

# CS 377 Project Report

By

Bhawna(180010009)

Partiksha(180010020)

Yashi Agarwal (180010030)

We have built an experimental OS (eXpOS by nitc) in 10 stages. Code files of all the assignment questions of all stages have been uploaded in the google classroom.

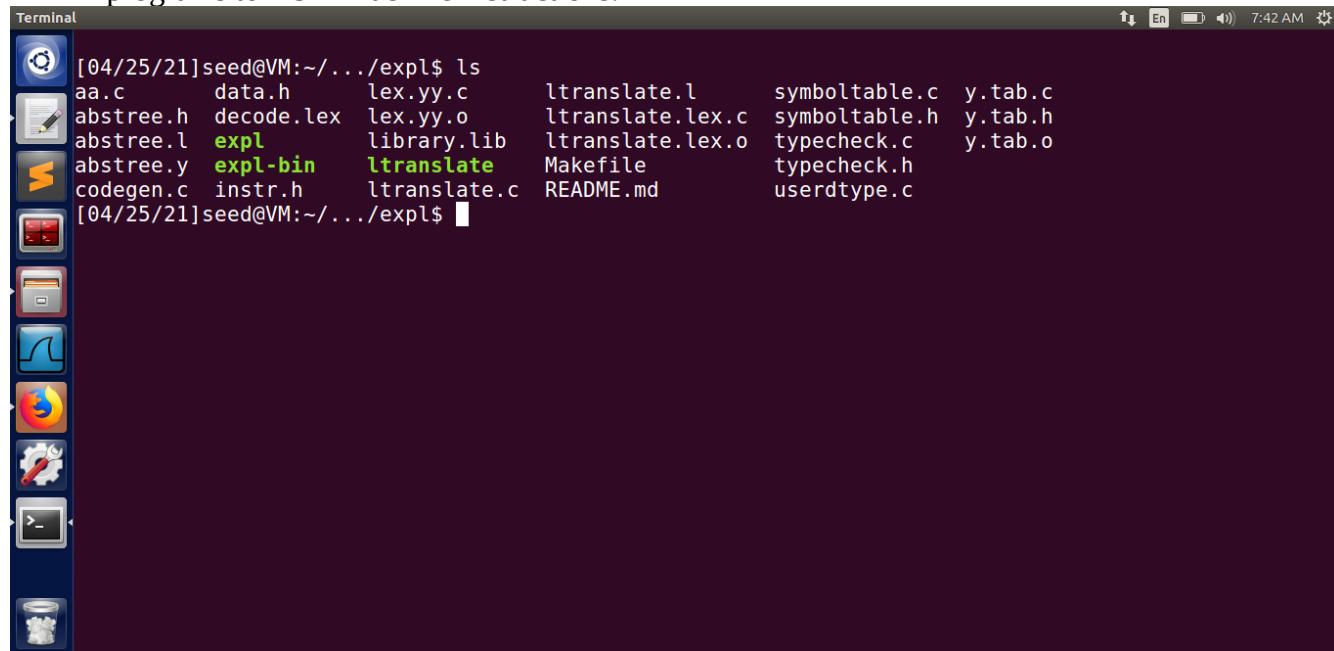
Code files are named in the below given format -  
fileName\_stage\_stageNumber.extension(.spl/.xsm)

This report contains screenshots and an explanation of the outputs of the assignment questions stage-vise.

## STAGE 1 : Set up the environment to build eXpOS

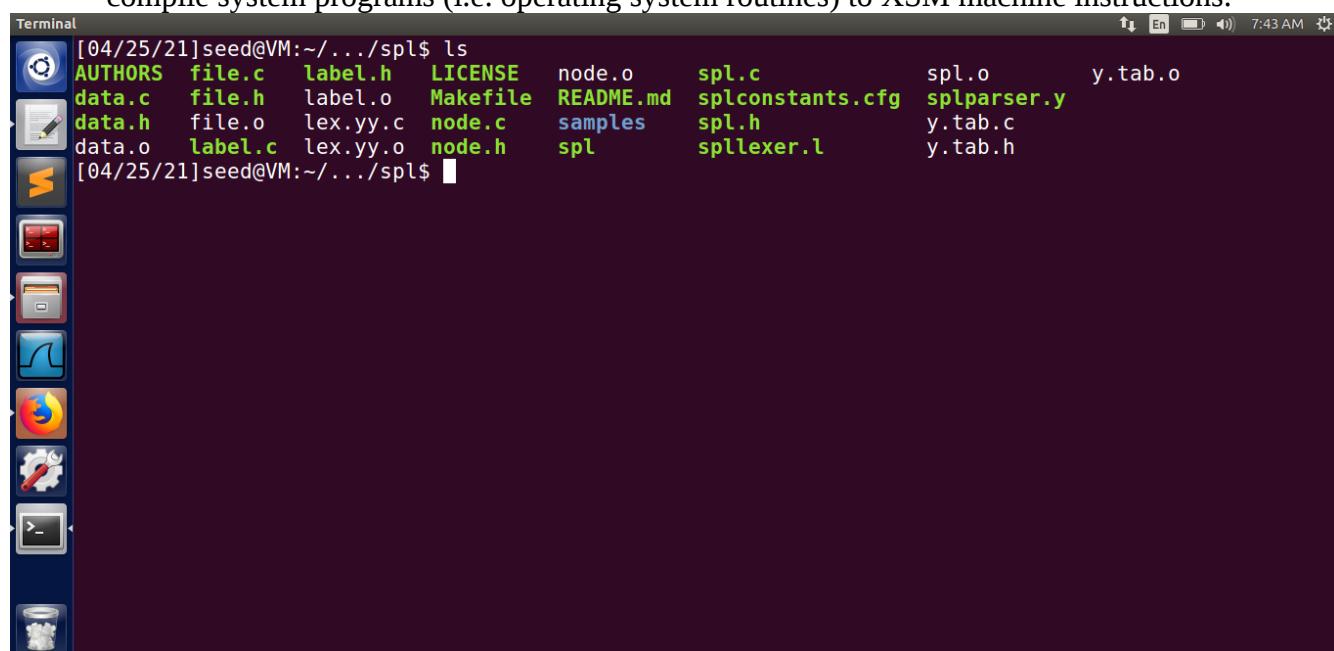
1. sudo apt-get install libreadline-dev flex bison make gcc wget curl
2. curl -sSf <https://raw.githubusercontent.com/eXp0SNitc/expo-bootstrap/main/download.sh> | sh
3. cd \$HOME/myexpos
4. make

- This directory contains the Expl (Experimental Language) compiler required to compile user programs to XSM machine instructions.



