Bahir Dar University
Bahir Dar Institute of Technology (Bit)
Faculty of Computing
Department of Software Engineering
Operating system individual Assignment

TITLE:Project work on installation of NetBSD OB

PREPARED BY: ***ID***

1. VERONICA ZELALEM ***1602647***

Submitted to: **Lec Wondimu Baye**
***Submission date: April 30*/2016EC**

1. Introduction

❖

❖ Background

NetBSD is a free, open-source operating system that has earned a reputation for its flexibility, reliability, and focus on portability. It belongs to the BSD (Berkeley Software Distribution) family, which has roots in the original Unix operating system, developed in the 1970s at AT&T Bell Labs. Over the years, the BSD family grew into a collection of Unix-like operating systems, each offering unique features and improvements over the original Unix design.

NetBSD was first released in 1993, created by a team of developers, including Charles M. Hannum, who was particularly driven to build an operating system that was not only open and accessible but also capable of running on a wide range of hardware. What makes NetBSD truly unique is its dedication to portability. While many operating systems are limited to certain hardware or devices, NetBSD is designed to run on over 70 different platforms, from modern Intel and ARM processors to legacy systems like the VAX and PDP-11. This level of versatility allows it to be used in everything from personal computers to embedded devices, making it a favorite for a wide variety of applications.

❖ Motivation Behind NetBSD

The driving force behind the creation and ongoing development of NetBSD stems from several core goals that the project's developers hold dear:

- Portability: The main reason NetBSD exists is to create an operating system that can run on a wide variety of hardware. In an era where many other operating systems were tied to specific hardware or processors, the creators of NetBSD saw the potential for an OS that could break these boundaries. Their vision was to build a system that could be adapted to any platform, whether it's an old server, a modern desktop, or a specialized embedded device. The sheer number of hardware platforms NetBSD supports is a testament to how successful this vision has been.

- Clean, Maintainable Code: NetBSD's developers value simplicity, clarity, and maintainability. The system's source code is known for being elegant and well-structured, which makes it easier for developers to understand and contribute to. This approach ensures that the operating system remains stable and reliable, even as it evolves. For many, this focus on clean code also makes NetBSD an excellent resource for learning how operating systems work, especially for students and developers new to the field.and also there are others motivations:Security and Stability,Community and Open Source Philosophy, Educational Value.

1.B Objective

The main goal of NetBSD is to create an operating system that is highly portable, meaning it can run on a wide range of hardware, from modern computers to older or niche devices. It focuses on being secure, stable, and easy to maintain, with clean and well-organized code. Beyond that, NetBSD aims to provide excellent performance while staying simple and efficient.

Additional objectives include fostering a strong open-source community, where developers from around the world can collaborate, contribute, and improve the system. NetBSD also prioritizes flexibility, allowing users to adapt the OS to suit various needs, whether for personal projects, academic research, or critical infrastructure. It is designed to be a reliable, transparent platform, offering both power and versatility in diverse computing environments.

1.C Requirements

➢ **Hardware Requirements for NetBSD OS:**

✓ Processor: NetBSD works on a wide range of processors, from older ones like the Intel 386 to modern 64-bit processors (Intel, AMD, ARM, etc.). You don’t need the latest high-end CPU; even older ones from the past couple of decades will run NetBSD, though newer systems provide better performance.

✓ Memory (RAM): NetBSD can run on systems with as little as 16 MB of RAM for older setups, but for a smoother experience on modern hardware, it's best to have at least 256 MB. More RAM is always better, especially if you plan to run additional applications.

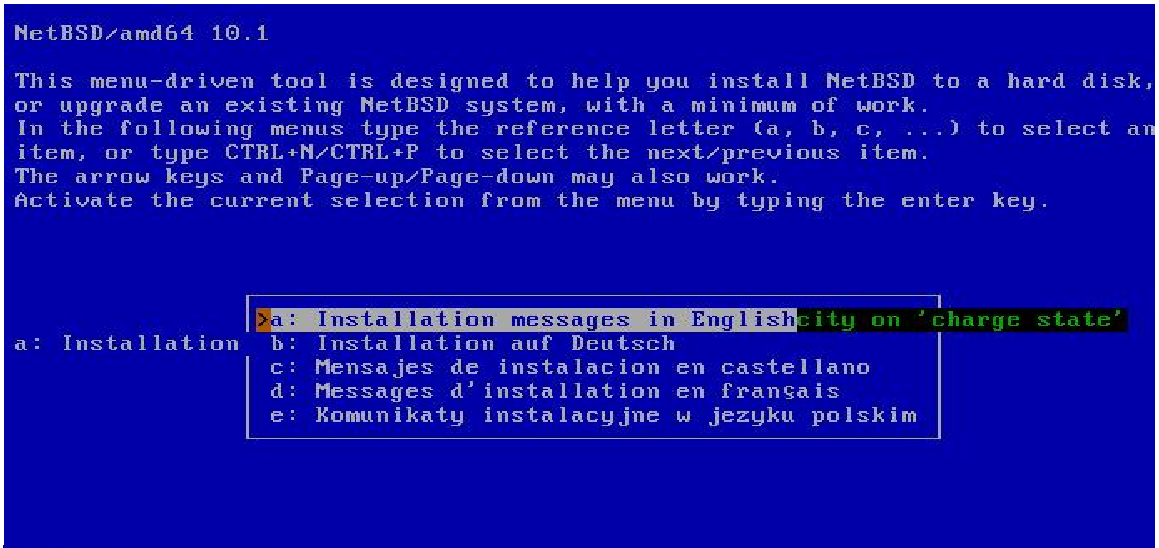
✓ Storage: The operating system can be installed on almost any type of storage, from old hard drives to newer SSDs, or even USB drives. For a basic installation, around 1 GB of disk space is recommended, but this will grow depending on what additional software you want to install.

- ✓ Graphics: NetBSD supports most basic graphics hardware. If you have older or basic graphics hardware, you’ll likely be fine. For newer or specialized hardware, you might need to configure additional drivers to get everything working smoothly.
- ✓ Network: A network card (Ethernet or Wi-Fi) is needed for online access. NetBSD supports a wide variety of network interfaces, but for some newer or less common models, you may need to manually configure the appropriate drivers.

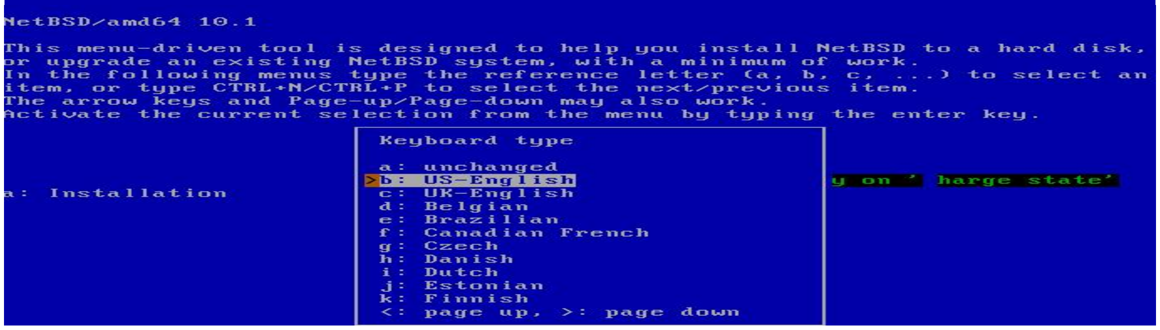
➢ **Software Requirements for NetBSD OS:**

- ✓ Bootloader: When you install NetBSD, it requires a bootloader, which is a small program that starts the OS when you power up your computer. It’s typically set up automatically during installation based on your system.
- ✓ Filesystem: NetBSD supports a variety of file systems, like FFS (Fast File System) or LFS (Log-structured File System). You can choose the one that fits your needs when installing, depending on what you're planning to do with your system.
- ✓ Kernel: The kernel is the heart of the operating system. NetBSD comes with its own kernel, designed to be simple and efficient, handling everything from memory management to hardware control. This makes it highly adaptable to different types of hardware.
- ✓ Utilities & Package Manager: NetBSD comes with essential system tools out of the box. For additional software, you’ll use pkgsrc, a package management system that makes installing new programs easy.
- ✓ Drivers: To make sure all your hardware works, you’ll need to install the right drivers. NetBSD has many built-in drivers, but for newer hardware, you may need to manually configure or install additional drivers. Don’t worry—there’s plenty of support and documentation to help with that.

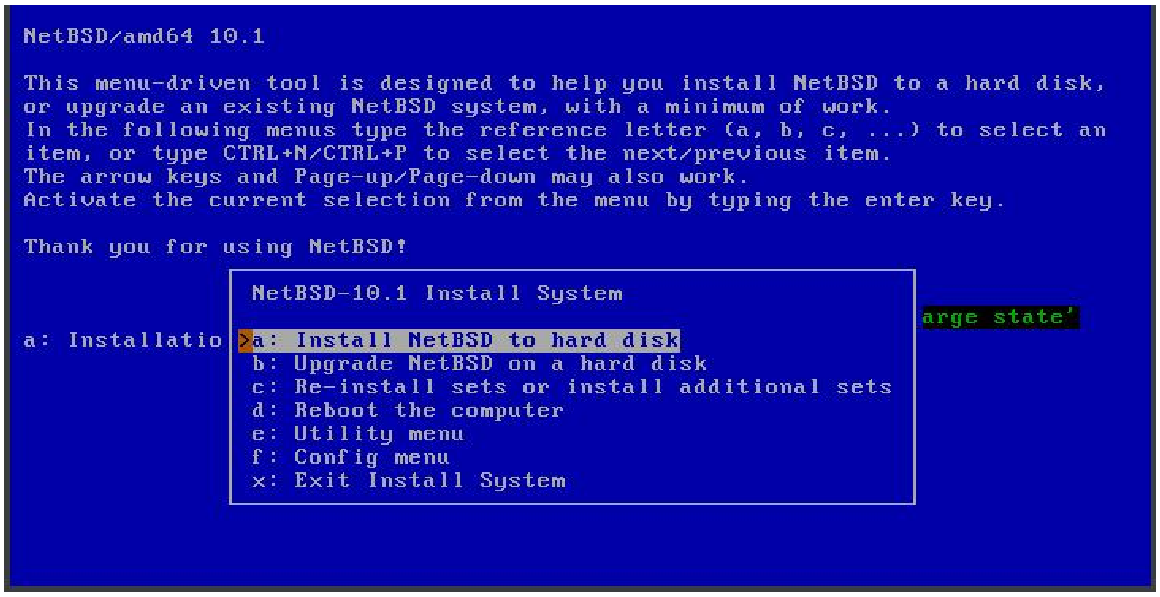
D installetion steps



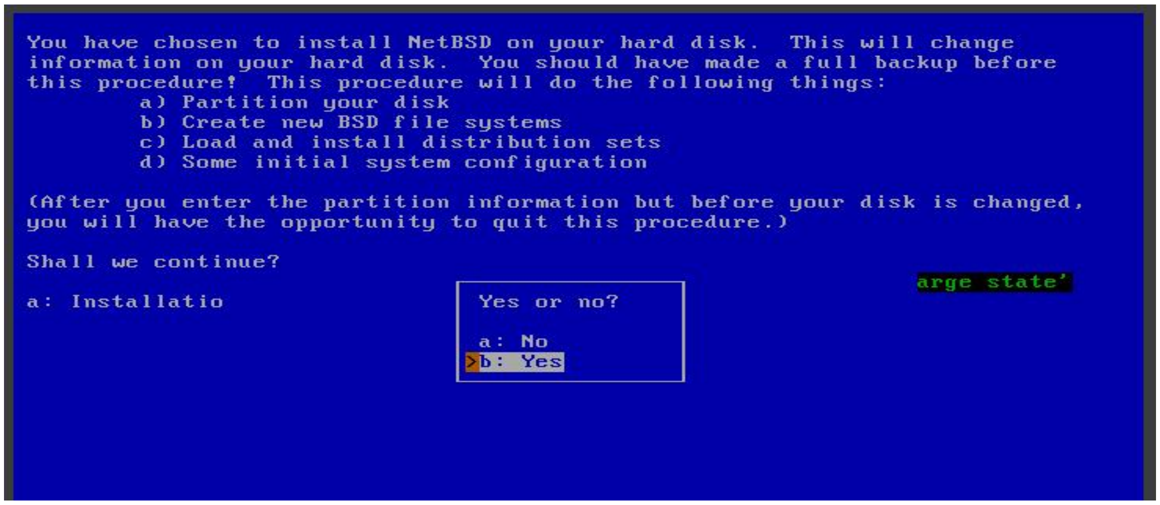
```
NetBSD/amd64 10.1

This menu-driven tool is designed to help you install NetBSD to a hard disk,
or upgrade an existing NetBSD system, with a minimum of work.
In the following menus type the reference letter (a, b, c, ...) to select an
item, or type CTRL+N/CTRL+P to select the next/previous item.
The arrow keys and Page-up/Page-down may also work.
Activate the current selection from the menu by typing the enter key.


                    >a: Installation messages in Englishcity on 'charge state'
a: Installation      b: Installation auf Deutsch
                     c: Mensajes de instalacion en castellano
                     d: Messages d'installation en français
                     e: Komunikaty instalacyjne w jezyku polskim
```



