfd_lookup(fd, &err);
    if (!sock)
        return err;

    err = security_socket_listen(sock, backlog);
    if (err)
        goto out;

    err = sock->ops->listen(sock, backlog);
out:
    fput(sock->file);
    return err;
}
```

5. Protocol-specific listen()

`sock->ops->listen` points to the specific protocol implementation:

- For TCP, this is:
- `inet_listen() in net/ipv4/af_inet.c`

This sets the socket's internal state to **LISTEN**, allocates a **request queue**, and prepares it to accept connections.

Implementation code for Listen call in manjaro linux

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define BACKLOG 5

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
        perror("setsockopt");
```

```
        exit(EXIT_FAILURE);
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    if (listen(server_fd, BACKLOG) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    printf("Listening on port %d...\n", PORT);

    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    printf("Connection accepted.\n");

    close(new_socket);
    close(server_fd);

    return 0;
}
```