Bahir Dar University Bahir Dar Institute of Technology

Faculty of Computing

Department of Software Engineering

Course Title: Operating System and System Programming

Documentation for Mac-OS Sonoma

Name: Robel Yihelu

ID: BDU1602340

Section: B

Submitted to: Lec. Wendmu B.

April, 2017

Implementing system call

What Is a System Call?

A system call serves as a controlled interface within an operating system, enabling applications in user mode to delegate sensitive tasks—such as hardware interaction, file management, or network communication—to the kernel, which operates with elevated privileges to execute these operations securely.

Examples of system calls:

- read(), write() → file I/O
- fork(), exec() → process management
- sendto(), recvfrom() → socket communication

we focus on sendto() system call and we will see later.

System calls act as the interface between user space and kernel space.

Implementing a system call in macOS Sonoma (a UNIX-certified OS based on the XNU kernel) is a complex process that involves modifying the kernel source code, recompiling it, and bypassing Apple's security protections. Below is a detailed guide, including technical steps, code examples, and caveats.

1. Understanding macOS System Calls

- **XNU Kernel**: macOS uses a hybrid kernel (Mach + BSD). System calls are handled via the BSD layer.

- **System Call Table**: Defined in syscalls.master (legacy) or via syscall.h macros in modern XNU.

- **Security Restrictions:**

- System Integrity Protection (SIP): Prevents kernel modification.

- **Apple Notarization:** Kernel extensions (kexts) require Apple's approval (deprecated in favor of system extensions).

What is sendto()?

- ✓ **sendto()** system call—a key function in **network programming** used for sending data over sockets. Since you're interested in both **system call fundamentals** and how sendto() works under the hood (especially in macOS), we'll cover:
- ✓ **sendto()** is a **network system call** used to send data over a socket to a specific destination (IP + port), typically used in **UDP** communication.

Function Prototype:

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
        const struct sockaddr *dest_addr, socklen_t addrlen);
```

Parameters:

Parameter	Description
sockfd	The socket file descriptor (created using socket())
buf	Pointer to the data to be sent
len	Length of the data
flags	Flags to modify behavior (usually 0)
dest_addr	Destination address (IP and port)
addrlen	Size of the address struct

Return Value:

- Returns number of bytes sent, or -1 on error (use errno to check error).

3. Where is sendto() Implemented? (System Layer View)

User Space:

In user programs, sendto() is part of the **C standard library (glibc / libSystem on macOS)**.

Transition to Kernel:

When you call sendto():

1. Your code calls the **libc wrapper function**.
2. This triggers a **trap into the kernel** using a **Mach trap or syscall** (macOS is based on the XNU kernel).
3. The kernel handles the request via its **BSD socket layer**.
4. The request reaches the **network stack**, and data is sent out.

4. XNU Kernel Internals (macOS View)

Location in Kernel:

In macOS (XNU), the actual kernel function that handles sendto() is:

int sendto(struct proc *p, struct sendto_args *uap, int *retval)

Defined in:

- bsd/kern/uipc_syscalls.c
- It validates inputs, copies data from user space, checks the socket, and calls lower networking layers (e.g., udp_output() if UDP).

Related Kernel Functions:

- sockargs() – Parses and prepares the sockaddr struct.
- sosend() – Core function that handles sending data through the socket.
- udp_output() / tcp_output() – Actual protocols for sending data.