# System call

**System calls are requests from user programs to the OS kernel.**

**Examples include read(), write(), fork(), open().**

**Problem: For system calls to work, the kernel must be running.**

**If OmniOS cannot boot, the kernel never loads, so system calls cannot be executed at all.**

## Why System Calls Are Failing

**Bootloader Failure**

**The bootloader (GRUB / bootadm) is responsible for loading the kernel.**

**If the bootloader is missing, corrupted, or misconfigured, the kernel never starts → no system calls can be processed.**

**BIOS / UEFI Mismatch**

**If the system was installed in UEFI mode but BIOS is in Legacy (or vice versa), the CPU cannot find and load the kernel.**

**Without a running kernel, the system cannot accept system call requests.**

**ZFS / Root Filesystem Issues**

**OmniOS uses ZFS for its root filesystem.**

**If the root pool (rpool) is corrupted or the kernel cannot mount it, the OS halts during boot.**

**Result: system calls fail because the OS environment is not active.**

**CPU or Hardware Issues**

**An incompatible CPU (older instruction set) or unsupported disk/RAID controller may prevent the kernel from starting.**

**Without kernel execution, system calls cannot be handled.**

## How This Relates to System Calls

**Normally, a system call works like this**

**User Program -> System Call -> Kernel -> Hardware**

**If the kernel does not start, the flow is broken:Copy code**

**User Program -> (Kernel not loaded) -> No system call execution In short, the system call layer depends entirely on a successfully booted OS.4. Solutions to Enable System Calls (Fix Boot First)**

**To use system calls, you first need to get OmniOS booting:**

**Check BIOS/UEFI Mode → Ensure it matches the installation mode (UEFI or Legacy).**

**Repair Bootloader → Boot from USB/ISO and run:**

**Copy code**

**bootadm install-bootloader -v /dev/dsk/cXtXdX**

**Check ZFS Root Pool → Import it if necessary:**

**Copy code**

**zpool import -f rpool**

**zfs list**

**Verify CPU Compatibility → Use prtdiag from Live USB; if missing instructions, consider older OmniOS build.**

**Reinstall OmniOS → If bootloader or ZFS is corrupted, reinstall ensuring correct disk selection and BIOS mode.**

```
# include <stdio.h>
# include <fcntl.h>
# include <unistd.h>
 int main() {
   int fd;
   char buffer[100];
   ssize_t bytes;
   // Open a file for reading
   fd = open("/etc/passwd", O_RDONLY);
```

```
    if (fd == -1) {
        perror("open");
        return 1;
    }

    // Read up to 100 bytes from the file
    bytes = read(fd, buffer, sizeof(buffer)-1);
    if (bytes == -1) {
        perror("read");
        close(fd);
        return 1;
}
    buffer[bytes] = '\0'; // Null-terminate
    printf("File content:\n%s\n", buffer);
    // Close the file
    close(fd);
return 0;}
```

