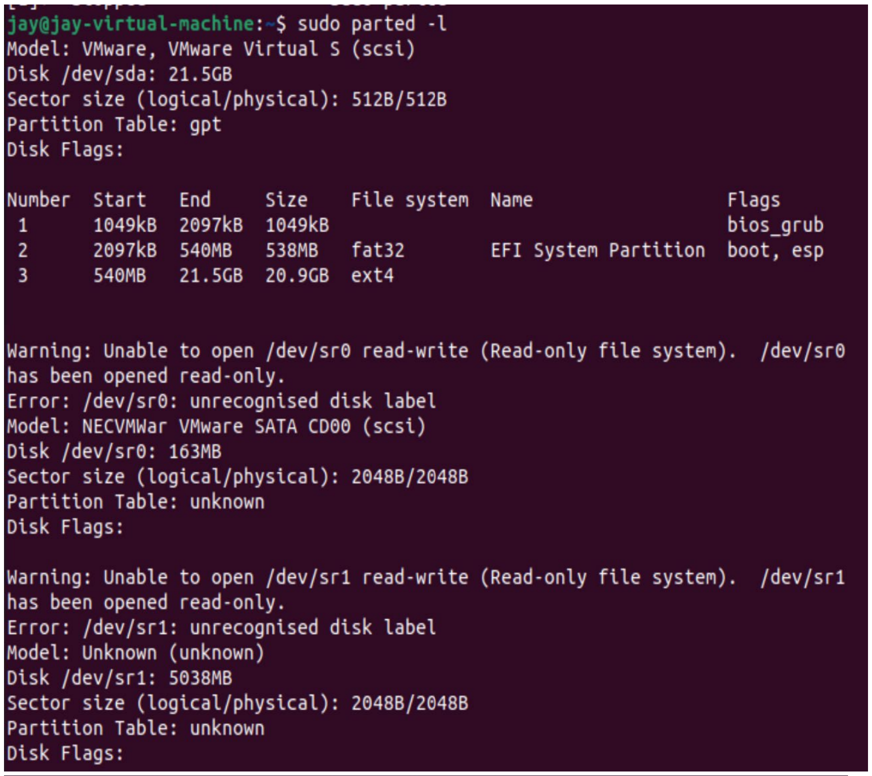
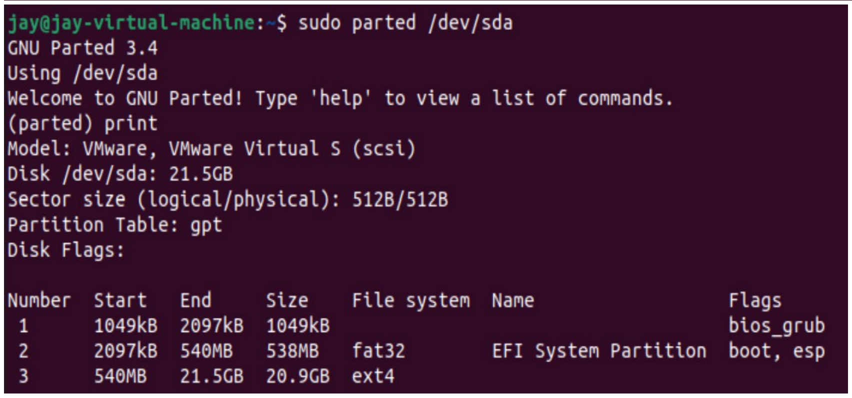
Part 1: File Management on Linux OS