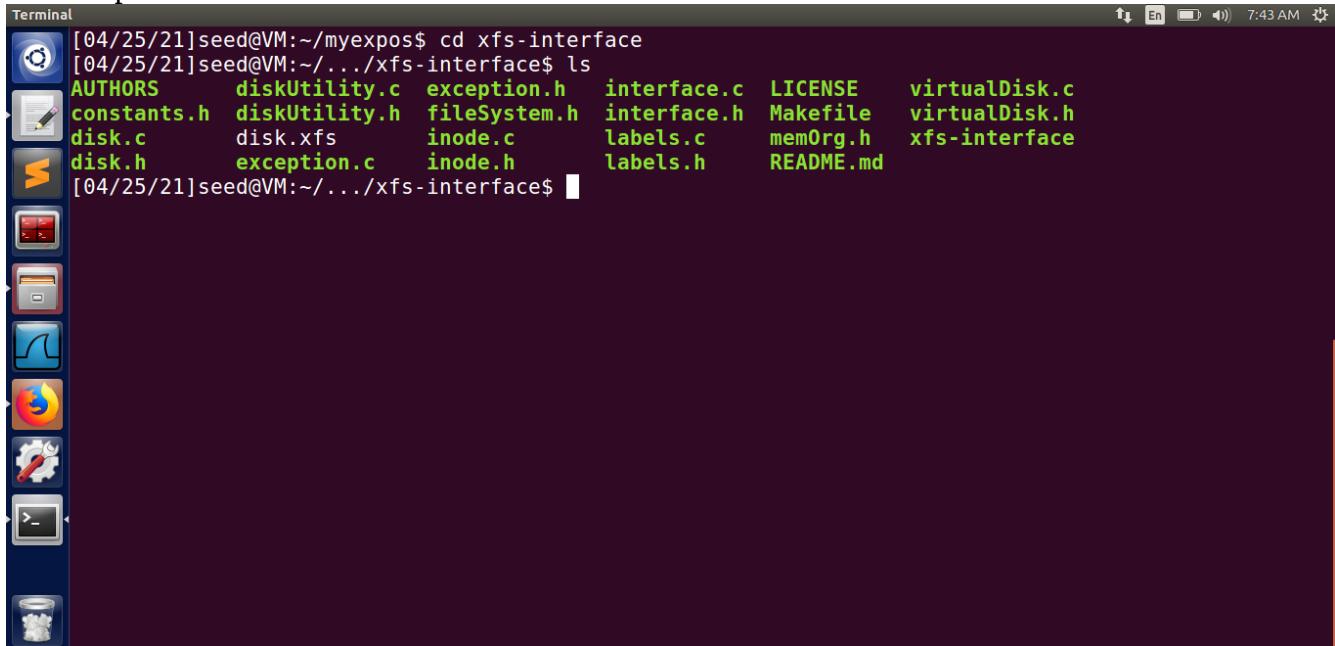
```
[04/25/21]seed@VM:~/.../expl$ ls
aa.c      data.h    lex.yy.c      ltranslate.l      symboltable.c  y.tab.c
abstree.h  decode.lex lex.yy.o     ltranslate.lex.c  symboltable.h  y.tab.h
abstree.l  expl      library.lib   ltranslate.lex.o  typecheck.c   y.tab.o
abstree.y  expl-bin  ltranslate   Makefile        typecheck.h
codegen.c  instr.h   ltranslate.c README.md       userdtype.c
```

- This directory contains the SPL (System Programmer's Language) Compiler required to compile system programs (i.e. operating system routines) to XSM machine instructions.



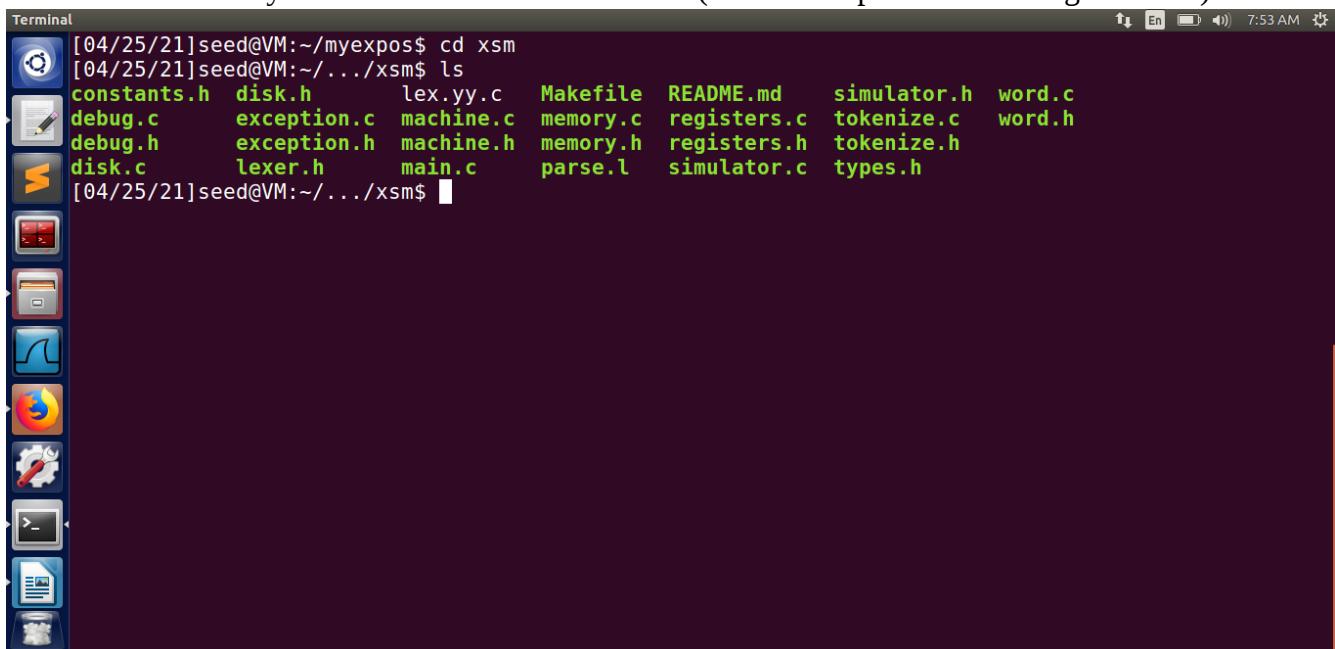
```
[04/25/21]seed@VM:~/.../spl$ ls
AUTHORS  file.c  label.h  LICENSE  node.o   spl.c      spl.o   y.tab.o
data.c   file.h  label.o  Makefile  README.md  splconstants.cfg  splparser.y
data.h   file.o  lex.yy.c  node.c   samples  spl.h      y.tab.c
data.o   label.c  lex.yy.o  node.h   spl      spllexer.l  y.tab.h
[04/25/21]seed@VM:~/.../spl$
```

- This directory contains an interface (XFS Interface or eXperimental File System Interface) through which files from your UNIX machine can be loaded into the File System of XSM. The interface can format the disk, dump the disk data structures, load/remove files, list files, transfer data and executable files between eXpFS filesystem and the host (UNIX) file system and copy specified blocks of the XFS disk to a UNIX file.