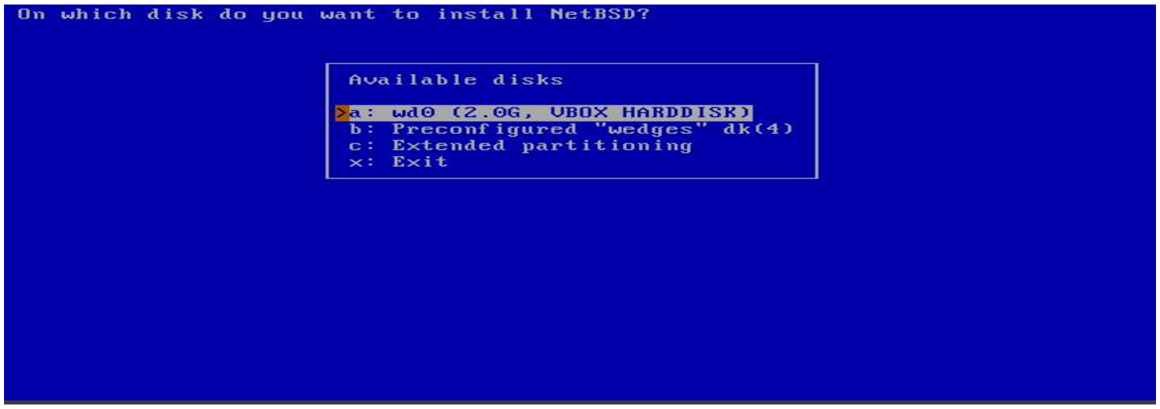
```
NetBSD/amd64 10.1

This menu-driven tool is designed to help you install NetBSD to a hard disk,
or upgrade an existing NetBSD system, with a minimum of work.
In the following menus type the reference letter (a, b, c, ...) to select an
item, or type CTRL+N/CTRL+P to select the next/previous item.
The arrow keys and Page-up/Page-down may also work.
Activate the current selection from the menu by typing the enter key.
                          Keyboard type

                          a: unchanged
                         >b: US-English                y on ' harge state'
a: Installation           c: UK-English
                          d: Belgian
                          e: Brazilian
                          f: Canadian French
                          g: Czech
                          h: Danish
                          i: Dutch
                          j: Estonian
                          k: Finnish
                          <: page up, >: page down
```



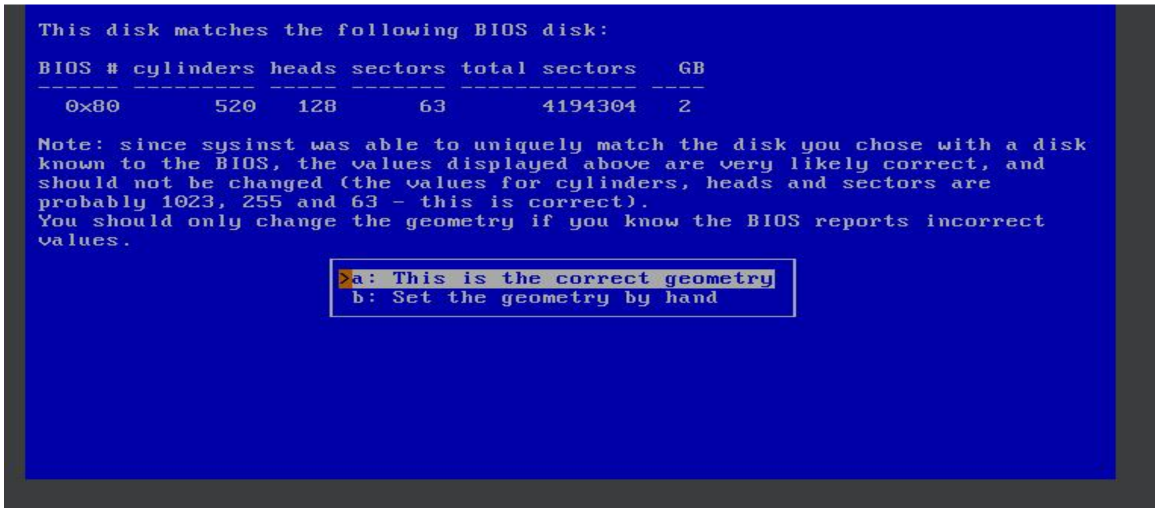
```
NetBSD/amd64 10.1

This menu-driven tool is designed to help you install NetBSD to a hard disk,
or upgrade an existing NetBSD system, with a minimum of work.
In the following menus type the reference letter (a, b, c, ...) to select an
item, or type CTRL+N/CTRL+P to select the next/previous item.
The arrow keys and Page-up/Page-down may also work.
Activate the current selection from the menu by typing the enter key.

Thank you for using NetBSD!

                 NetBSD-10.1 Install System
                                                      arge state'
a: Installatio  >a: Install NetBSD to hard disk
                 b: Upgrade NetBSD on a hard disk
                 c: Re-install sets or install additional sets
                 d: Reboot the computer
                 e: Utility menu
                 f: Config menu
                 x: Exit Install System
```



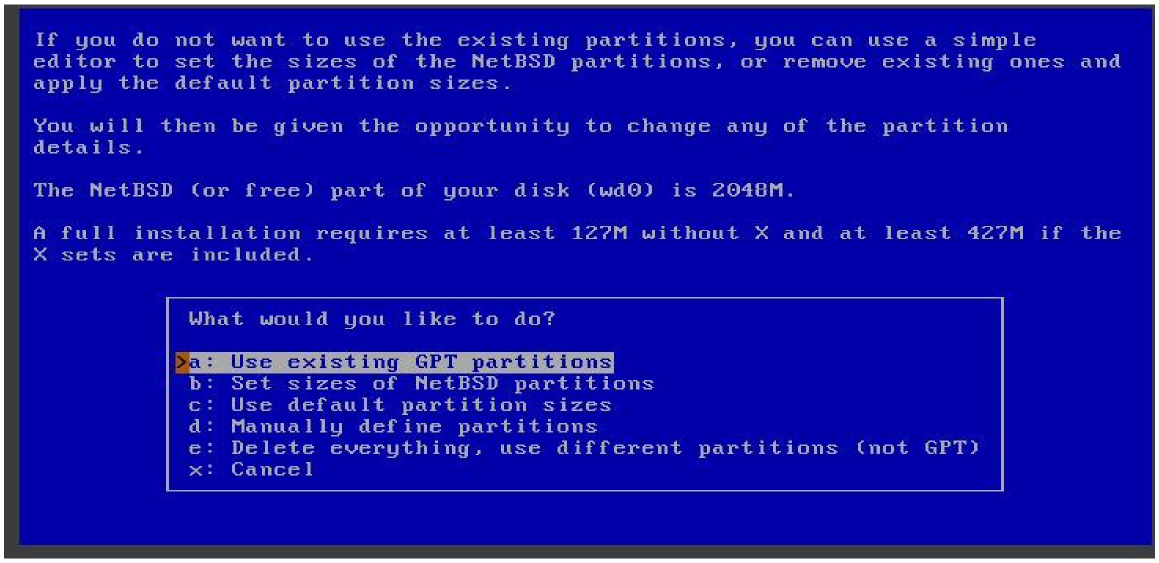
```
You have chosen to install NetBSD on your hard disk.  This will change
information on your hard disk.  You should have made a full backup before
this procedure!  This procedure will do the following things:
        a) Partition your disk
        b) Create new BSD file systems
        c) Load and install distribution sets
        d) Some initial system configuration

(After you enter the partition information but before your disk is changed,
you will have the opportunity to quit this procedure.)

Shall we continue?
                                                                arge state'
a: Installatio               Yes or no?

                             a: No
                            >b: Yes
```



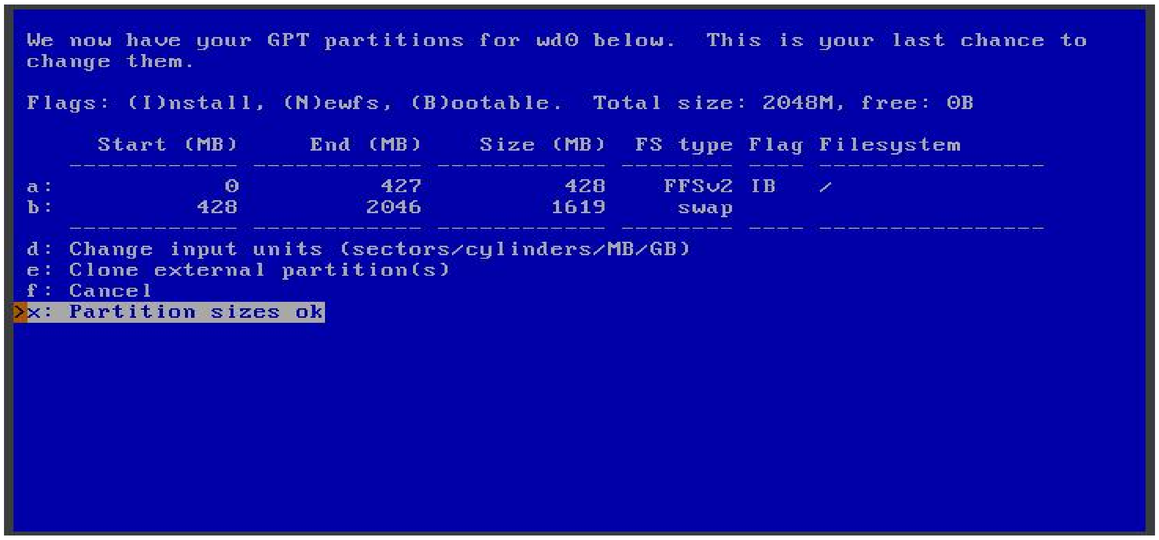
```
On which disk do you want to install NetBSD?

                 Available disks

                >a: wd0 (2.0G, VBOX HARDDISK)
                 b: Preconfigured "wedges" dk(4)
                 c: Extended partitioning
                 x: Exit
```



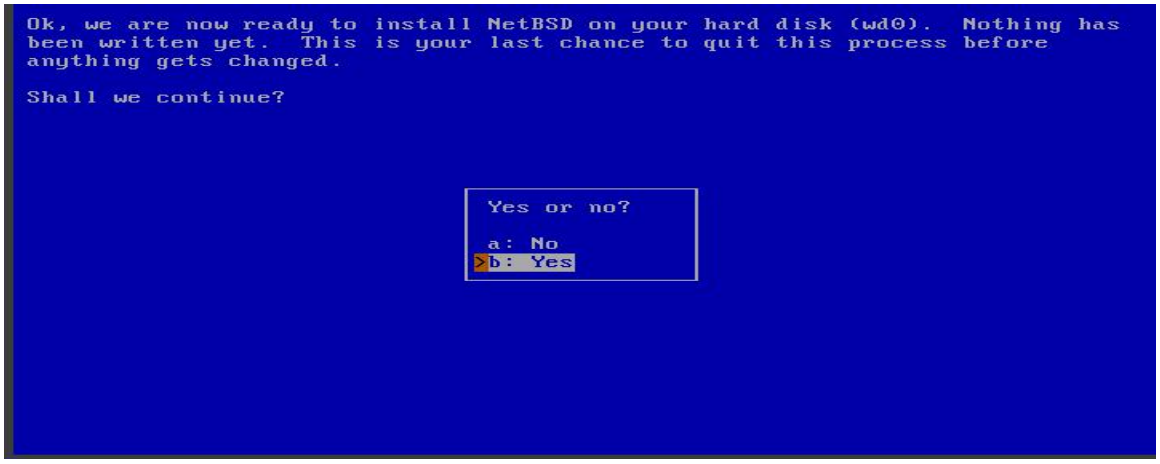
```
This disk matches the following BIOS disk:

BIOS # cylinders heads sectors total sectors   GB
------ --------- ----- ------- ------------- ----
  0x80       520   128      63       4194304    2

Note: since sysinst was able to uniquely match the disk you chose with a disk
known to the BIOS, the values displayed above are very likely correct, and
should not be changed (the values for cylinders, heads and sectors are
probably 1023, 255 and 63 - this is correct).
You should only change the geometry if you know the BIOS reports incorrect
values.
                    >a: This is the correct geometry
                     b: Set the geometry by hand
```