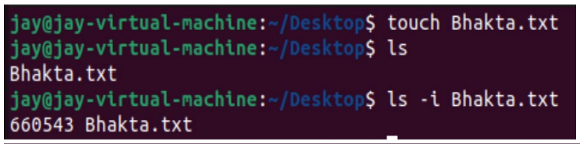
```
jay@jay-virtual-machine:~$ sudo parted -l
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sda: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name                  Flags
 1      1049kB  2097kB  1049kB                                     bios_grub
 2      2097kB  540MB   538MB   fat32        EFI System Partition  boot, esp
 3      540MB   21.5GB  20.9GB  ext4


Warning: Unable to open /dev/sr0 read-write (Read-only file system).  /dev/sr0
has been opened read-only.
Error: /dev/sr0: unrecognised disk label
Model: NECVMWar VMware SATA CD00 (scsi)
Disk /dev/sr0: 163MB
Sector size (logical/physical): 2048B/2048B
Partition Table: unknown
Disk Flags:

Warning: Unable to open /dev/sr1 read-write (Read-only file system).  /dev/sr1
has been opened read-only.
Error: /dev/sr1: unrecognised disk label
Model: Unknown (unknown)
Disk /dev/sr1: 5038MB
Sector size (logical/physical): 2048B/2048B
Partition Table: unknown
Disk Flags:
```



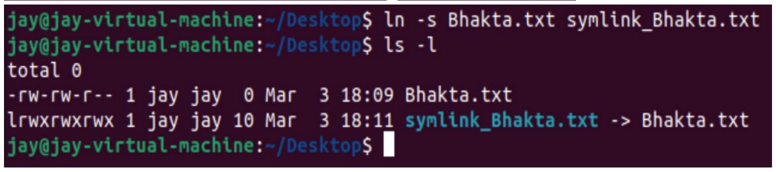
```
jay@jay-virtual-machine:~$ sudo parted /dev/sda
GNU Parted 3.4
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sda: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name                  Flags
 1      1049kB  2097kB  1049kB                                     bios_grub
 2      2097kB  540MB   538MB   fat32        EFI System Partition  boot, esp
 3      540MB   21.5GB  20.9GB  ext4
```

- **Is there any free space?** Yes, I think there is free space because “Disk /dev/sda: 21.5GB”.
- **What are your File System Types? : Note the file system types (like ext4, ntfs, or fat32) as they determine compatibility with different operating systems.** My file system types include fat32 and ext4.
- **Are you partitioned using MBR or GPT? What is the difference?** I am partitioned using gpt. MBR works with disks up to 2 TB in size and only supports up to four primary partitions. While, GPT is a newer standard in which drives can be much larger and allows for an unlimited number of partitions, with size limits dependent on the OS and its file system. Also, on an MBR disk, the partitioning and boot data is stored in one place. In contrast, GPT stores multiple copies of the data across the disk, making it more robust.

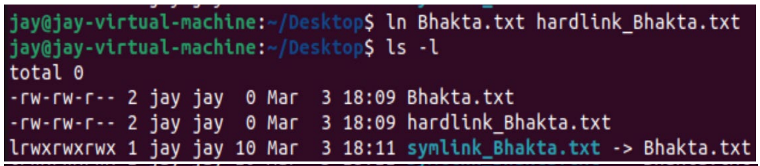
- **What are your partitions being used for?: Ensure that each partition has adequate space for its intended use. The first partition is the install partition and is used as the root of the filesystem tree. The other two partitions are used for the FAT32 (organizes storage) and ext4 (extends storage limits) file systems.**



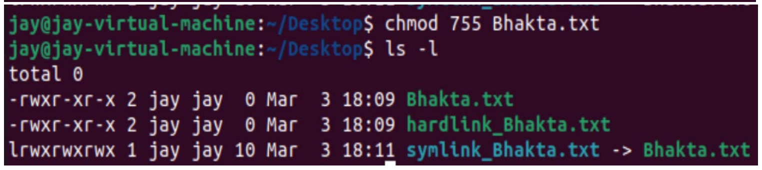
```
jay@jay-virtual-machine:~/Desktop$ touch Bhakta.txt
jay@jay-virtual-machine:~/Desktop$ ls
Bhakta.txt
jay@jay-virtual-machine:~/Desktop$ ls -i Bhakta.txt
660543 Bhakta.txt
```



```
jay@jay-virtual-machine:~/Desktop$ ln -s Bhakta.txt symlink_Bhakta.txt
jay@jay-virtual-machine:~/Desktop$ ls -l
total 0
-rw-rw-r-- 1 jay jay  0 Mar  3 18:09 Bhakta.txt
lrwxrwxrwx 1 jay jay 10 Mar  3 18:11 symlink_Bhakta.txt -> Bhakta.txt
jay@jay-virtual-machine:~/Desktop$
```



```
jay@jay-virtual-machine:~/Desktop$ ln Bhakta.txt hardlink_Bhakta.txt
jay@jay-virtual-machine:~/Desktop$ ls -l
total 0
-rw-rw-r-- 2 jay jay  0 Mar  3 18:09 Bhakta.txt
-rw-rw-r-- 2 jay jay  0 Mar  3 18:09 hardlink_Bhakta.txt
lrwxrwxrwx 1 jay jay 10 Mar  3 18:11 symlink_Bhakta.txt -> Bhakta.txt
```