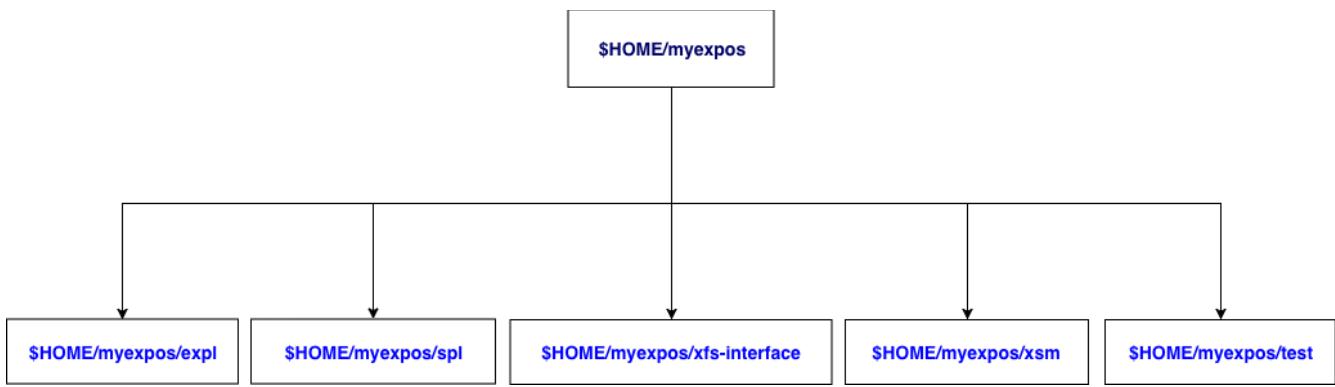
```
[04/25/21] seed@VM:~/myexpos$ cd xfs-interface
[04/25/21] seed@VM:~/.../xfs-interface$ ls
AUTHORS      diskUtility.c  exception.h    interface.c   LICENSE     virtualDisk.c
constants.h   diskUtility.h  fileSystem.h  interface.h  Makefile    virtualDisk.h
disk.c        disk.xfs      inode.c       labels.c    memOrg.h   xfs-interface
disk.h        exception.c  inode.h       labels.h    README.md
[04/25/21] seed@VM:~/.../xfs-interface$
```

- This directory contains the machine simulator (XSM or eXperimental String Machine).



```
[04/25/21] seed@VM:~/myexpos$ cd xsm
[04/25/21] seed@VM:~/.../xsm$ ls
constants.h  disk.h      lex.yy.c    Makefile  README.md  simulator.h  word.c
debug.c       exception.c machine.c   memory.c  registers.c tokenize.c  word.h
debug.h       exception.h machine.h   memory.h  registers.h tokenize.h
disk.c        lexer.h    main.c     parse.l   simulator.c types.h
[04/25/21] seed@VM:~/.../xsm$
```

We have done the setting of system correctly.  
Below image shows the set of directories.



## Stage 2 : Understanding the Filesystem

The eXpOS package that we had downloaded in the previous stage consists mainly of a machine simulator. The machine is called the **eXperimental String Machine (XSM)** and consists of a processor, memory and disk. Some support tools that help us to program the machine are also provided.

*In this stage, we will create a text file and load it to the XFS disk using xfs-interface.*

1. cd \$HOME/myexpos/xfs-interface  
./xfs-interface

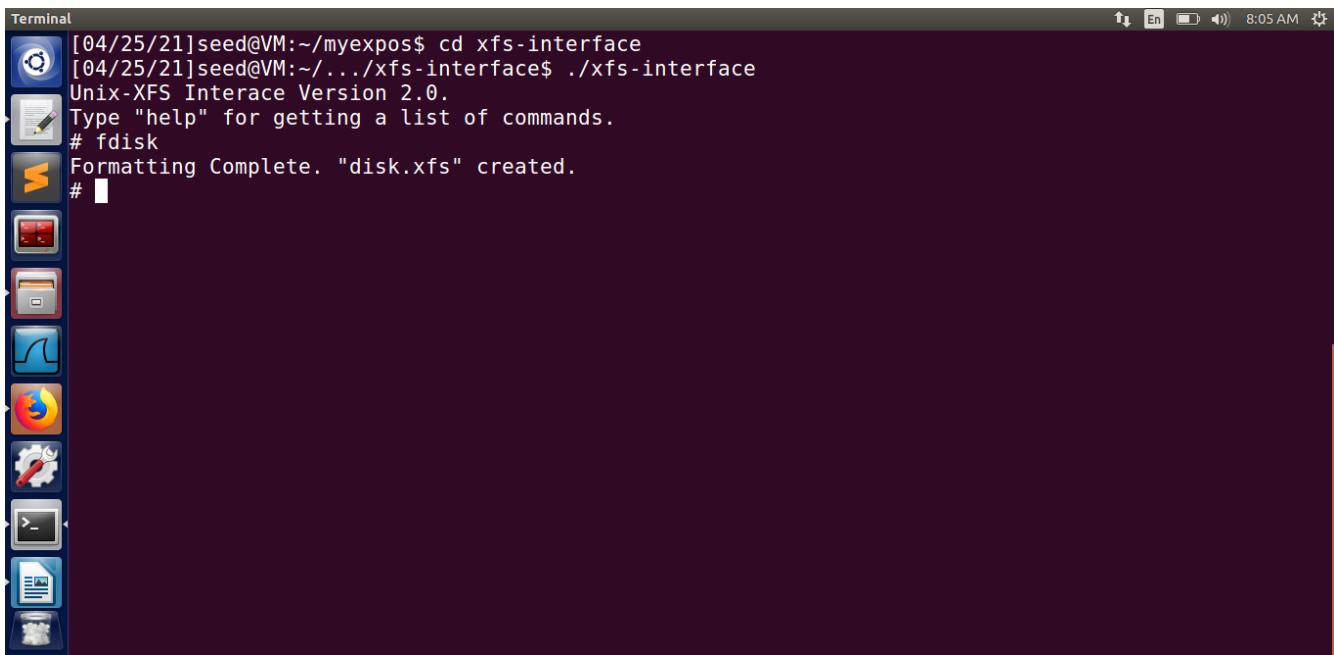


The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a toolbar with icons for file operations like copy, paste, and search. The title bar says "Terminal". The date and time "04/25/21" and "8:03 AM" are visible. The main area of the terminal displays the following text:

```
[04/25/21]seed@VM:~/myexpos$ cd $HOME/myexpos/xfs-interface
[04/25/21]seed@VM:~/.xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# help
fdisk
    Format the disk with XFS filesystem
run <pathname>
    Executes the set of xfs-interface commands sequentially
load --exec <pathname>
    Loads an executable file to XFS disk
load --data <pathname>
    Loads a data file to XFS disk
load --init <pathname>
    Loads INIT code to XFS disk
load --os <pathname>
    Loads OS startup code to XFS disk
load --idle <pathname>
    Loads Idle code to XFS disk
load --shell <pathname>
    Loads Shell code to XFS disk
load --library <pathname>
    Loads Library code to XFS disk
load --int=timer <pathname>
```

