Bahir Dar University

Faculty of Computing

Department of Software Engineering

OSSP Individual Assignment

System Call Implementation

Full Name – Hiywot Teshome

Student Id – BDU1601788

Section – A

System Call – recvfrom()

Submitted To: Lecturer Wondimu B.

IMPLEMENTING SYSTEM CALL

A system call is a way for a user-space program to request a low-level service from the operating system's kernel.

A fundamental network communication tool that allows applications to receive data from a socket is recvfrom(). This report details the purpose, operation, and my experience implementing recvfrom(), which was initially intended for the OpenVMS operating system but was ultimately performed on the Windows OS due to setup constraints.

What is recvfrom()?

The system call recvfrom() is provided by various operating systems, including those adhering to the POSIX standard and Windows. Its core function is to receive a message from a socket, capturing not only the data but also the address of the sender.

With recvfrom(), an application can listen for incoming network packets on a specified socket. This system call is essential for network programming, enabling applications to communicate with other processes across a network. It allows for connectionless communication, where the sender's address is explicitly provided with each incoming message.

How recvfrom() works?

When recvfrom() is called, the operating system waits until some data is received on the specified socket. Once data arrives, the system call copies the data into a temporary storage area provided by the user application and also fills in a structure with the sender's address, so the program knows where the data came from.

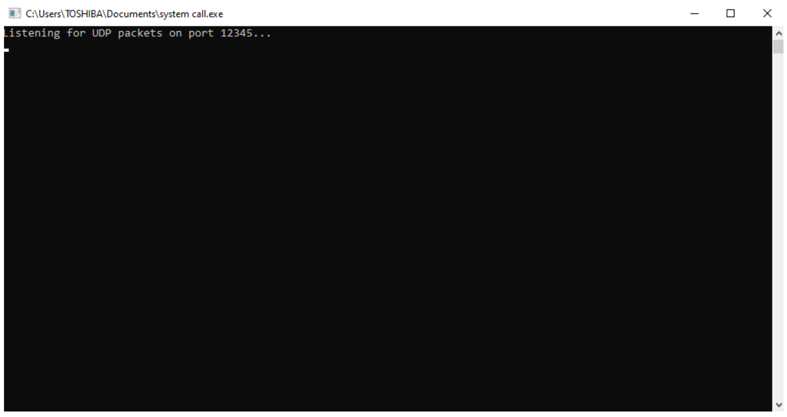
Switching to Windows: A Result of OpenVMS Virtualization Challenges

I was tasked to perform the recvfrom() system call within the OpenVMS operating system. However, I encountered significant obstacles while attempting to install and configure OpenVMS in a suitable environment:

- **Missing or disabled hardware virtualization (VT-x/AMD-V)** on my system, which is essential for running virtual machines.
- **Hypervisor not available** error, indicating that the virtual machine could not access the necessary virtualization environment.
- **Repeated security-related failures** due to VirtualBox's hardening process, which failed to verify the system's memory and environment integrity.
- **The same error persisted** even after switching to a different device.

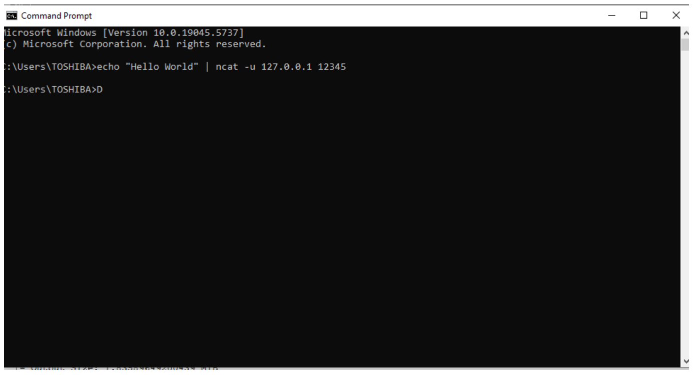
Because of the persistent issues above, I decided to use Windows, a stable and widely supported operating system, to carry out my work instead.

After running the system call on Windows, the following initial output was generated:



```
C:\Users\TOSHIBA\Documents\system call.exe
Listening for UDP packets on port 12345...
```

To test the call, the command echo "Hello World" | ncat -u 127.0.0.1 12345 was executed in the command prompt. This command's purpose is to send the text 'Hello World' as a UDP packet to the local computer (127.0.0.1) via port 12345.

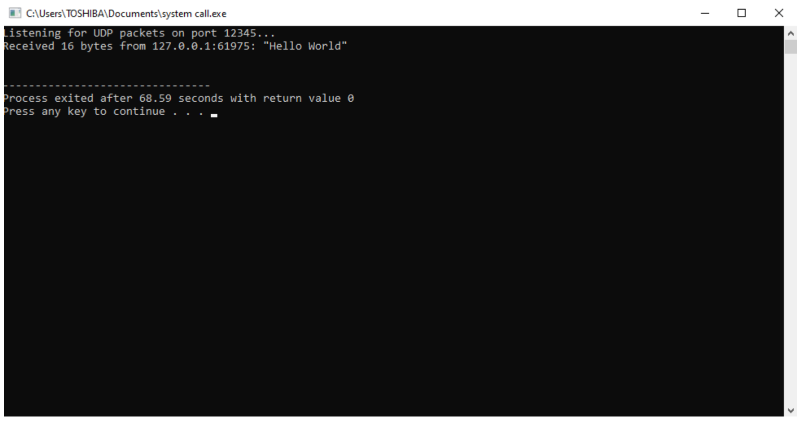


```
Command Prompt
Microsoft Windows [Version 10.0.19045.5737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TOSHIBA>echo "Hello World" | ncat -u 127.0.0.1 12345

C:\Users\TOSHIBA>D
```

The sent message is then expected to appear in the compiler's output.



```
C:\Users\TOSHIBA\Documents\system call.exe
Listening for UDP packets on port 12345...
Received 16 bytes from 127.0.0.1:61975: "Hello World"


--------------------------------
Process exited after 68.59 seconds with return value 0
Press any key to continue . . .
```