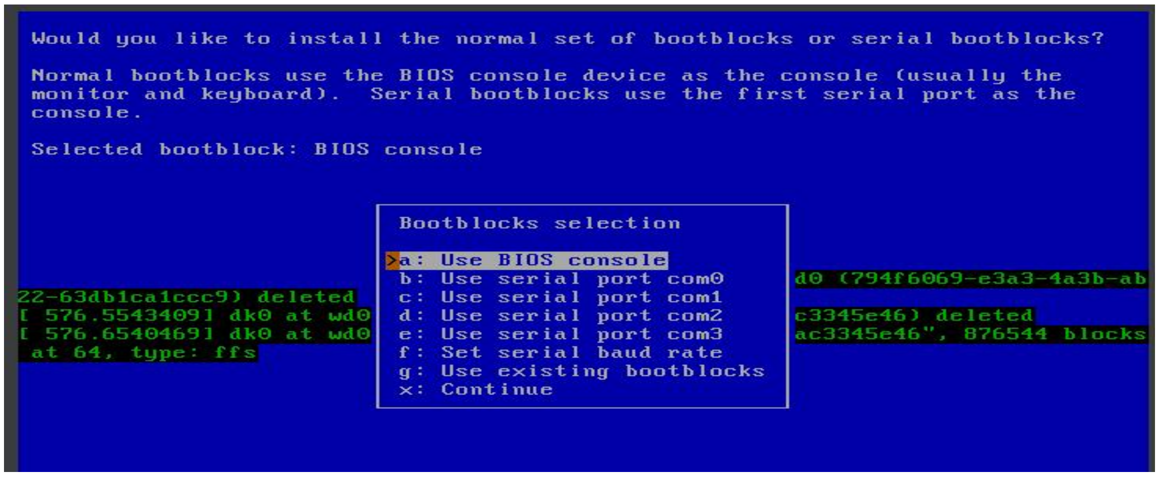
```
If you do not want to use the existing partitions, you can use a simple
editor to set the sizes of the NetBSD partitions, or remove existing ones and
apply the default partition sizes.

You will then be given the opportunity to change any of the partition
details.

The NetBSD (or free) part of your disk (wd0) is 2048M.

A full installation requires at least 127M without X and at least 427M if the
X sets are included.

          What would you like to do?

         >a: Use existing GPT partitions
          b: Set sizes of NetBSD partitions
          c: Use default partition sizes
          d: Manually define partitions
          e: Delete everything, use different partitions (not GPT)
          x: Cancel
```



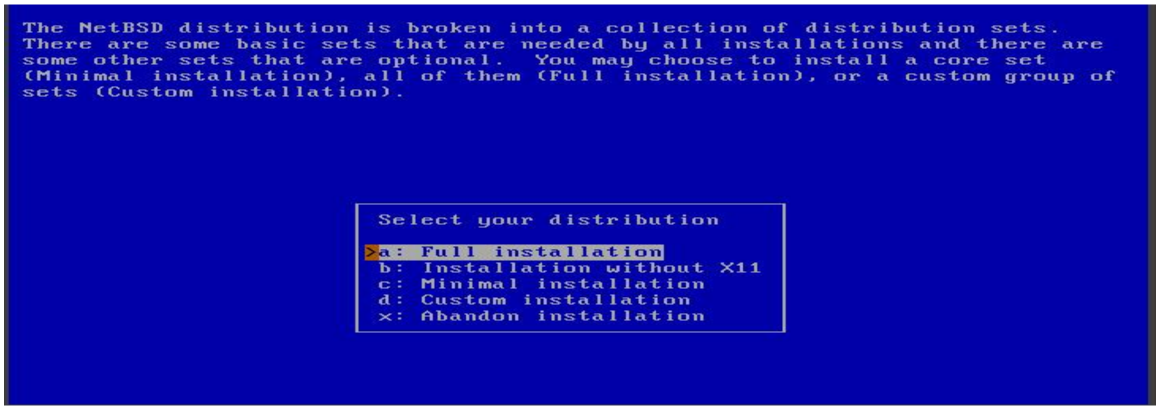
```
We now have your GPT partitions for wd0 below.  This is your last chance to
change them.

Flags: (I)nstall, (N)ewfs, (B)ootable.  Total size: 2048M, free: 0B

     Start (MB)     End (MB)     Size (MB)  FS type Flag Filesystem
   ------------ ------------ ------------ -------- ---- ----------------
a:            0          427          428    FFSv2 IB   /
b:          428         2046         1619     swap
   ------------ ------------ ------------ -------- ---- ----------------
d: Change input units (sectors/cylinders/MB/GB)
e: Clone external partition(s)
f: Cancel
>x: Partition sizes ok
```



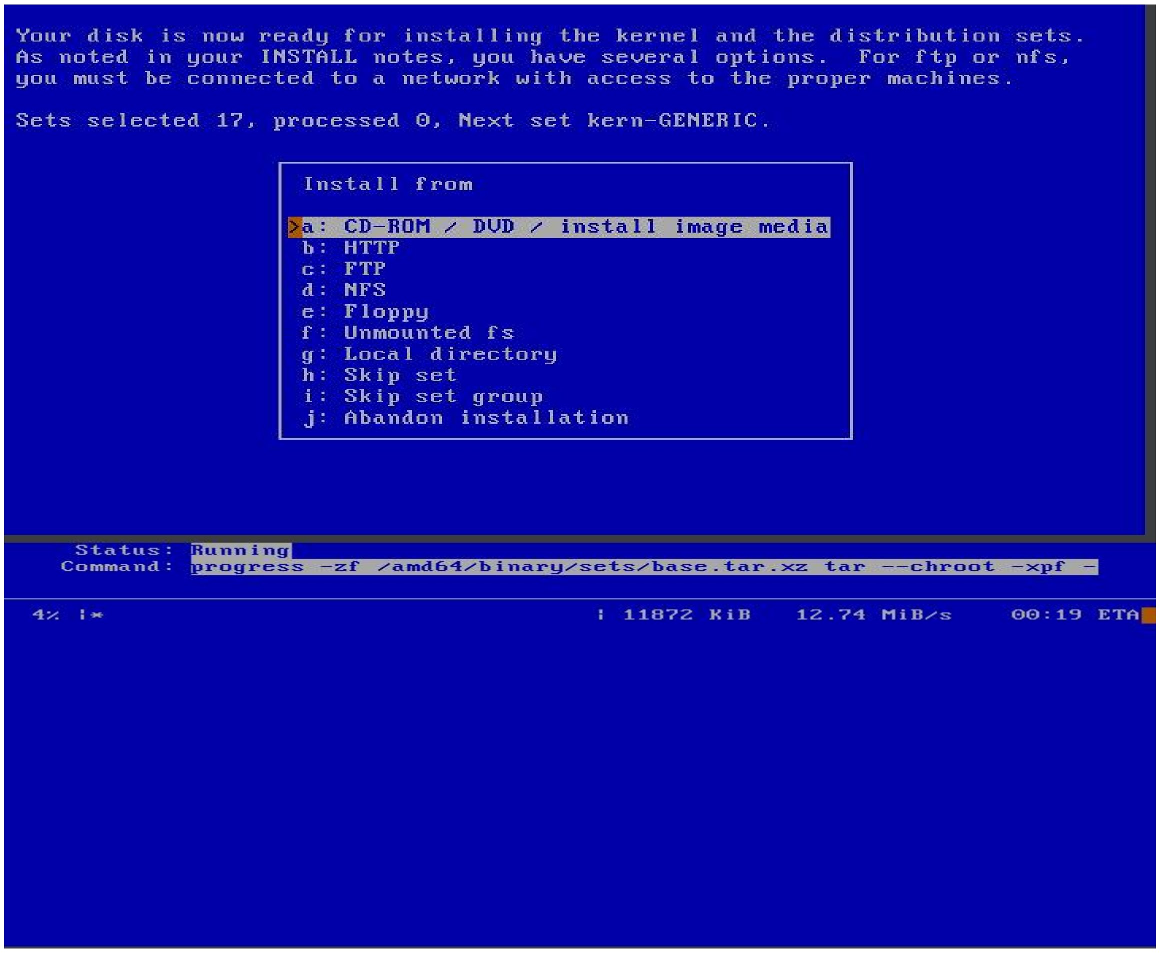
```
Ok, we are now ready to install NetBSD on your hard disk (wd0).  Nothing has
been written yet.  This is your last chance to quit this process before
anything gets changed.

Shall we continue?

                              Yes or no?

                              a: No
                             >b: Yes
```



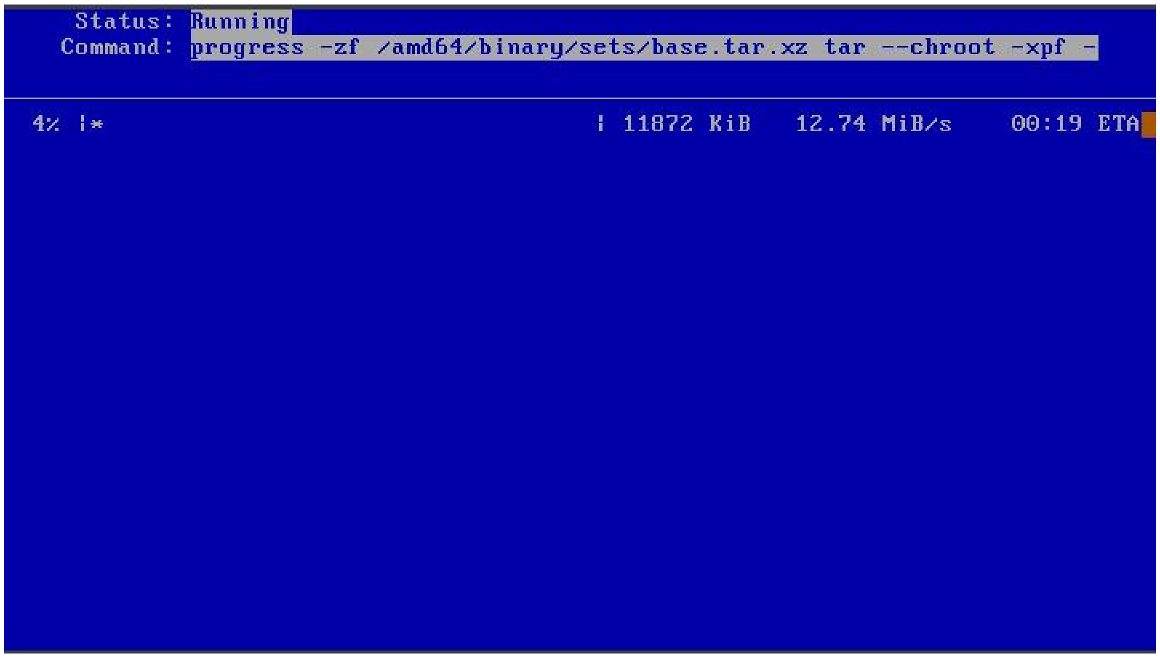
```
Would you like to install the normal set of bootblocks or serial bootblocks?

Normal bootblocks use the BIOS console device as the console (usually the
monitor and keyboard).  Serial bootblocks use the first serial port as the
console.

Selected bootblock: BIOS console


                          Bootblocks selection

                         >a: Use BIOS console
                          b: Use serial port com0     d0 (794f6069-e3a3-4a3b-ab
22-63db1ca1ccc9) deleted  c: Use serial port com1
[ 576.5543409] dk0 at wd0 d: Use serial port com2     c3345e46) deleted
[ 576.6540469] dk0 at wd0 e: Use serial port com3     ac3345e46", 876544 blocks
 at 64, type: ffs         f: Set serial baud rate
                          g: Use existing bootblocks
                          x: Continue
```



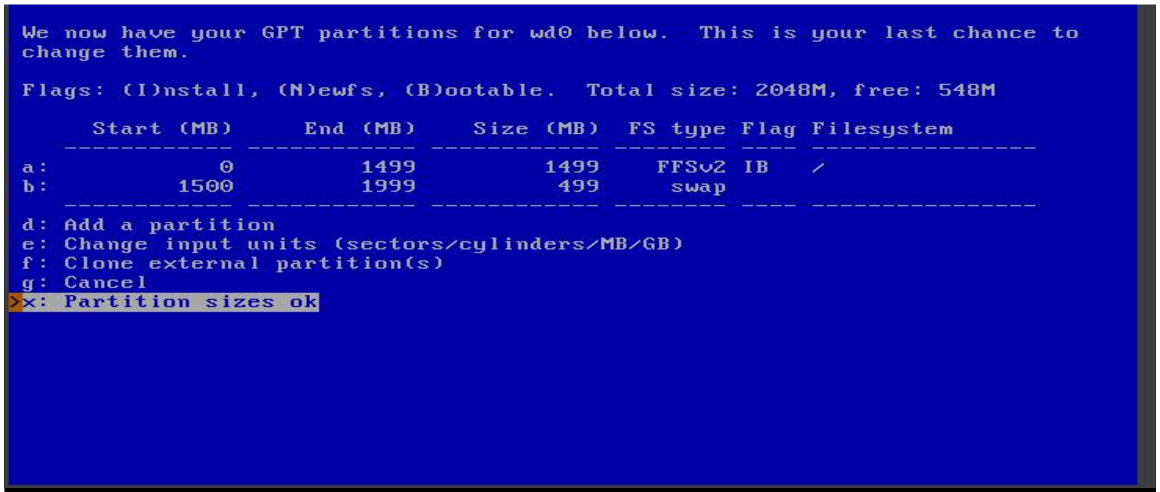
```
The NetBSD distribution is broken into a collection of distribution sets.
There are some basic sets that are needed by all installations and there are
some other sets that are optional.  You may choose to install a core set
(Minimal installation), all of them (Full installation), or a custom group of
sets (Custom installation).


                    Select your distribution

                   >a: Full installation
                    b: Installation without X11
                    c: Minimal installation
                    d: Custom installation
                    x: Abandon installation
```