2. # fdisk  
# exit

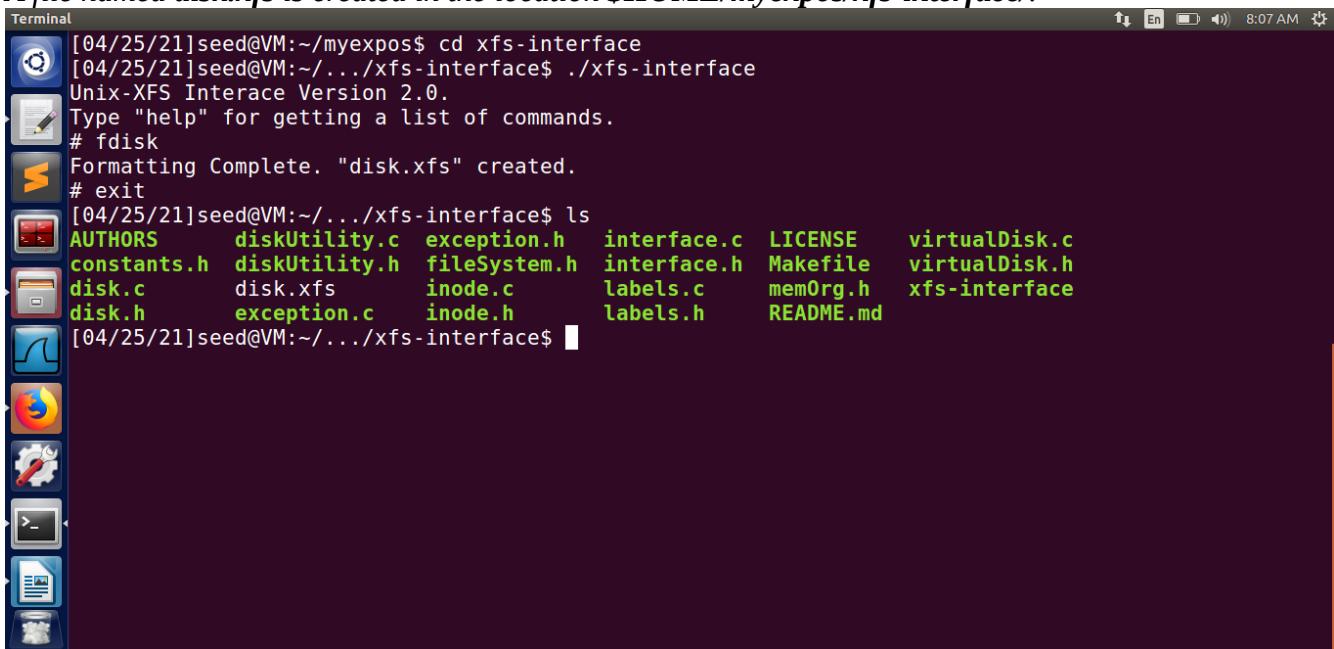
*The fdisk command converts the raw disk into the filesystem format recognised by the eXpOS operating system.*



A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window has a dark background and contains the following text:

```
[04/25/21]seed@VM:~/myexpos$ cd xfs-interface
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# fdisk
Formatting Complete. "disk.xfs" created.
#
```

*A file named **disk.xfs** is created in the location \$HOME/myexpos/xfs-interface/.*



A screenshot of a Linux desktop environment, likely Ubuntu, showing a terminal window. The terminal window has a dark background and contains the following text:

```
[04/25/21]seed@VM:~/myexpos$ cd xfs-interface
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# fdisk
Formatting Complete. "disk.xfs" created.
# exit
[04/25/21]seed@VM:~/.../xfs-interface$ ls
AUTHORS      diskUtility.c  exception.h   interface.c  LICENSE    virtualDisk.c
constants.h   diskUtility.h  fileSystem.h  interface.h  Makefile   virtualDisk.h
disk.c        disk.xfs     inode.c       labels.c    memOrg.h   xfs-interface
disk.h        exception.c  inode.h      labels.h    README.md
[04/25/21]seed@VM:~/.../xfs-interface$
```

3. The Disk Free List in XFS is a data structure which keeps track of used and unused blocks in the disk. An unused block is indicated by 0 and a used block is indicated by 1.

```
Terminal [04/25/21]seed@VM:~/.../xfs-interface$ ls
AUTHORS      diskUtility.c exception.h   interface.c LICENSE   virtualDisk.c
constants.h  diskUtility.h fileSystem.h  interface.h Makefile   virtualDisk.h
disk.c       disk.xfs     inode.c      labels.c  memOrg.h   xfs-interface
disk.h       exception.c inode.h      labels.h  README.md

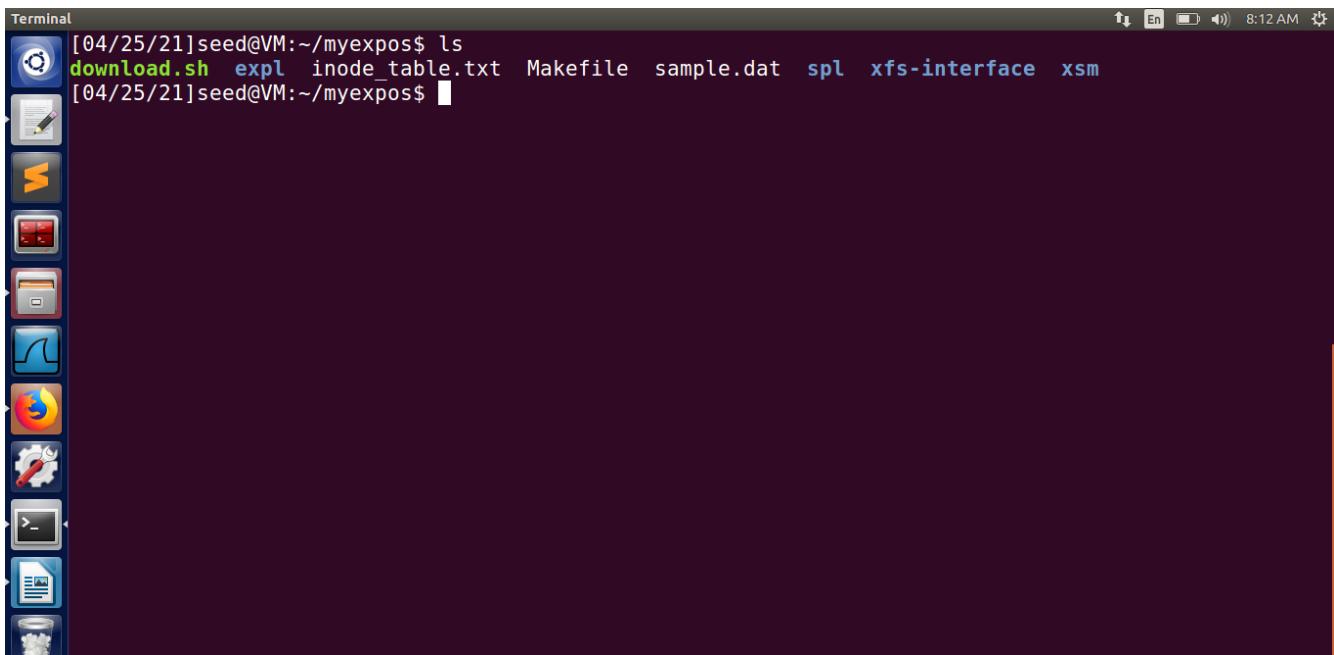
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# df
0          -      1
1          -      1
2          -      1
3          -      1
4          -      1
5          -      1
6          -      1
7          -      1
8          -      1
9          -      1
10         -      1
11         -      1
12         -      1
13         -      1
14         -      1
15         -      1
```

```
Terminal 491      -      0
492      -      0
493      -      0
494      -      0
495      -      0
496      -      0
497      -      0
498      -      0
499      -      0
500      -      0
501      -      0
502      -      0
503      -      0
504      -      0
505      -      0
506      -      0
507      -      0
508      -      0
509      -      0
510      -      0
511      -      0

[04/25/21]seed@VM:~/.../xfs-interface$ No of Free Blocks = 443
[04/25/21]seed@VM:~/.../xfs-interface$ Total no of Blocks = 512
#
```