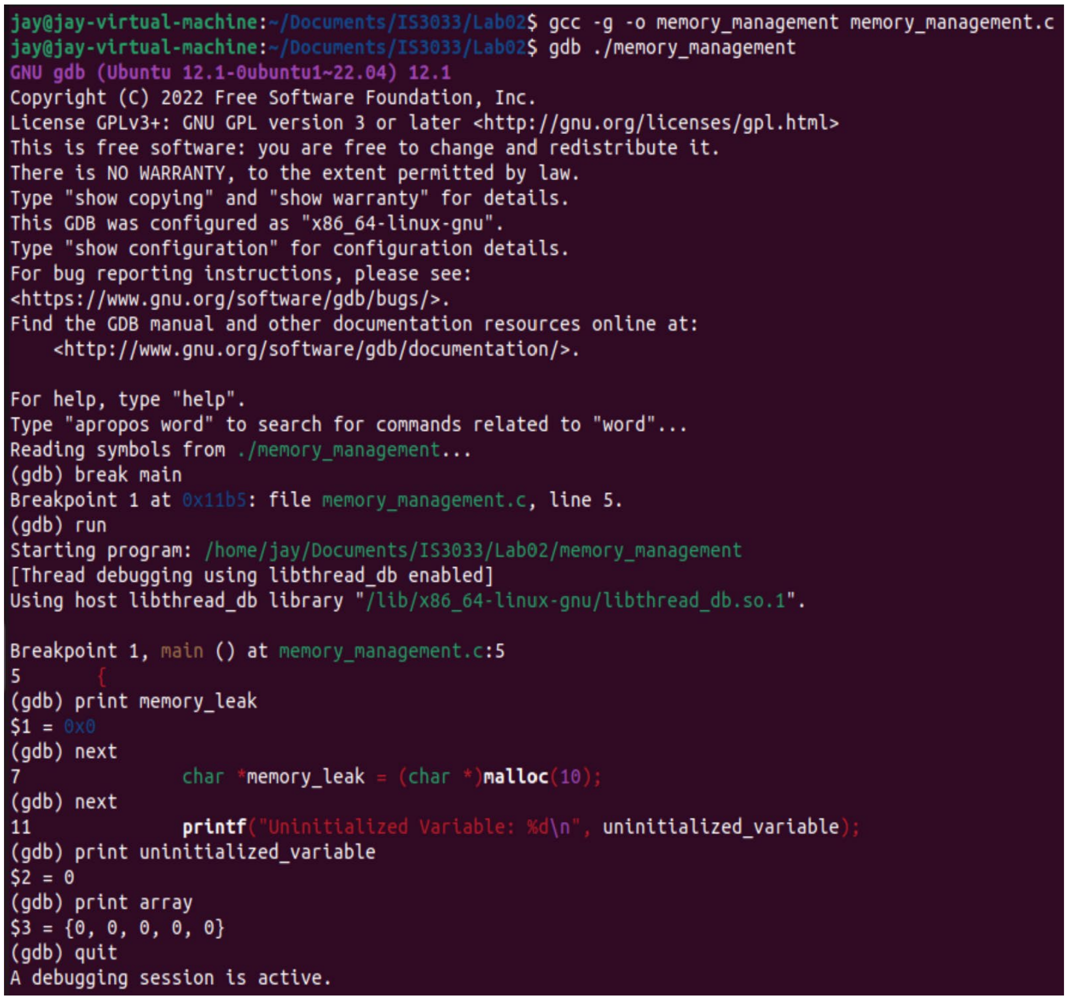
```
jay@jay-virtual-machine:~/Desktop$ chmod 755 Bhakta.txt
jay@jay-virtual-machine:~/Desktop$ ls -l
total 0
-rwxr-xr-x 2 jay jay  0 Mar  3 18:09 Bhakta.txt
-rwxr-xr-x 2 jay jay  0 Mar  3 18:09 hardlink_Bhakta.txt
lrwxrwxrwx 1 jay jay 10 Mar  3 18:11 symlink_Bhakta.txt -> Bhakta.txt
```