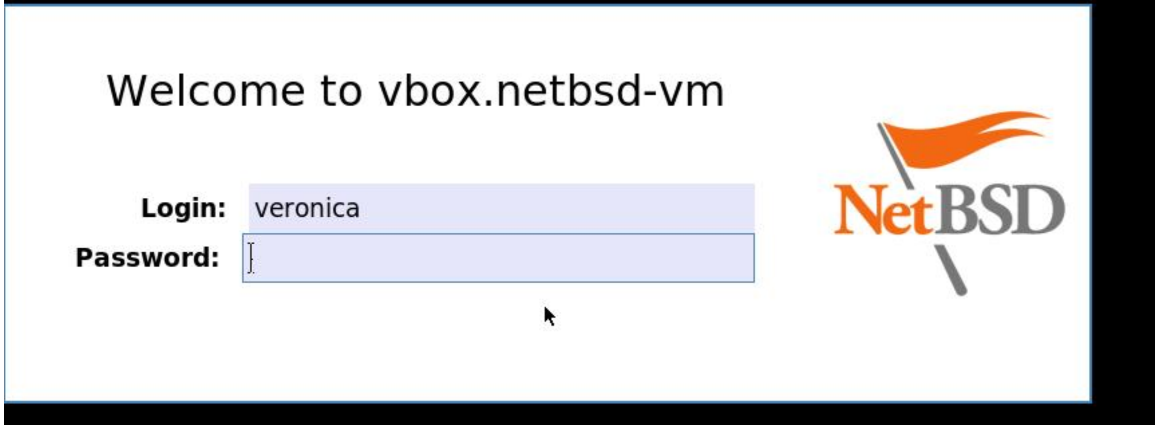


```
Your disk is now ready for installing the kernel and the distribution sets.
As noted in your INSTALL notes, you have several options.  For ftp or nfs,
you must be connected to a network with access to the proper machines.

Sets selected 17, processed 0, Next set kern-GENERIC.

                    Install from

                   >a: CD-ROM / DVD / install image media
                    b: HTTP
                    c: FTP
                    d: NFS
                    e: Floppy
                    f: Unmounted fs
                    g: Local directory
                    h: Skip set
                    i: Skip set group
                    j: Abandon installation

     Status: Running
    Command: progress -zf /amd64/binary/sets/base.tar.xz tar --chroot -xpf -

  4% |*                                        | 11872 KiB   12.74 MiB/s    00:19 ETA
```



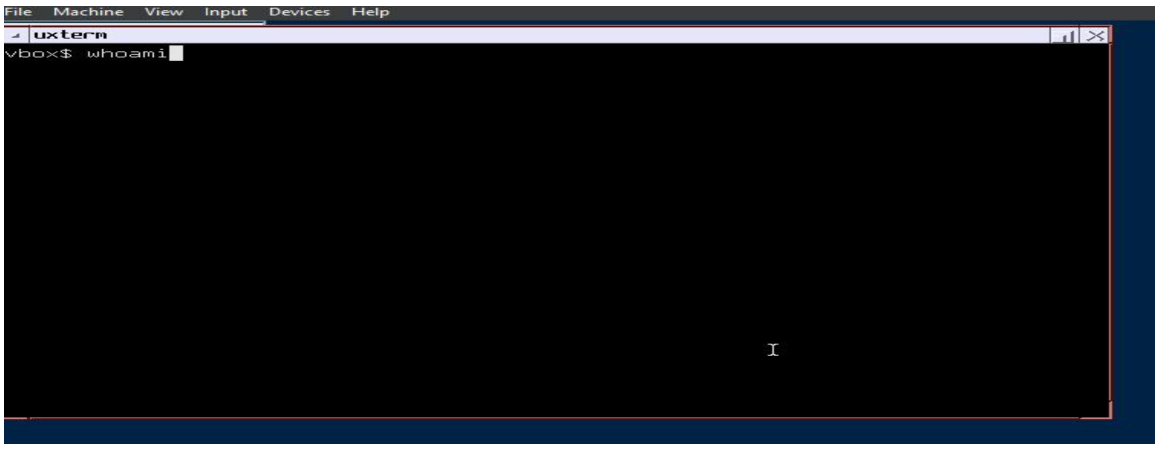
```
     Status: Running
    Command: progress -zf /amd64/binary/sets/base.tar.xz tar --chroot -xpf -

  4% |*                                        | 11872 KiB   12.74 MiB/s    00:19 ETA
```


```
We now have your GPT partitions for wd0 below.  This is your last chance to
change them.

Flags: (I)nstall, (N)ewfs, (B)ootable.  Total size: 2048M, free: 548M

     Start (MB)     End (MB)     Size (MB)   FS type Flag Filesystem
   ------------ ------------ ------------ -------- ---- ----------------
a:            0         1499         1499   FFSv2  IB   /
b:         1500         1999          499    swap
   ------------ ------------ ------------ -------- ---- ----------------
d: Add a partition
e: Change input units (sectors/cylinders/MB/GB)
f: Clone external partition(s)
g: Cancel
>x: Partition sizes ok
```

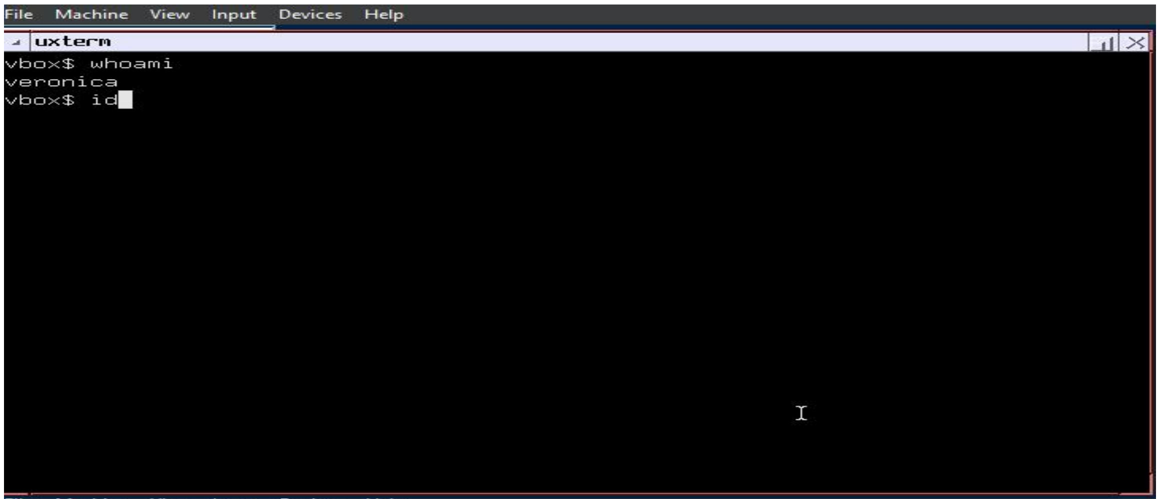

Welcome to vbox.netbsd-vm

Login: veronica

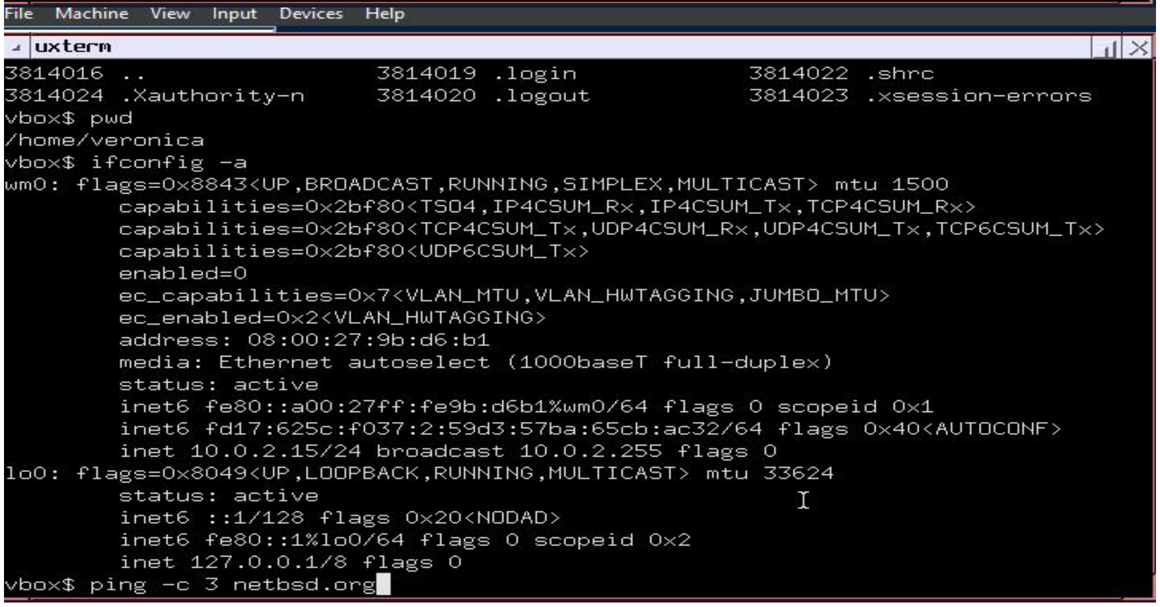
Password:



```
vbox$ whoami
```



```
vbox$ whoami
veronica
vbox$ id
```



```
3814016 ..                3814019 .login           3814022 .shrc
3814024 .Xauthority-n     3814020 .logout          3814023 .xsession-errors
vbox$ pwd
/home/veronica
vbox$ ifconfig -a
wm0: flags=0x8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        capabilities=0x2bf80<TSO4,IP4CSUM_Rx,IP4CSUM_Tx,TCP4CSUM_Rx>
        capabilities=0x2bf80<TCP4CSUM_Tx,UDP4CSUM_Rx,UDP4CSUM_Tx,TCP6CSUM_Tx>
        capabilities=0x2bf80<UDP6CSUM_Tx>
        enabled=0
        ec_capabilities=0x7<VLAN_MTU,VLAN_HWTAGGING,JUMBO_MTU>
        ec_enabled=0x2<VLAN_HWTAGGING>
        address: 08:00:27:9b:d6:b1
        media: Ethernet autoselect (1000baseT full-duplex)
        status: active
        inet6 fe80::a00:27ff:fe9b:d6b1%wm0/64 flags 0 scopeid 0x1
        inet6 fd17:625c:f037:2:59d3:57ba:65cb:ac32/64 flags 0x40<AUTOCONF>
        inet 10.0.2.15/24 broadcast 10.0.2.255 flags 0
lo0: flags=0x8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33624
        status: active
        inet6 ::1/128 flags 0x20<NODAD>
        inet6 fe80::1%lo0/64 flags 0 scopeid 0x2
        inet 127.0.0.1/8 flags 0
vbox$ ping -c 3 netbsd.org
```

E ISSUES (PROBLEM FACED)

I. Potential Problems I faced when I am working the project work

- VM Performance Issues: The Virtual Machine is very slow, and kernel compilation takes an excessively long time.
- Incorrect NetBSD ISO Downloaded: The VM fails to boot from the ISO, or the NetBSD installation fails with architecture mismatch errors.
- NetBSD Installation Errors: The NetBSD installation process hangs, fails during partitioning, or encounters errors while extracting package sets.
- Kernel Source Download/Extraction Issues: The ftp command fails to download the source tarballs, or tar reports errors like "corrupted file" during extraction.

- Typos in Code or Configuration Files: Kernel compilation fails with syntax errors, reports undeclared identifiers, or encounters linking errors. The syscalls.master file might cause parsing errors during build.
- Incorrect System Call Number Usage: Your user-space test program fails with errors like "Function not implemented" or "Invalid argument," or it appears to be invoking a different system call.
- Kernel Build Failures (General): The build.sh script or make command fails during kernel compilation with errors that are not immediately clear.
- Forgetting to Install New Kernel or Reboot: The user-space test program fails to find or execute the new system call, and uname -a still shows the old kernel version/identifier.
- Kernel Panic on Boot or During System Call Execution: The VM crashes, displays a "panic" message, or spontaneously reboots either when trying to boot with the new kernel or when the custom system call is invoked by the test program.
- Permissions Issues: You encounter "Permission denied" errors when trying to edit system files, create directories in system locations, or run commands like make install.
- User-space Test Program Fails to Compile (SYS_your_syscall Undeclared): The C compiler complains that SYS_my_first_syscall (or your syscall's equivalent macro) is an undeclared identifier when compiling test_syscall.c.

- Forgetting make depend or make dependall (If Using Manual make Flow): If you are using the older config MYKERNEL; cd ../compile/MYKERNEL; make flow, the build might fail due to unresolved dependencies or using stale object files if dependencies weren't updated.
- Disk Space Full During Compilation or Operation: Operations, especially kernel compilation, fail with messages like "No space left on device."
- visudo Syntax Error Locks Out sudo: After editing the /etc/sudoers file, sudo no longer works, and you get syntax error messages

Solutions for that problems

II. Solutions to Potential Problems

- For VM Performance Issues:

Allocate more RAM to the VM in VirtualBox settings (e.g., 2GB or more if your host has 8GB+).

Allocate more CPU cores to the VM if your host processor has multiple cores (e.g., 2 cores if your host has 4 or more).

Ensure your host machine is not overloaded by other applications. Close unnecessary programs on the host.

- For Incorrect NetBSD ISO Downloaded:

Verify that you downloaded the correct ISO image for your VM's architecture setting in VirtualBox (e.g., amd64.iso for a "NetBSD (64-bit)" VM type, or i386.iso for "NetBSD (32-bit)").

❖ For NetBSD Installation Errors:

Ensure you have allocated sufficient virtual disk space (at least 15-20GB, more is better for kernel development).