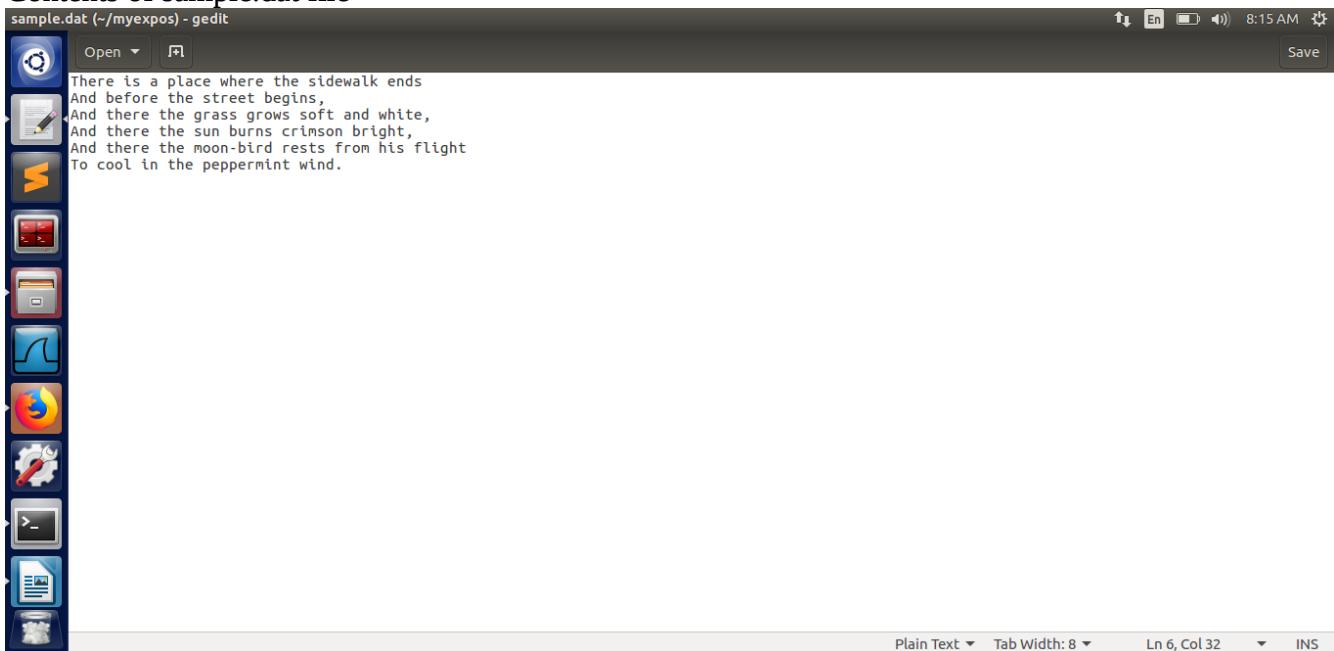
The first 69 blocks (blocks 0 to 68) are reserved for stroing various OS data structures and routines as well as Idle code, INIT program, etc. Hence the Disk Free List entries for these are marked as 1 (used) and the remaining entries for blocks 69 to 511 are 0.

4. Created a sample.dat file.



```
[04/25/21]seed@VM:~/myexpos$ ls
download.sh  expl  inode_table.txt  Makefile  sample.dat  spl  xfs-interface  xsm
[04/25/21]seed@VM:~/myexpos$
```