- **What is your inode number and what is it for?** The inode number is 660543 and it is used to identify the file I created within the Linux database.
- **What is your symlink and what can you use it for?** The symlink I created was "symlink_Bhakta.txt" and it can be used as a reference to the "Bhakta.txt" file.
- **What is your hardlink and how is it different from your symlink?** The hardlink I created was "hardlink_Bhakta.txt". While the symlink was used as a reference, the hardlink directly points to the original file, Bhakta.txt.
- **When you use chmod, what permissions is a file set at 755? What about 644? What are common files with these permissions (755 and 644) and why?** A file set at 755 gives the owner read, write, and execute permissions and group members + all others read and execute permissions only. A file set at 644 gives the owner read and write permissions and group members + all others read permissions only This is because with 755 the hundreds place digit being 7 represents the owner having read + write + execute permissions (4-read, 2-write; 1- execute, 0-no permissions), the tens place digit being 5 represents group members having read + execute permissions, and the ones place digit being 5 represents all others having read + execute permissions also. With 644 the hundreds place digit represents the owner having read + write permissions, the tens and ones place digit represents group members and all others having only read permissions. . Therefore, owners with these permissions (755 and 644) both have read and write

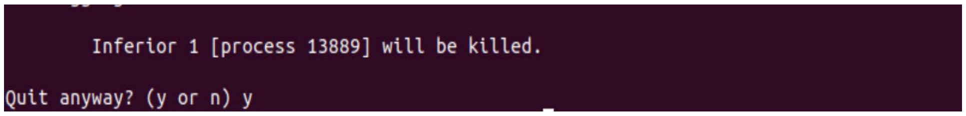
permissions in common; while, the group members + all others only have read permissions in common.

- **How does changing the permissions of the original file example.txt affect the permissions of your symlink and hardlink?** Changing the permissions of the original file caused the hardlink to have the changed permissions, while the symlink had the same permissions of (lrwxrwxrwx) as before.