Try using the default partitioning scheme suggested by the installer.

If installing sets from the network, verify your VM's internet connectivity.

If problems persist, the downloaded ISO might be corrupted. Try downloading it again from a reliable mirror.

For Kernel Source Download/Extraction Issues:

Check your VM's internet connection.

Verify the ftp URL for the source sets is correct and the files exist on the server.

Ensure you have enough free disk space in the download location (e.g., /tmp) and the extraction target (/usr/src).

❖ For Typos in Code or Configuration Files

Meticulously review all modifications:

Check the syntax of your entry in /usr/src/sys/kern/syscalls.master.

Verify the C function signature and implementation in kern_my_syscall.c.

Ensure the SRCS+= kern_my_syscall.c line is correctly added to /usr/src/sys/kern/Makefile.kern.

Pay close attention to compiler error messages; they usually pinpoint the line and nature of the error.

❖ For Incorrect System Call Number Usage:

Confirm that the system call number you chose in syscalls.master is unique and not already in use.

After a kernel build, inspect the generated file /usr/src/sys/kern/syscalls.const (or a similar path within your build's object directory, e.g., /usr/obj/sys/kern/syscalls.const). This file will show the actual number assigned to my_first_syscall.

Ensure the SYS_my_first_syscall macro in your user-space test program (or the numeric value you directly pass to the syscall() function) precisely matches this generated number. The user-space header /usr/include/sys/syscall.h should ideally be updated by the kernel build/install process.

❖ For Kernel Build Failures (General):

Carefully read all error messages leading up to the failure.

Ensure all necessary development tools and libraries are installed (usually handled by a "Full installation" of NetBSD and build.sh's toolchain build).

Verify sufficient disk space in /usr/src, /usr/obj (if used by build.sh), and RAM for the VM.

Confirm you are executing build commands from the correct directory (e.g., /usr/src for build.sh).

Try cleaning previous build artifacts: sudo ./build.sh ... clean (in /usr/src) or sudo make clean (in the kernel compile directory like /usr/src/sys/arch/amd64/compile/MYKERNEL) and then attempt the build again.

For Forgetting to Install New Kernel or Reboot:

Double-check that you successfully copied your newly built kernel (e.g., netbsd-MYKERNEL or netbsd) to the / directory, replacing the old /netbsd.

Ensure you have rebooted the Virtual Machine after installing the new kernel.

❖ For Kernel Panic on Boot or During System Call Execution:

Enable Kernel Debugger (DDB): Add options DDB to your MYKERNEL configuration file and rebuild. If a panic occurs, you might get a db> prompt, allowing for basic inspection.

Review Your Code: The most common causes are pointer errors (e.g., dereferencing a NULL or invalid pointer), using uninitialized variables, incorrect assumptions about kernel data structures, or stack overflows within your system call implementation (kern_my_syscall.c).

Simplify: Revert to the most basic version of your system call (e.g., just returning a value, no printf) to isolate the issue.

Revert to Old Kernel: Boot the VM using your backup kernel (e.g., at the NetBSD bootloader, type boot netbsd.old or boot /netbsd.old). Then, fix the code and try rebuilding.

❖ For Permissions Issues:

Ensure you are operating as the root user (e.g., after su -) or are using sudo correctly before commands that require superuser privileges (e.g., sudo vim ..., sudo make install).

For User-space Test Program Fails to Compile (SYS_your_syscall Undeclared):

The build.sh ... kernel=MYKERNEL process, especially if also building distribution, should update the system headers, including /usr/include/sys/syscall.h.

If this header is not updated, you may need to manually locate the generated syscall.h within your build's object/destination directory (e.g., /usr/obj/destdir.amd64/usr/include/sys/syscall.h if you used build.sh -U -m amd64 ...) and copy it to /usr/include/sys/.

Alternative/Quick Fix: Find the actual system call number from the generated /usr/src/sys/kern/syscalls.const (or similar in your build obj dir) and hardcode it in your test program: result = syscall(560); (replace 560 with the correct number).

❖ For Forgetting make depend or make dependall (If Using Manual make Flow):

Always run sudo make dependall in the kernel compile directory (e.g., /usr/src/sys/arch/amd64/compile/MYKERNEL) before running sudo make if you are not using the build.sh script for the kernel.

❖ For Disk Space Full During Compilation or Operation:

Ensure you allocated a sufficiently large virtual hard disk for the VM initially (20-30GB is a good starting point for kernel dev).

Clean up unnecessary files: remove old build artifacts (make clean), delete downloaded tarballs from /tmp once extracted.
If the disk is chronically full, you might need to consider expanding the virtual disk (a more involved VirtualBox procedure) or starting over with a larger VM disk.

❖ For visudo Syntax Error Locks Out sudo:

Always use the visudo command to edit the /etc/sudoers file. visudo performs a syntax check before saving changes, preventing lockout.

1.F NetBSD Filesystem Support:

✧ NTFS: NetBSD can read NTFS drives by default, and with a tool like ntfs-3g, you can also write to them.

✧ FAT32: Fully supported for both reading and writing, making it perfect for USB drives and external storage that need to work across different devices.

✧ exFAT: It can read exFAT drives, and with an extra tool like fuse-exfat, it can also write to them.

✧ ext4: No direct support in NetBSD, but you can access ext4 drives using Linux emulation or some third-party tools.

✧ Btrfs: NetBSD doesn’t support Btrfs, as it's designed for Linux and doesn’t fit with NetBSD’s simpler approach.

✧ ZFS: ZFS isn’t natively supported, but there are third-party solutions if you really need it.

✧ HFS+: You can read and write to HFS+ drives, but there may be some limitations compared to how macOS handles them.

✧ APFS: No support for APFS, as it’s specific to Apple and not compatible with NetBSD’s design.

In short, NetBSD supports commonly used, open file systems like FAT32 and offers partial support for others through additional tools. However, it doesn’t natively support Linux-specific or proprietary systems like Btrfs, APFS, or ZFS.

1.H ADVANTAGE AND DISADVANTAGE

Advantages of NetBSD:

- ◆ Super Portable: NetBSD can run on a huge variety of devices—over 70 different platforms, from old machines to modern tech. This makes it incredibly flexible for all kinds of hardware.
- ◆ Solid Security and Stability: It's built to be secure and stable, so it’s great for environments where reliability is key. You can count on it to keep things running smoothly.
- ◆ Clean and Simple Code: The system is designed with clarity and simplicity in mind, which makes it easy to understand and modify. It’s a favorite for developers who want a clean slate.
- ◆ Completely Open-Source: NetBSD is free to use, and since it’s open-source, you can tweak the system, contribute to it, or share it with others. This has helped create a strong community around it.
- ◆ Lightweight: It doesn't come with unnecessary bloat, which makes it efficient and well-suited for specialized tasks, like running on embedded systems or low-powered devices.
- ◆ Wide Hardware Support: Whether you’re working with an old laptop or a specialized device, NetBSD can handle it, thanks to its broad hardware support.

➢ **Disadvantages of NetBSD**:

- ◆ Limited Software Options: NetBSD doesn’t have the massive software library that Linux or Windows offers, so if you’re looking for a lot of pre-packaged software, it might feel a bit limiting.
- ◆ Not Great with Modern File Systems: It doesn’t support newer file systems like Btrfs or APFS, which can be a problem if you need those advanced features.
- ◆ Smaller Community: Since NetBSD is more niche, it doesn’t have as large or active a community as Linux or Windows. That means fewer people to help with troubleshooting or share tips.

- Not the Easiest for Beginners: While it’s simple for those who know their way around an OS, NetBSD can be a bit tricky for newcomers or anyone looking for a “set it and forget it” experience.
- Driver Issues: Although it works on many types of hardware, newer devices might need some manual configuration or extra work to get everything running properly.

1.I NetBSD Summary

NetBSD is an open-source operating system that stands out for its portability, security, and simplicity. It can run on over 70 different types of hardware, making it a flexible choice for everything from old machines to modern devices. Its clean, maintainable code is a big draw for developers who appreciate a straightforward, reliable system. It supports basic file systems like FAT32 and NTFS (with extra tools) but doesn't natively support more modern systems like Btrfs, APFS, or ZFS. NetBSD is lightweight, which makes it great for specialized tasks and embedded systems, but it can feel a bit limiting when it comes to software options and user-friendliness, especially for beginners.

In short, NetBSD is perfect for those who need a stable, secure, and flexible OS, but it may not be the best fit for users looking for a smooth, easy experience or the latest file system features.

J. Future Outlook & Recommendations:

NetBSD has a solid future ahead, especially for those who value its portability, security, and simplicity. As tech evolves, it will be important for NetBSD to keep improving support for modern file systems and newer hardware to stay competitive.

For developers and tech enthusiasts, it’s still a great choice, but expanding its user base could be key. Making it more accessible to newcomers, with better tools and clearer documentation, would help bring more people on board. If NetBSD continues to adapt and integrate with newer technologies, it’ll stay relevant in the ever-changing open-source world.

2.What is Virtualization in Modern Operating Systems?

Virtualization lets a single physical computer run multiple virtual machines (VMs), each with its own operating system and applications. This is done through a tool called a hypervisor, which acts as a manager, allocating resources like CPU, memory, and storage to each virtual machine.

Why is Virtualization Important?

- ◆ Better Use of Resources: Instead of having multiple physical servers, you can run several VMs on just one machine, making the most of your hardware.
- ◆ Separation and Safety: Each VM operates independently, so if one crashes, it doesn’t affect the others—great for avoiding system-wide issues.
- ◆ Flexibility: VMs are easy to create, move, and scale, offering a lot of flexibility for businesses and developers.
- ◆ Cost Savings: Virtualization cuts down on the need for extra hardware, saving money on machines, energy, and maintenance.

How Does Virtualization Work?

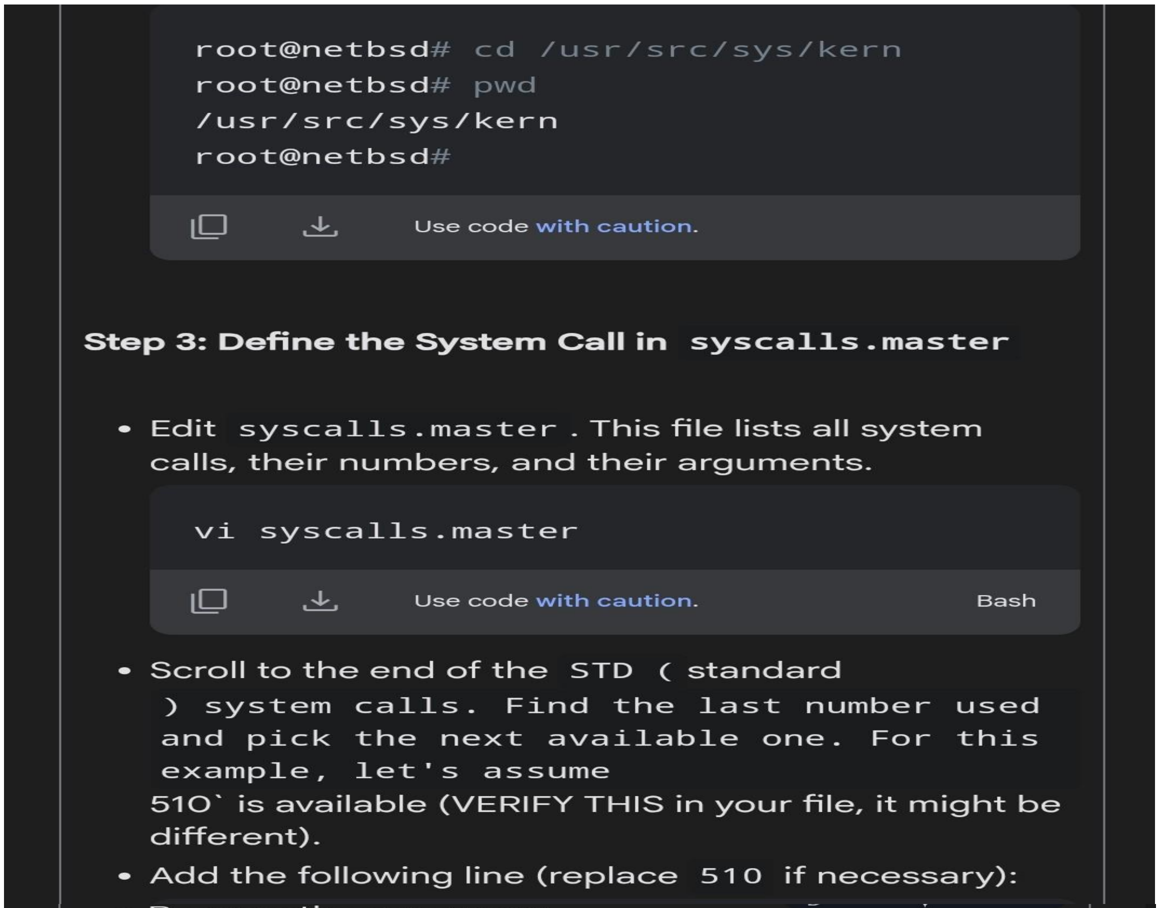
It works through a hypervisor, which comes in two flavors:

- ✓ Type 1 (bare-metal): This hypervisor runs directly on the hardware and is more efficient (think VMware ESXi or Microsoft Hyper-V).
- ✓ Type 2 (hosted): This one runs on top of an existing operating system and is easier to set up but might not perform as well (like VirtualBox or VMware Workstation).