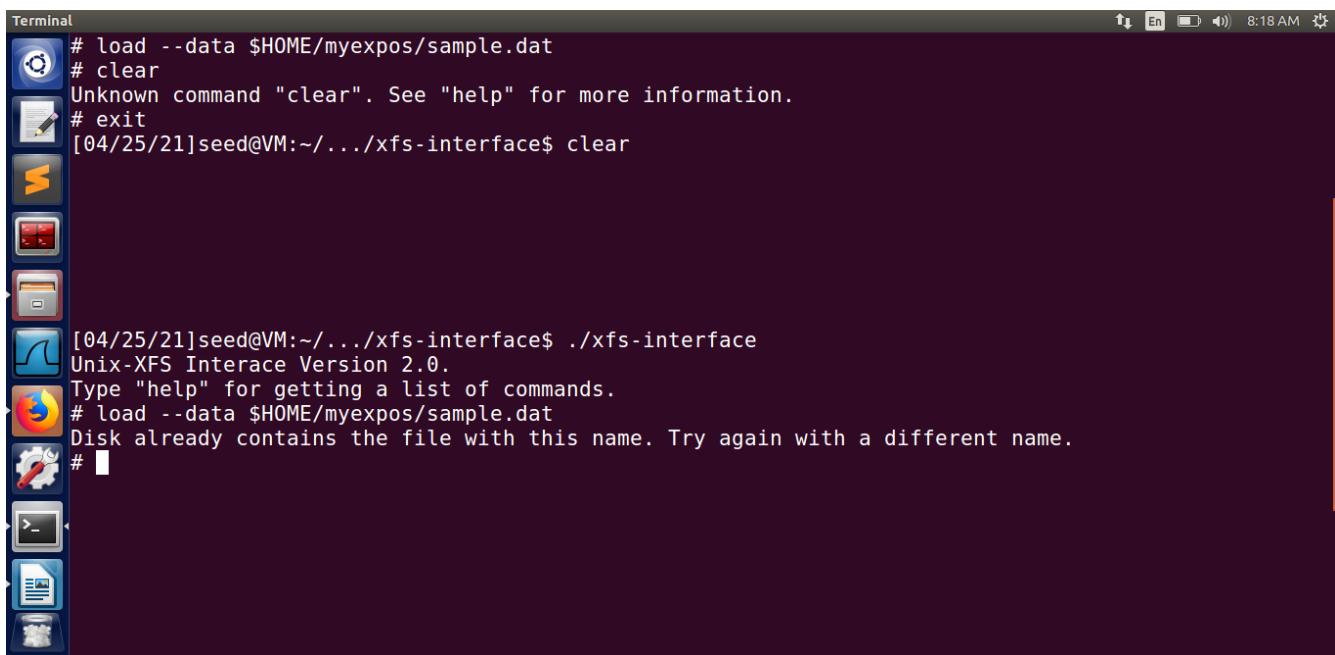
### Contents of sample.dat file -



```
sample.dat (~/myexpos) - gedit
Open ▾ Save
There is a place where the sidewalk ends
And before the street begins,
And there the grass grows soft and white,
And there the sun burns crimson bright,
And there the moon-bird rests from his flight
To cool in the peppermint wind.
```

5. Loading this data file (`$HOME/myexpos/sample.dat`) to the XFS disk from your UNIX machine.

Command - # load --data \$HOME/myexpos/sample.dat



```
# load --data $HOME/myexpos/sample.dat
# clear
Unknown command "clear". See "help" for more information.
# exit
[04/25/21]seed@VM:~/.../xfs-interface$ clear
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# load --data $HOME/myexpos/sample.dat
Disk already contains the file with this name. Try again with a different name.
#
```

- A disk block will be allocated for the file (as `sample.dat` contains less than 512 words) and corresponding to this allocated block (here block 69 - this is because the 1<sup>st</sup> free block is allocated by the allocator), an entry will be marked as 1 (used) in the Disk Free List.



```
51      -      1
52      -      1
53      -      1
54      -      1
55      -      1
56      -      1
57      -      1
58      -      1
59      -      1
60      -      1
61      -      1
62      -      1
63      -      1
64      -      1
65      -      1
66      -      1
67      -      1
68      -      1
69      -      1
70      -      0
71      -      0
72      -      0
73      -      0
74      -      0
75      -      0
```

```
Terminal 491 - 0
492 - 0
493 - 0
494 - 0
495 - 0
496 - 0
497 - 0
498 - 0
499 - 0
500 - 0
501 - 0
502 - 0
503 - 0
504 - 0
505 - 0
506 - 0
507 - 0
508 - 0
509 - 0
510 - 0
511 - 0
No of Free Blocks = 442
Total no of Blocks = 512
#
```

6. Use the **copy** command to copy the *Inode Table*(Inode Table is stored in disk blocks 3 and 4) to a UNIX file (say \$HOME/myexpos/inode\_table.txt).

Command - # copy 3 4 \$HOME/myexpos/inode\_table.txt

```
Terminal [04/25/21]seed@VM:~/myexpos$ ls
download.sh expl inode_table.txt Makefile sample.dat spl xfs-interface xsm
[04/25/21]seed@VM:~/myexpos$
```

Contents of inode\_table.txt.

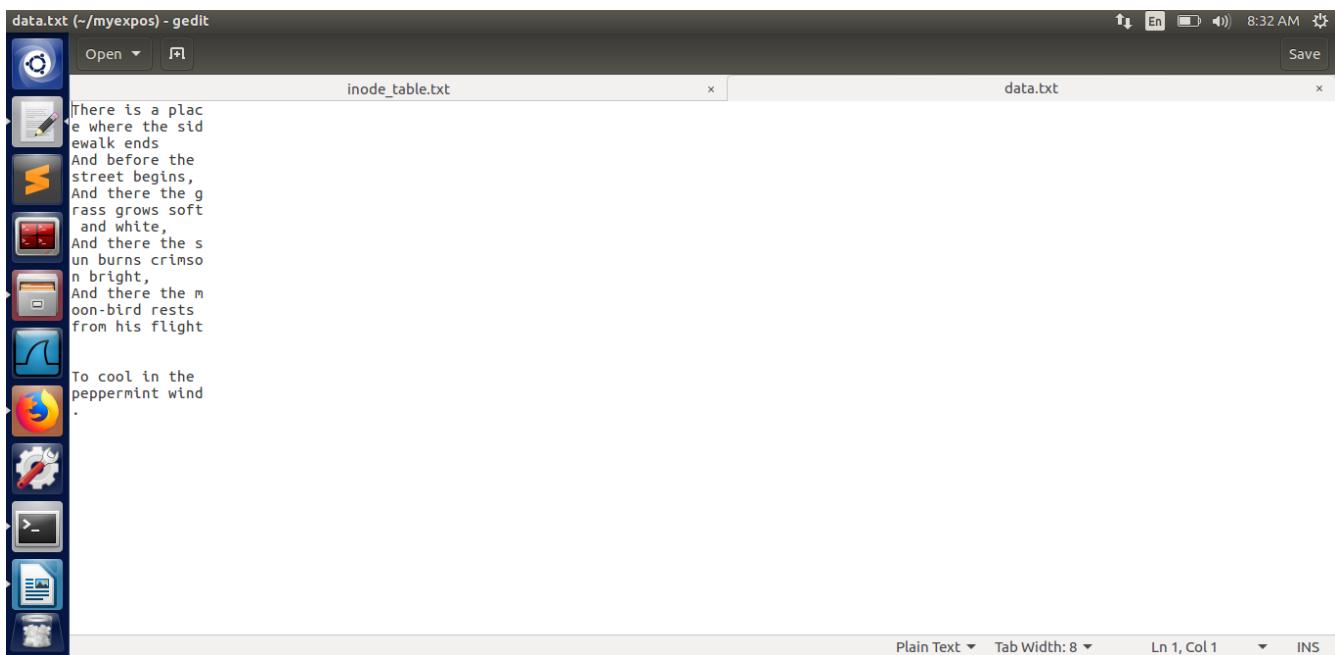
```
inode_table.txt (~~/myexpos) - gedit
Open Save
1
root
512
0
0
-1
-1
-1
5
-1
-1
-1
-1
-1
-1
-1
2
sample.dat
18
1
1
-1
-1
-1
69
-1
-1
-1
-1
-1
-1
-1
-1
-1
```

7. Copy the data blocks from the XFS disk and display it as a UNIX file.

Command - # copy 69 69 \$HOME/myexpos/data.txt

```
Terminal [04/25/21]seed@VM:~/myexpos$ ls  
download.sh  expl  inode_table.txt  Makefile  sample.dat  spl  xfs-interface  xsm  
[04/25/21]seed@VM:~/myexpos$ cd xfs-interface  
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface  
Unix-XFS Interace Version 2.0.  
Type "help" for getting a list of commands.  
# copy 69 69 $HOME/myexpos/data.txt  
# █
```

Each word is displayed in a line because a word in XFS is 16 characters long. Sample data.txt file is shown below.