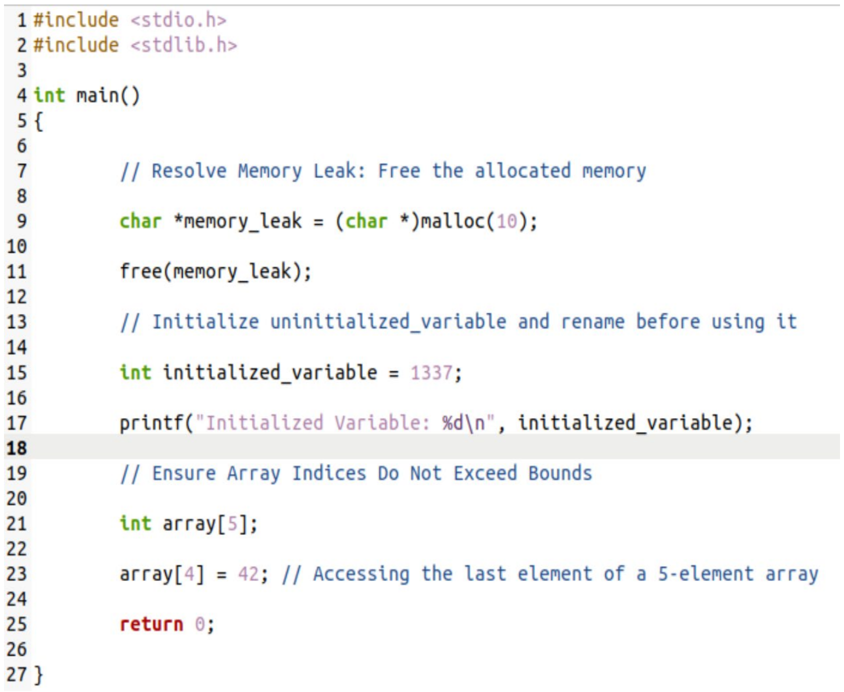
Part 2: Memory Management on Linux OS



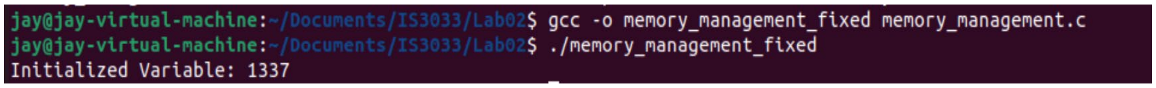
```
jay@jay-virtual-machine:~/Documents/IS3033/Lab02$ gcc -g -o memory_management memory_management.c
jay@jay-virtual-machine:~/Documents/IS3033/Lab02$ gdb ./memory_management
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./memory_management...
(gdb) break main
Breakpoint 1 at 0x11b5: file memory_management.c, line 5.
(gdb) run
Starting program: /home/jay/Documents/IS3033/Lab02/memory_management
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at memory_management.c:5
5       {
(gdb) print memory_leak
$1 = 0x0
(gdb) next
7               char *memory_leak = (char *)malloc(10);
(gdb) next
11              printf("Uninitialized Variable: %d\n", uninitialized_variable);
(gdb) print uninitialized_variable
$2 = 0
(gdb) print array
$3 = {0, 0, 0, 0, 0}
(gdb) quit
A debugging session is active.
```



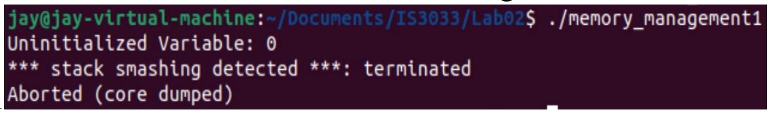
```
        Inferior 1 [process 13889] will be killed.

Quit anyway? (y or n) y
```



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

int main()
{

        // Resolve Memory Leak: Free the allocated memory

        char *memory_leak = (char *)malloc(10);

        free(memory_leak);

        // Initialize uninitialized_variable and rename before using it

        int initialized_variable = 1337;

        printf("Initialized Variable: %d\n", initialized_variable);

        // Ensure Array Indices Do Not Exceed Bounds

        int array[5];

        array[4] = 42; // Accessing the last element of a 5-element array

        return 0;

}
```



```
jay@jay-virtual-machine:~/Documents/IS3033/Lab02$ gcc -o memory_management_fixed memory_management.c
jay@jay-virtual-machine:~/Documents/IS3033/Lab02$ ./memory_management_fixed
Initialized Variable: 1337
```

- **What is the result after compiling and running memory_management.c in Step 1?**
  The uninitialized variable is set to 0, stack smashing is detected, and there is core dump.
  111111111111

```
jay@jay-virtual-machine:~/Documents/IS3033/Lab02$ ./memory_management1
Uninitialized Variable: 0
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

- **Does memory_management.c have a memory leak and and why?\**
  Yes, it does have a memory leak because I allocated memory and did not give it back.
- **Why use GDB to step through and troubleshoot the memory leak?**
  Using GDB allows me to look closely at memory usage of programs.
- **What is the indicator that you have fixed the memory leak in Step 3?**
  I think the indicator was the command "free(memory_leak);".
- **Why does the Unitialized variable print out after fixing the memory leak?**
  I am confused on this question because it isn't supposed to be why the "Initialized Variable" printed out. But the initialized variable prints out because I assigned 1337 to that variable and printed it using the printf command.