The hypervisor creates these separate "virtual" environments, allowing different operating systems to run independently on the same machine.

In a nutshell, virtualization makes systems more efficient, flexible, and cost-effective, and is a key technology in modern operating systems.

4. Implementing system call



```
root@netbsd# cd /usr/src/sys/kern
root@netbsd# pwd
/usr/src/sys/kern
root@netbsd#
```

Use code with caution.

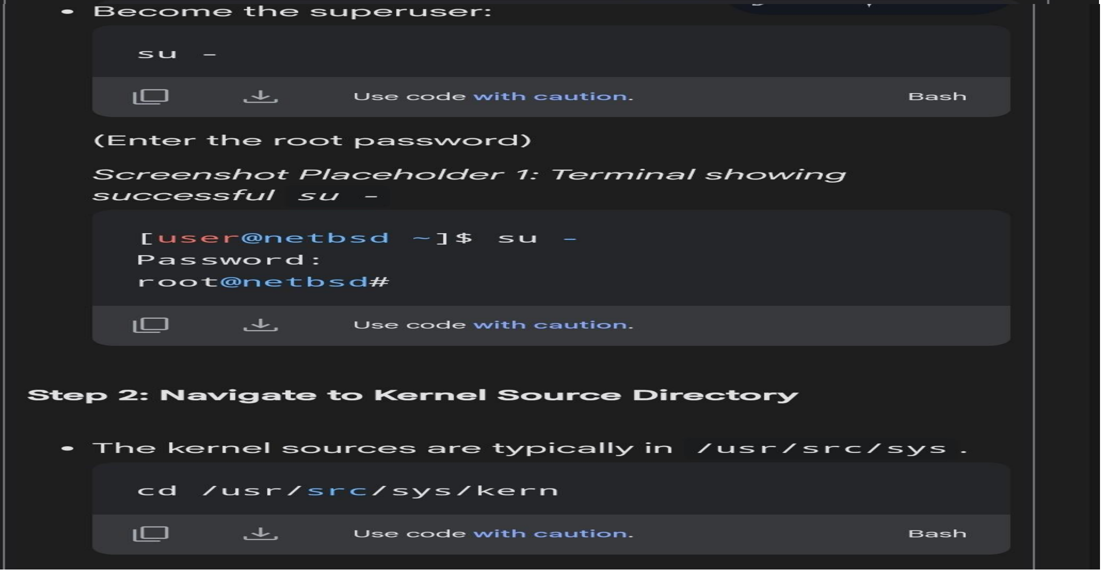
Step 3: Define the System Call in `syscalls.master`

- Edit `syscalls.master`. This file lists all system calls, their numbers, and their arguments.

```
vi syscalls.master
```

Use code with caution. Bash

- Scroll to the end of the `STD ( standard ) system calls. Find the last number used and pick the next available one. For this example, let's assume` 510` is available (VERIFY THIS in your file, it might be different).
- Add the following line (replace `510` if necessary):



- Become the superuser:

```
su -
```

Use code with caution. Bash

(Enter the root password)

Screenshot Placeholder 1: Terminal showing successful `su -`

```
[user@netbsd ~]$ su -
Password:
root@netbsd#
```

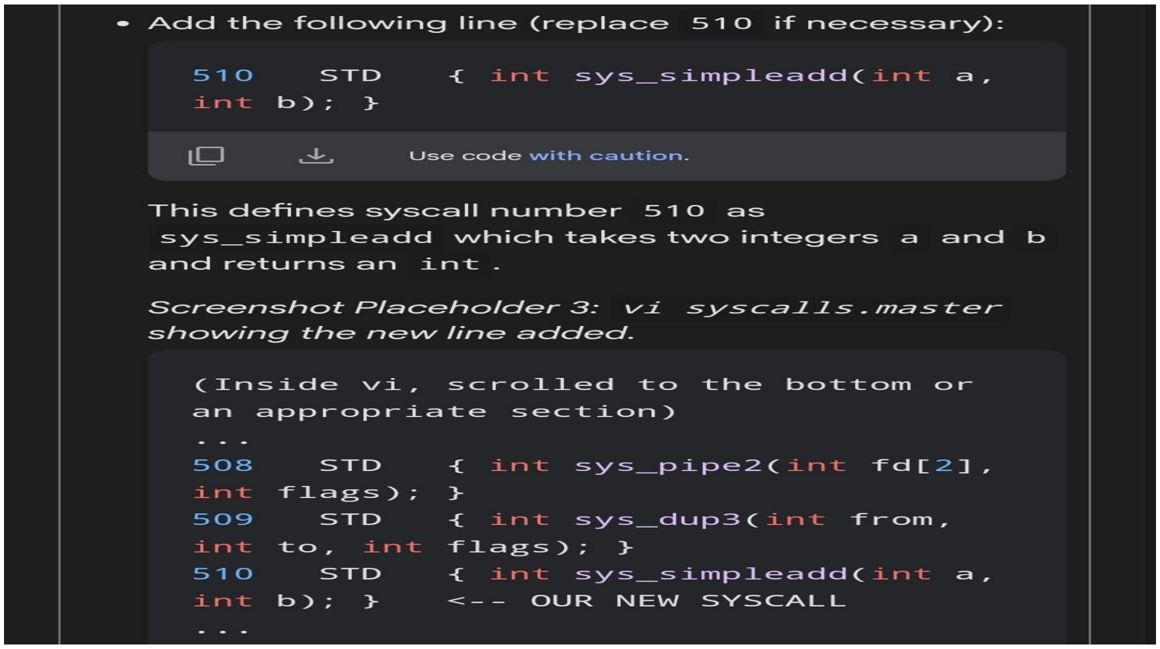
Use code with caution.

Step 2: Navigate to Kernel Source Directory

- The kernel sources are typically in `/usr/src/sys`.

```
cd /usr/src/sys/kern
```

Use code with caution. Bash



- Add the following line (replace `510` if necessary):

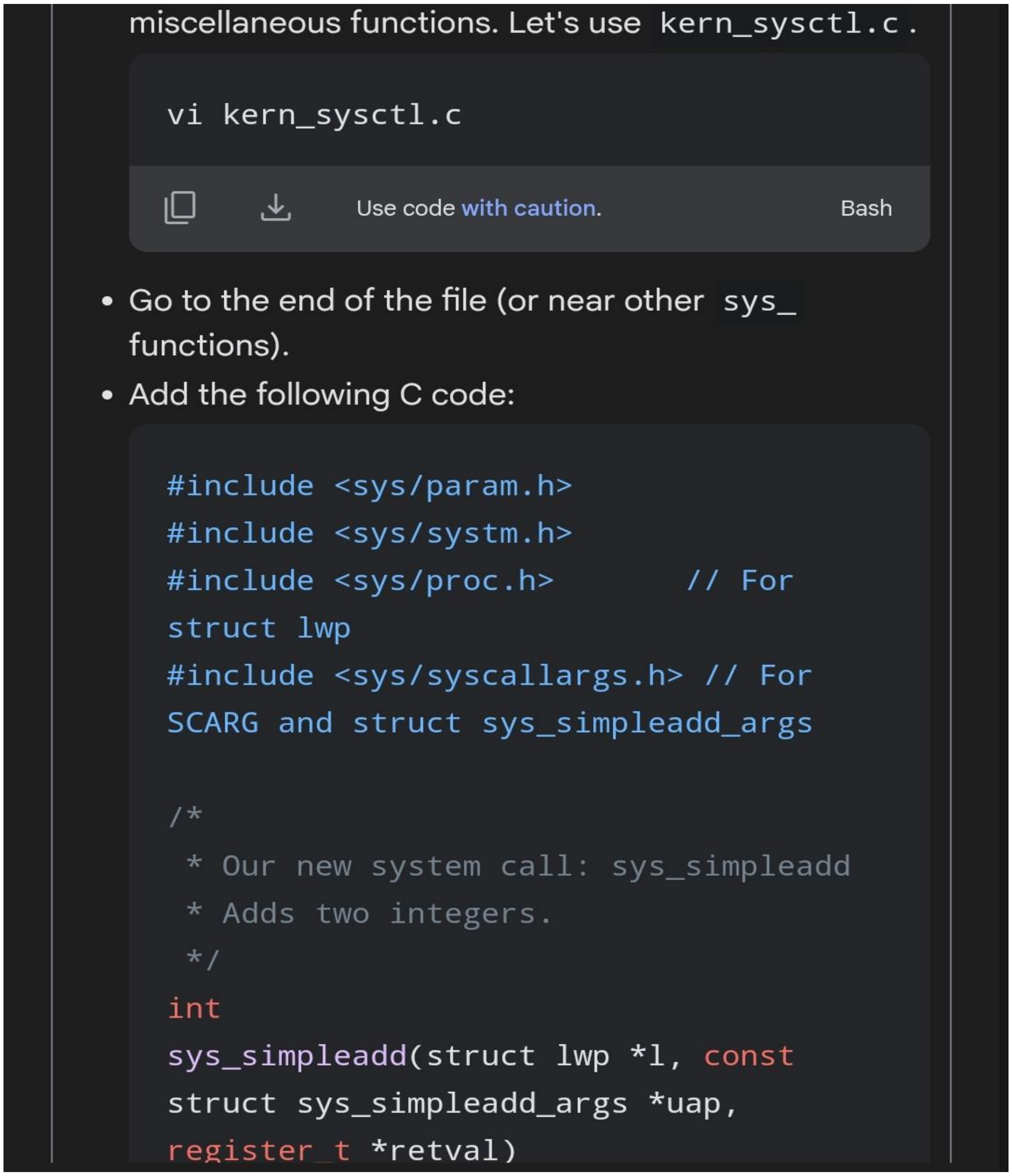
```
510    STD    { int sys_simpleadd(int a,
int b); }
```

Use code with caution.

This defines syscall number `510` as `sys_simpleadd` which takes two integers `a` and `b` and returns an `int`.

Screenshot Placeholder 3: `vi syscalls.master` showing the new line added.

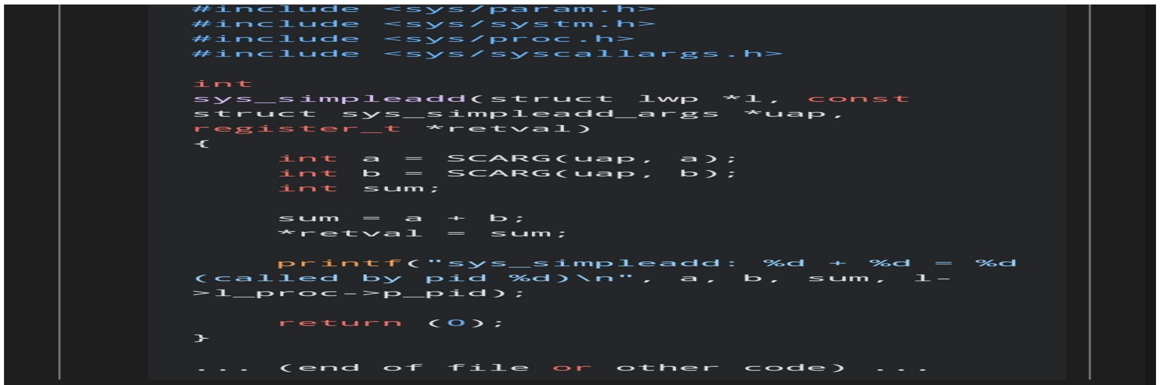
```
(Inside vi, scrolled to the bottom or
an appropriate section)
...
508    STD    { int sys_pipe2(int fd[2],
int flags); }
509    STD    { int sys_dup3(int from,
int to, int flags); }
510    STD    { int sys_simpleadd(int a,
int b); }    <-- OUR NEW SYSCALL
...
```


miscellaneous functions. Let's use `kern_sysctl.c`.