8. Export the file `sample.dat` to the UNIX file `$HOME/myexpos/data.txt` using `xfs-interface`

*Command - # export sample.dat \$HOME/myexpos/data.txt*

```
[04/25/21]seed@VM:~/myexpos$ ls
download.sh  expl  inode_table.txt  Makefile  sample.dat  spl  xfs-interface  xsm
[04/25/21]seed@VM:~/myexpos$ cd xfs-interface
[04/25/21]seed@VM:~/.xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# copy 69 69 $HOME/myexpos/data.txt
# export sample.dat $HOME/myexpos/data.txt
Segmentation fault
[04/25/21]seed@VM:~/.xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# export sample.dat $HOME/myexpos/data.txt
#
```

---

*Question 1: When a file is created entries are made in the Inode table as well as the Root file. What is the need for this duplication?*

*Answer - The Inode table is stored in the disk and has an entry for each file present in the disk. The eXpoS reserves the first entry in the Inode table for the root file. The root file is a special file containing details about other files stored in the system.*

The reason for this duplication is that root file is designed to be readable by user programs using the *Read* system call (unlike the inode table, which is accessed exclusively by OS routines only). An application readable root file allows implementation of commands like “ls” as user mode programs. *Write* and *Delete* system calls are not permitted on the root file.

The only data in the root file entry of a file that is not present in the inode table is the user name of the owner of the file. The inode table entry of a file contains a user-id of the owner. The user-id value can be used to index into the user table to find the username corresponding to the user-id.

Assignment 1: Copy the contents of Root File (from Block 5 of XFS disk) to a UNIX file \$HOME/myexpos/root\_file.txt and verify that an entry for sample.dat is made in it also.

### **Answer -**

Command - copy 5 5 \$HOME/myexpos/root\_file.txt

Contents of root\_file.txt are shown.

Entry of sample.dat is made in the Root File.

```
Terminal [04/25/21]seed@VM:~/myexpos$ ls
download.sh  expl  inode_table.txt  Makefile  sample.dat  spl  xfs-interface  xsm
[04/25/21]seed@VM:~/myexpos$ cd xfs-interface
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# copy 69 69 $HOME/myexpos/data.txt
# export sample.dat $HOME/myexpos/data.txt
Segmentation fault
[04/25/21]seed@VM:~/.../xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# export sample.dat $HOME/myexpos/data.txt
# export sample.dat $HOME/myexpos/data.txt
# export sample.dat $HOME/myexpos/data1.txt
# copy 5 5 $HOME/myexpos/root_file.txt
#
```

Assignment 2: Delete the `sample.dat` from the XSM machine using `xfs-interface` and note the changes for the entries for this file in *inode table, root file and disk free list* .

Answer -

Command - # rm sample.dat

Disk Free List - 69<sup>th</sup> entry is 0

```
Terminal
51  -  1
52  -  1
53  -  1
54  -  1
55  -  1
56  -  1
57  -  1
58  -  1
59  -  1
60  -  1
61  -  1
62  -  1
63  -  1
64  -  1
65  -  1
66  -  1
67  -  1
68  -  1
69  -  0
70  -  0
71  -  0
72  -  0
73  -  0
74  -  0
75  -  0
```

Root file – No entry for `sample.dat` file.

Inode table – No entry for sample.dat.

## Commands used -

Terminal

494	-	0
495	-	0
496	-	0
497	-	0
498	-	0
499	-	0
500	-	0
501	-	0
502	-	0
503	-	0
504	-	0
505	-	0
506	-	0
507	-	0
508	-	0
509	-	0
510	-	0
511	-	0

No of Free Blocks = 443  
Total no of Blocks = 512

```
# copy 5 5 $HOME/myexpos/root_file.txt
# copy 5 5 $HOME/myexpos/root_file1.txt
# copy 3 4 $HOME/myexpos/inode_file2.txt
#
```

## Stage 3

### BOOTSTRAP LOADER

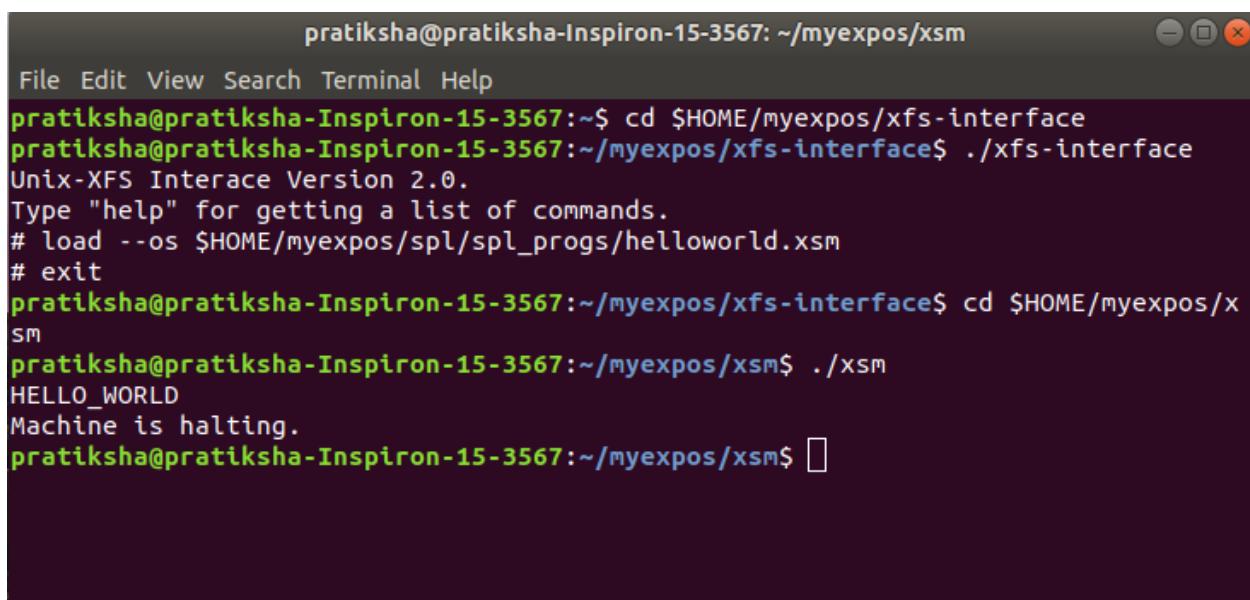
Whenever the machine starts, the execution of the program starts at the first word of the first page(page 0) of the memory. There is a pre-loaded ROM code on Page 0 called the **bootstrap loader**. This code loads the 1<sup>st</sup> disk block (block 0) called the **boot block** from the disk to page 1 of memory and then transferred control (using the jump instruction) to the 1<sup>st</sup> instruction on page 1.

