Implement system call

What is a System Call?

A **system call** is a mechanism that allows a user-space program to request services from the **kernel** of the operating system. This controlled interface enables programs to interact with low-level system components like hardware, memory, and files.

Example:-

- **File operations** – e.g., `open`, `read`, `write`, `close`
- **Process management** – e.g., `fork`, `exec`, `exit`
- **Memory management** – e.g., `mmap`, `brk`
- **Network communication** – e.g., `socket`, `send`, `recv`

Steps to Implement System Calls in Fedora Server 41

1. Update the System

Ensure Fedora Server 41 is up to date:

Write this in terminal

```
sudo dnf update
```

2. Install Development Tools

Install essential development tools like compilers and debuggers:

```
sudo dnf groupinstall "Development Tools"
```

3. Install Kernel Development Packages

Install headers and source files for the currently running kernel:`sudo dnf install kernel-devel kernel-headers`

4. Download the Kernel Source Code

Download the matching kernel source from Fedora's official repositories or https://www.kernel.org. For Fedora-specific kernels:

```
sudo dnf install kernel-source
```

5. Set Up the Development Environment

Navigate to the kernel source directory

```
cd /usr/src/kernels/$(uname -r)/
```

Alternatively, if you downloaded a tarball:

```
tar -xf linux-<version>.tar.xz
cd linux-<version>
```

6. Modify Kernel Files

Edit the required kernel source files to add or change a system call. This typically involves:

- Declaring the new system call in `include/linux/syscalls.h`
- Defining its implementation in a `.c` file (often under `kernel/` or `fs/`)
- Registering it in the syscall table (architecture-specific, e.g., `arch/x86/entry/syscalls/syscall_64.tbl`)

7. Rebuild the Kernel

Build the kernel with the new system call

```
make menuconfig     # Optional: configure kernel options
make -j$(nproc)     # Build using all cores
make modules
sudo make modules_install
sudo make install
```

8. Reboot into the New Kernel

After installation, reboot to use the new kernel

```
sudo reboot
```

Test with c language code

fork()system call

```
#include <stdio.h>
 #include <stdlib.h>
#include <unistd.h>
 int main() { pid_t pid = fork();
(pid < 0)
{ perror("Fork failed");
return 1; }
else if (pid == 0)
 { printf("Hello from the child process!\n"); }
 else {
printf("Hello from the parent process! Child PID: %d\n", pid); }
return 0; }
```

For write and read

```
#include <stdio.h>
 #include <stdlib.h>
 #include <unistd.h>
#include <string.h>
int main() { pid_t pid; int fd[2];
buffer[100];
(pipe(fd) == -1)
{ perror("Pipe failed");
 return 1; } pid = fork();
 (pid < 0) { perror("Fork failed");
 return 1; }
else if (pid == 0) { close(fd[1]);
read(fd[0], buffer, sizeof(buffer));
printf("Child received: %s\n", buffer);
close(fd[0]); } else { close(fd[0]);
*message = "Hello from the parent!";
 write(fd[1], message, strlen(message) + 1);
close(fd[1]); }
return 0; }
```