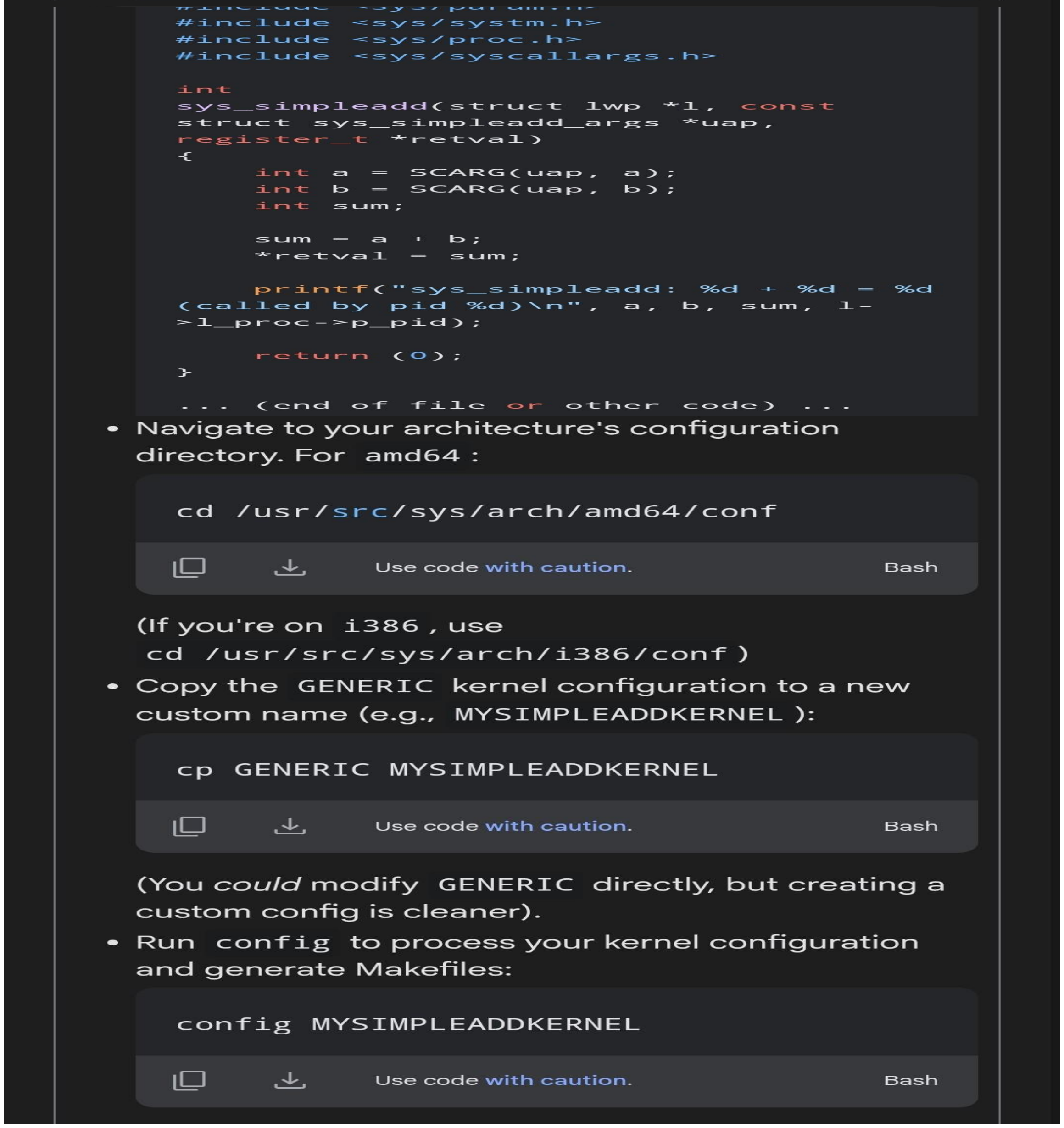
```
vi kern_sysctl.c
```

- Go to the end of the file (or near other `sys_` functions).
- Add the following C code:

```
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/proc.h>       // For
struct lwp
#include <sys/syscallargs.h> // For
SCARG and struct sys_simpleadd_args

/*
 * Our new system call: sys_simpleadd
 * Adds two integers.
 */
int
sys_simpleadd(struct lwp *l, const
struct sys_simpleadd_args *uap,
register_t *retval)
```



```
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/proc.h>
#include <sys/syscallargs.h>

int
sys_simpleadd(struct lwp *l, const
struct sys_simpleadd_args *uap,
register_t *retval)
{
        int a = SCARG(uap, a);
        int b = SCARG(uap, b);
        int sum;

        sum = a + b;
        *retval = sum;

        printf("sys_simpleadd: %d + %d = %d
(called by pid %d)\n", a, b, sum, l-
>l_proc->p_pid);

        return (0);
}

... (end of file or other code) ...
```



```
#include <sys/systm.h>
#include <sys/proc.h>
#include <sys/syscallargs.h>

int
sys_simpleadd(struct lwp *l, const
struct sys_simpleadd_args *uap,
register_t *retval)
{
        int a = SCARG(uap, a);
        int b = SCARG(uap, b);
        int sum;

        sum = a + b;
        *retval = sum;

        printf("sys_simpleadd: %d + %d = %d
(called by pid %d)\n", a, b, sum, l-
>l_proc->p_pid);

        return (0);
}

... (end of file or other code) ...
```

- Navigate to your architecture's configuration directory. For `amd64`:

```bash
cd /usr/src/sys/arch/amd64/conf
```

(If you're on `i386`, use `cd /usr/src/sys/arch/i386/conf`)

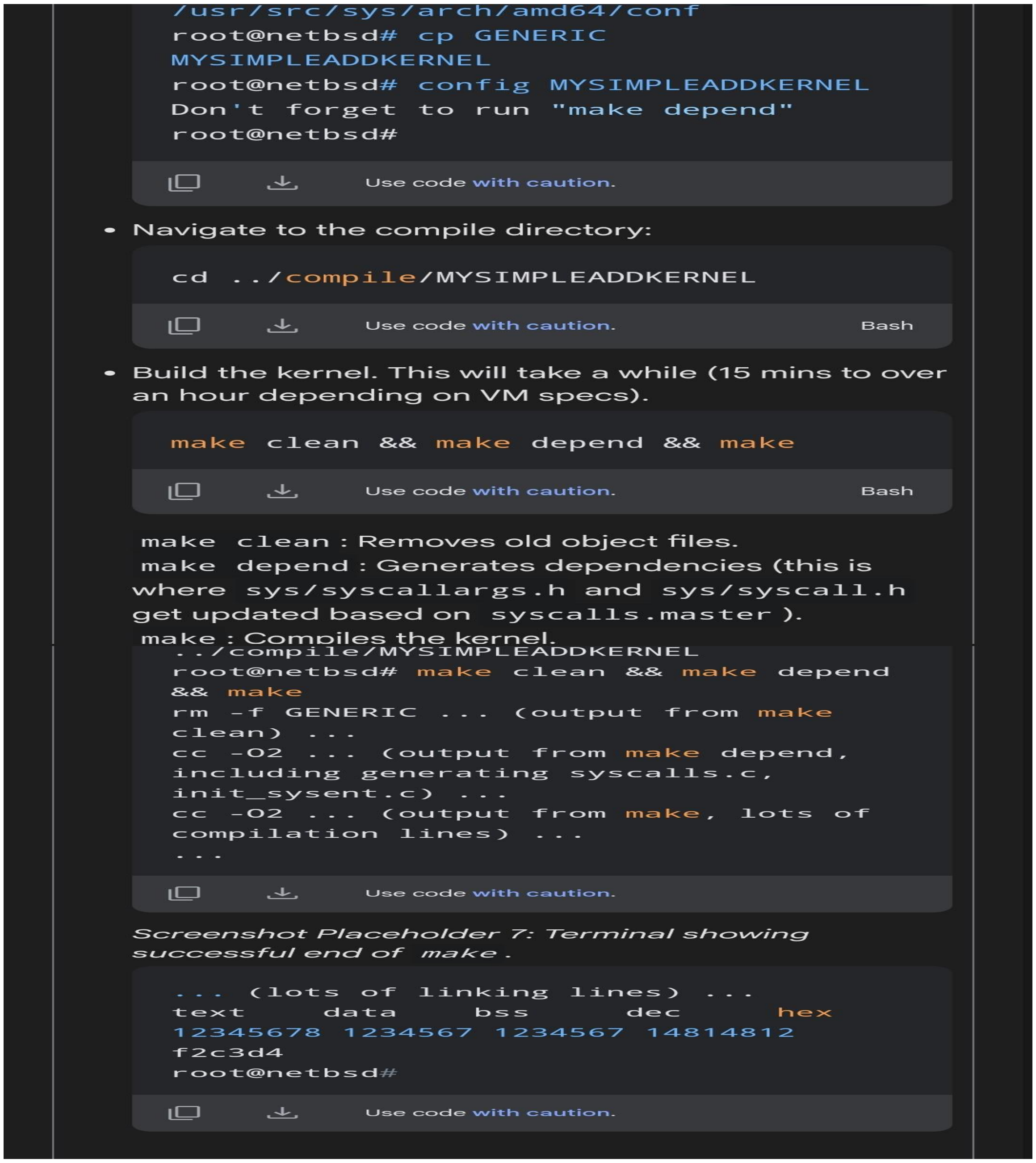
- Copy the `GENERIC` kernel configuration to a new custom name (e.g., `MYSIMPLEADDKERNEL`):

```bash
cp GENERIC MYSIMPLEADDKERNEL
```

(You *could* modify `GENERIC` directly, but creating a custom config is cleaner).

- Run `config` to process your kernel configuration and generate Makefiles:

```bash
config MYSIMPLEADDKERNEL
```



```
/usr/src/sys/arch/amd64/conf
root@netbsd# cp GENERIC
MYSIMPLEADDKERNEL
root@netbsd# config MYSIMPLEADDKERNEL
Don't forget to run "make depend"
root@netbsd#
```

Use code with caution.

- Navigate to the compile directory:

```bash
cd ../compile/MYSIMPLEADDKERNEL
```

Use code with caution. Bash

- Build the kernel. This will take a while (15 mins to over an hour depending on VM specs).

```bash
make clean && make depend && make
```

Use code with caution. Bash

`make clean` : Removes old object files.
`make depend` : Generates dependencies (this is where `sys/syscallargs.h` and `sys/syscall.h` get updated based on `syscalls.master`).
`make` : Compiles the kernel.

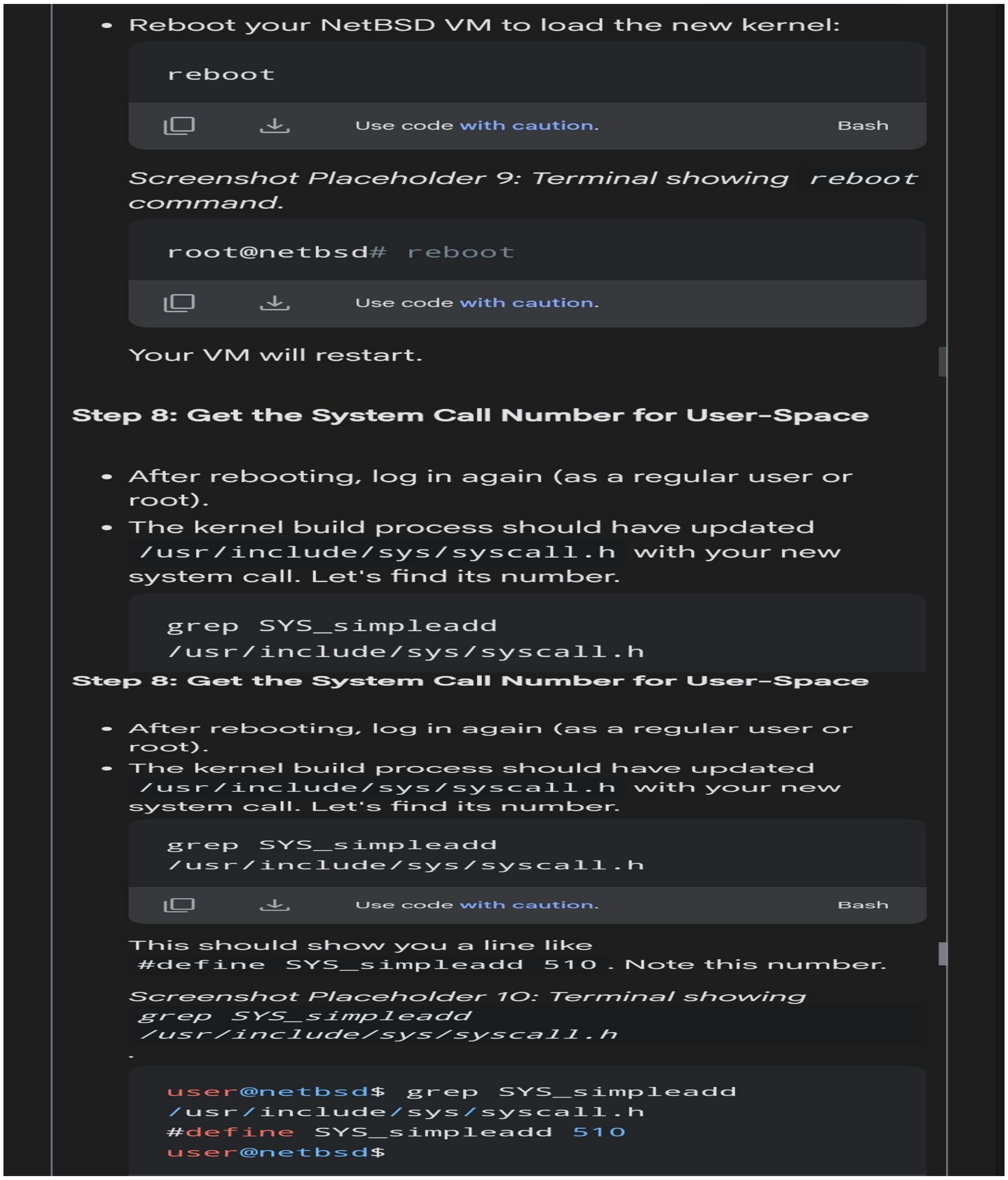
```
../compile/MYSIMPLEADDKERNEL
root@netbsd# make clean && make depend
&& make
rm -f GENERIC ... (output from make
clean) ...
cc -O2 ... (output from make depend,
including generating syscalls.c,
init_sysent.c) ...
cc -O2 ... (output from make, lots of
compilation lines) ...
...
```

Use code with caution.