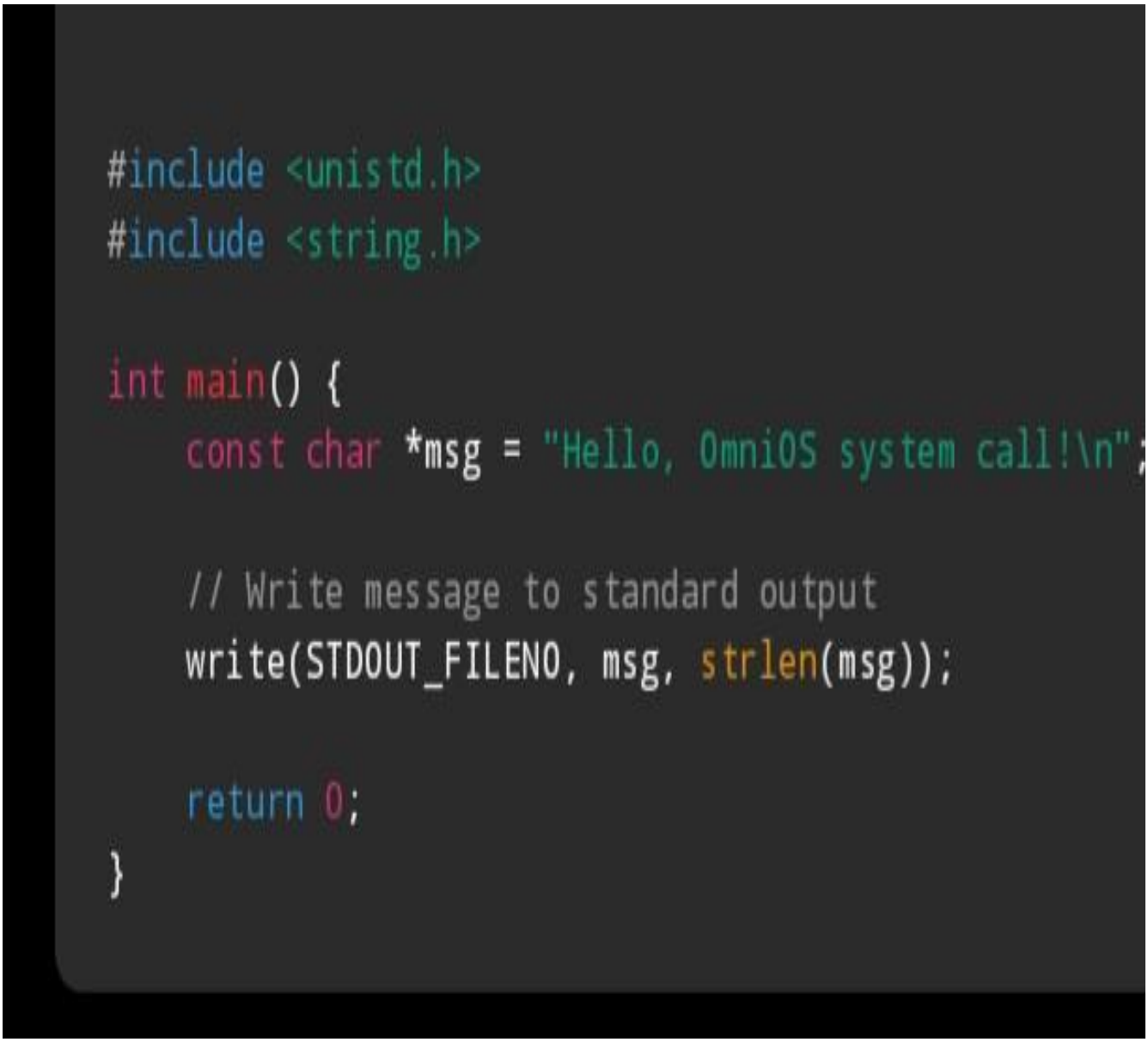
## 1. File Operations: open(), read(), write(), close()

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    int fd;
    char buffer[100];
    ssize_t bytes;

    // Open a file for reading
    fd = open("/etc/passwd", O_RDONLY);
    if (fd == -1) {
        perror("open");
        return 1;
    }

    // Read up to 100 bytes from the file
    bytes = read(fd, buffer, sizeof(buffer)-1)
    if (bytes == -1) {
        perror("read");
        close(fd);
        return 1;
    }

    buffer[bytes] = '\0'; // Null-terminate
    printf("File content:\n%s\n", buffer);

    // Close the file
    close(fd);

    return 0;
}
```

## 2. Process Management: fork(), getpid()

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid;

    // Create a new process
    pid = fork();

    if (pid < 0) {
        perror("fork");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("Child process, PID: %d\n", getpid()
    } else {
        // Parent process
        printf("Parent process, PID: %d, Child PID:
%d\n", getpid(), pid);
    }

    return 0;
}
```

## 3. Simple Output: write() to stdout



```c
#include <unistd.h>
#include <string.h>

int main() {
    const char *msg = "Hello, OmniOS system call!\n";

    // Write message to standard output
    write(STDOUT_FILENO, msg, strlen(msg));

    return 0;
}
```

Note for Your Current Situation These system calls will not work until your Omni OS system boots successfully.

Preparing the code now is useful for submission or testing later.Once booted, compile with:Copy code

cc filename.c -o output ./output