Screenshot Placeholder 7: Terminal showing successful end of `make` .

```
... (lots of linking lines) ...
text      data      bss      dec      hex
12345678 1234567 1234567 14814812
f2c3d4
root@netbsd#
```

Use code with caution.



- Reboot your NetBSD VM to load the new kernel:

```bash
reboot
```

Use code with caution. Bash

Screenshot Placeholder 9: Terminal showing `reboot` command.

```
root@netbsd# reboot
```

Use code with caution.

Your VM will restart.

Step 8: Get the System Call Number for User-Space

- After rebooting, log in again (as a regular user or root).
- The kernel build process should have updated `/usr/include/sys/syscall.h` with your new system call. Let's find its number.

```
grep SYS_simpleadd
/usr/include/sys/syscall.h
```

Step 8: Get the System Call Number for User-Space

- After rebooting, log in again (as a regular user or root).
- The kernel build process should have updated `/usr/include/sys/syscall.h` with your new system call. Let's find its number.

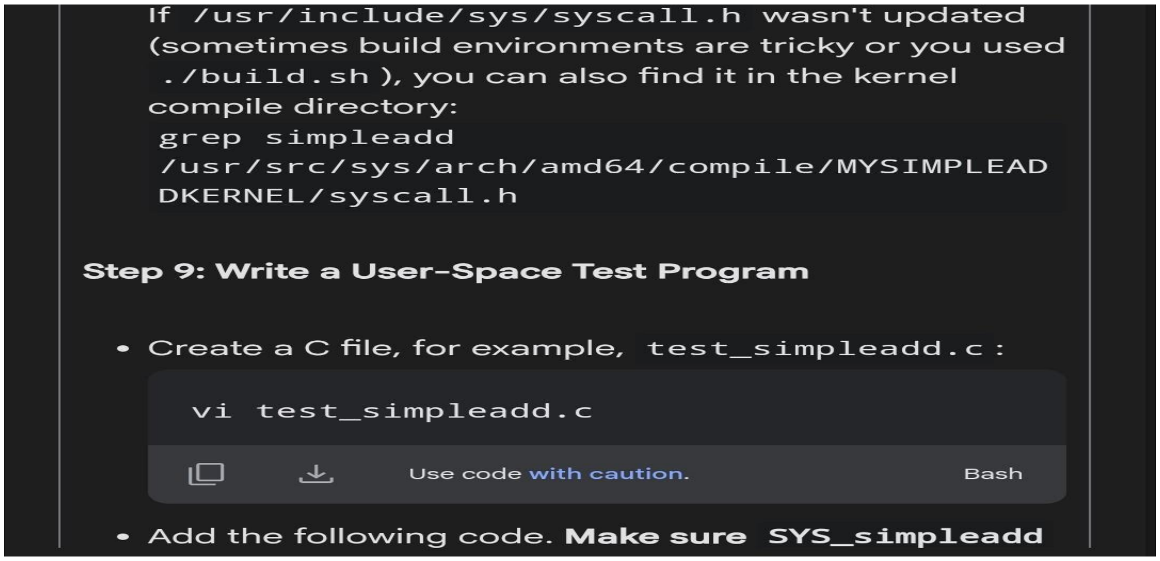
```bash
grep SYS_simpleadd
/usr/include/sys/syscall.h
```

Use code with caution. Bash

This should show you a line like `#define SYS_simpleadd 510`. Note this number.

Screenshot Placeholder 10: Terminal showing `grep SYS_simpleadd /usr/include/sys/syscall.h`
.

```
user@netbsd$ grep SYS_simpleadd
/usr/include/sys/syscall.h
#define SYS_simpleadd 510
user@netbsd$
```



If `/usr/include/sys/syscall.h` wasn't updated (sometimes build environments are tricky or you used `./build.sh`), you can also find it in the kernel compile directory:

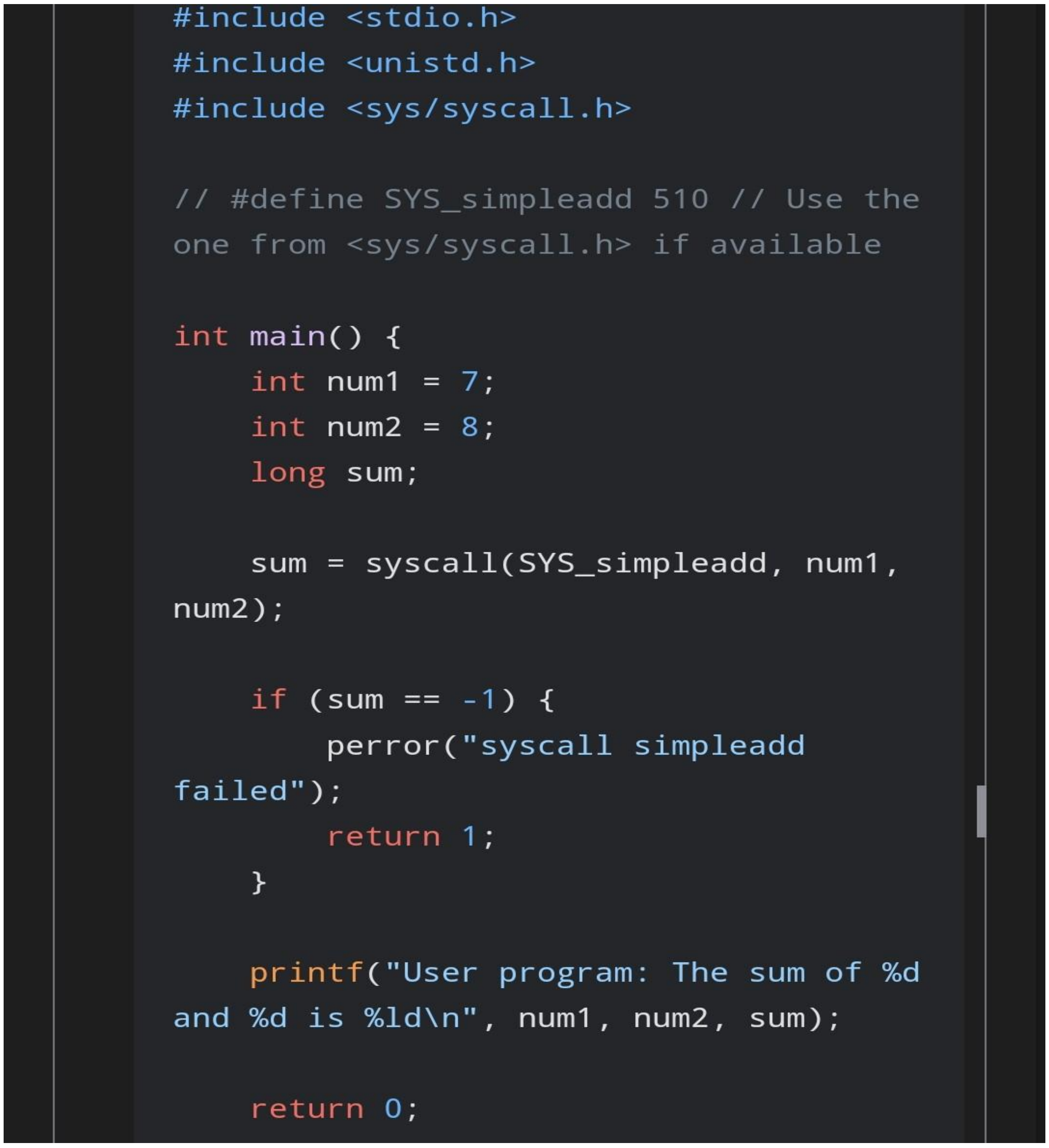
```
grep simpleadd
/usr/src/sys/arch/amd64/compile/MYSIMPLEAD
DKERNEL/syscall.h
```

Step 9: Write a User-Space Test Program

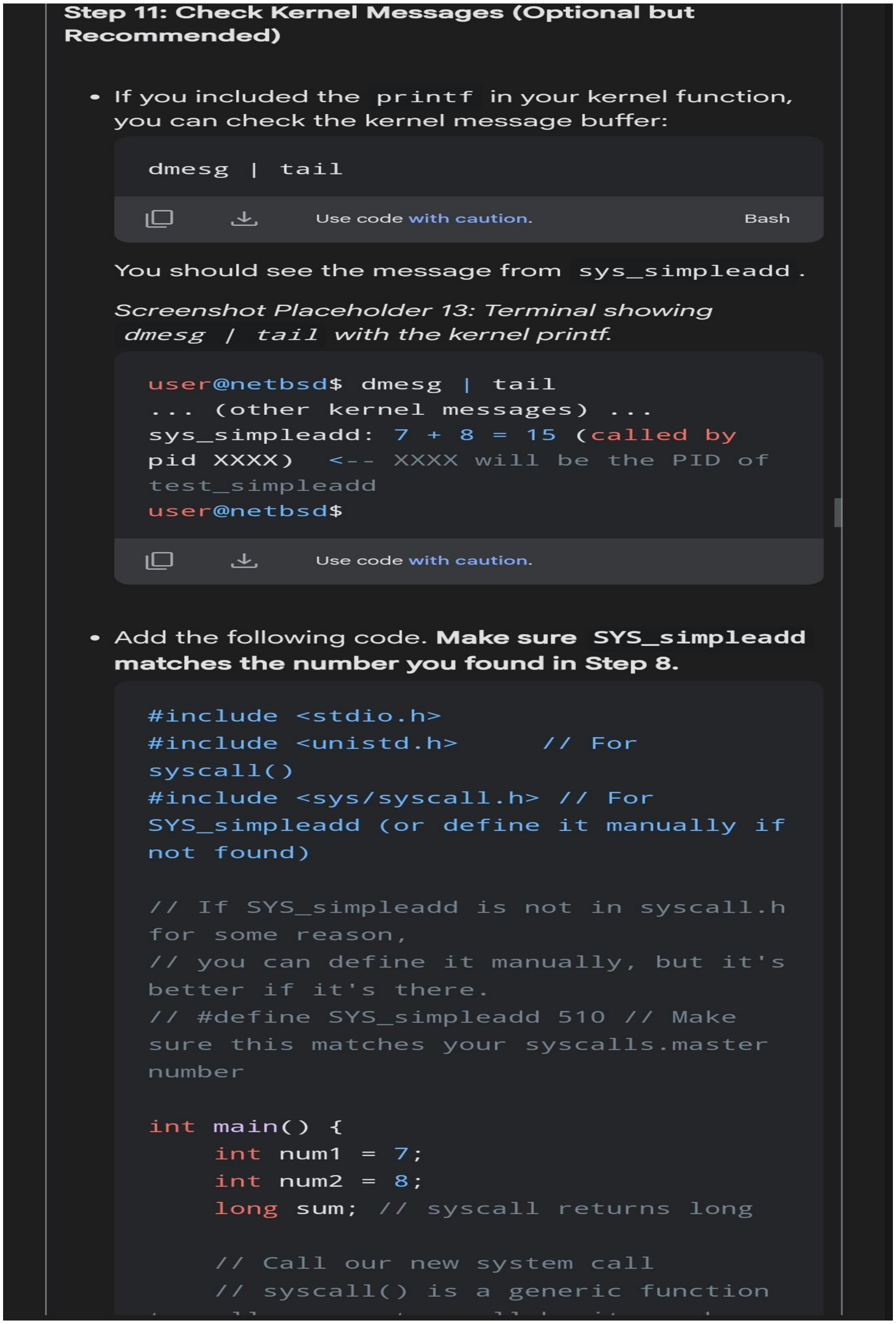
- Create a C file, for example, `test_simpleadd.c`:

```
vi test_simpleadd.c
```

- Add the following code. **Make sure `SYS_simpleadd`**



```c
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>

// #define SYS_simpleadd 510 // Use the
one from <sys/syscall.h> if available

int main() {
    int num1 = 7;
    int num2 = 8;
    long sum;

    sum = syscall(SYS_simpleadd, num1,
num2);

    if (sum == -1) {
        perror("syscall simpleadd
failed");
        return 1;
    }

    printf("User program: The sum of %d
and %d is %ld\n", num1, num2, sum);

    return 0;
```



Step 11: Check Kernel Messages (Optional but Recommended)

- If you included the `printf` in your kernel function, you can check the kernel message buffer:

  ```bash
  dmesg | tail
  ```

  Use code with caution. Bash

  You should see the message from `sys_simpleadd`.

  Screenshot Placeholder 13: Terminal showing `dmesg | tail` with the kernel printf.

  ```
  user@netbsd$ dmesg | tail
  ... (other kernel messages) ...
  sys_simpleadd: 7 + 8 = 15 (called by
  pid XXXX)  <-- XXXX will be the PID of
  test_simpleadd
  user@netbsd$
  ```

  Use code with caution.

- Add the following code. **Make sure `SYS_simpleadd` matches the number you found in Step 8.**

  ```c
  #include <stdio.h>
  #include <unistd.h>      // For
  syscall()
  #include <sys/syscall.h> // For
  SYS_simpleadd (or define it manually if
  not found)

  // If SYS_simpleadd is not in syscall.h
  for some reason,
  // you can define it manually, but it's
  better if it's there.
  // #define SYS_simpleadd 510 // Make
  sure this matches your syscalls.master
  number

  int main() {
      int num1 = 7;
      int num2 = 8;
      long sum; // syscall returns long

      // Call our new system call
      // syscall() is a generic function
  ```