Boot ROM code contains only two instructions, the first instruction is to load the boot block to page 1 and the second instruction is to jump to page 1.

Assembly code to print “HELLO WORLD”:-

```
MOV R0, "HELLO_WORLD"
MOV R16, R0
PORT P1, R16
OUT
HALT
```

Load the file as OS Startup code to disk.xfs using XFS-Interface. Invoke the XFS interface After that running the machine and here we go, it is printing “HELLO\_WORLD”:



The screenshot shows a terminal window titled "pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm". The terminal content is as follows:

```
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# load --os $HOME/myexpos/spl/spl_progs/helloworld.xsm
# exit
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ ./xsm
HELLO_WORLD
Machine is halting.
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$
```

Assignment: Write an assembly program to print numbers from 1 to 20 and run it as the OS Startup code.

Link to the code:-

[https://drive.google.com/file/d/1E8Vp2FJ4\\_0nA12eOudSfejgHHfo5kkHU/view?usp=sharing](https://drive.google.com/file/d/1E8Vp2FJ4_0nA12eOudSfejgHHfo5kkHU/view?usp=sharing)

The output:-

```
e                               pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~$ cd $HOME/myexpos/xfs-interface
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# load --os $HOME/myexpos/spl/spl_progs/Number.xsm
# exit
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ ./xsm
1
2
3
4
5
6
7
8
9
10
11
12
13
14
16
17
18
19
20
Machine is halting.
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ 
```

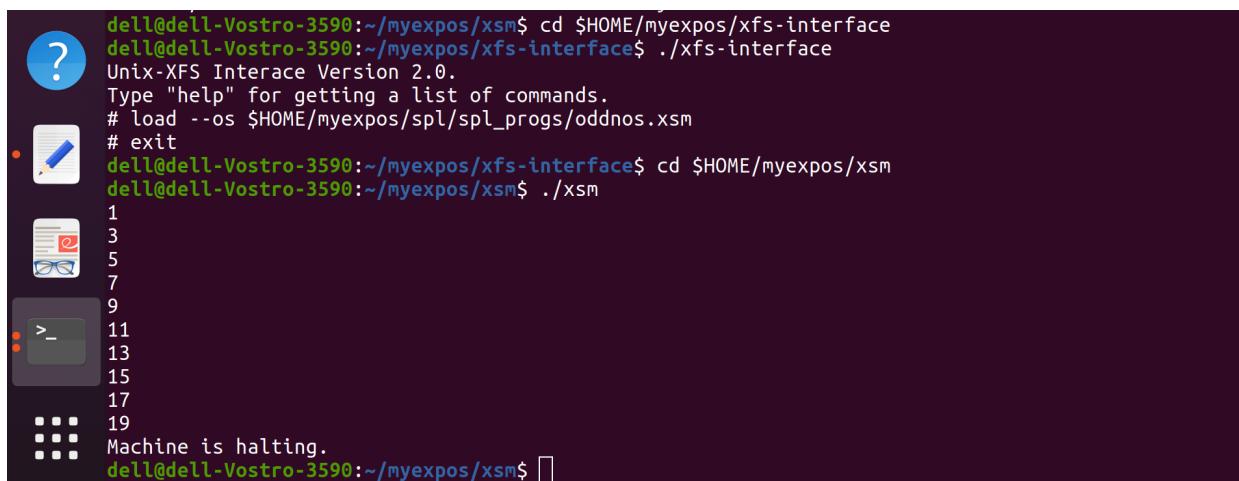
## Stage 4

SPL (Systems Programming Language) allows high-level programs to be written for the XSM machine (eliminating the need to write all the code in assembly language). SPL is not a full-fledged programming language but is an extension to the XSM assembly language with support for high-level constructs like if-then-else, while-do, etc.

SPL doesn't support variables. Instead, you can directly use XSM registers for storing program data

SPL Program to print odd numbers from 1 to 20 :

```
alias counter R0;  
counter = 0;  
while(counter <= 20) do  
    if(counter%2 != 0) then  
        print counter;  
    endif;  
    counter = counter + 1;  
endwhile;
```



```
dell@dell-Vostro-3590:~/myexpos/xsm$ cd $HOME/myexpos/xfs-interface  
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ ./xfs-interface  
Unix-XFS Interface Version 2.0.  
Type "help" for getting a list of commands.  
# load --os $HOME/myexpos/spl/spl_progs/oddnos.xsm  
# exit  
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm  
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm  
1  
3  
5  
7  
9  
11  
13  
15  
17  
19  
Machine is halting.  
dell@dell-Vostro-3590:~/myexpos/xsm$
```

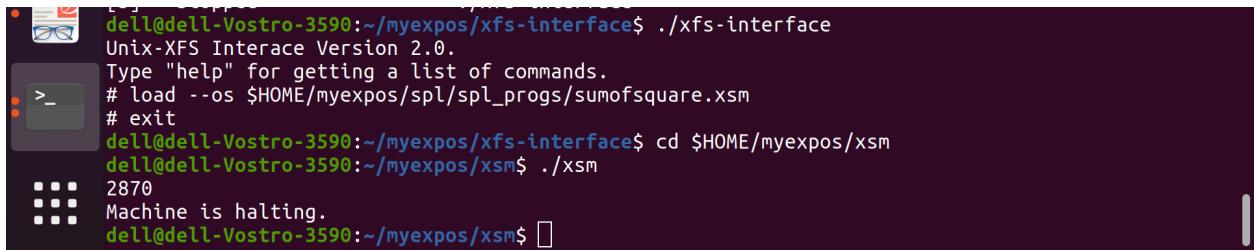
Assignment Question 1: Write the SPL program to print the sum of squares of the first 20 natural numbers. Load it using xfs interface and run the in the machine.

Code

```
alias counter R0;  
alias sum R1;  
alias temp R2;
```

```
counter = 0;  
sum = 0;  
temp = 0;  
  
while(counter <= 20) do  
    temp = counter * counter;  
    sum = sum + temp;  
    counter = counter + 1;  
endwhile;  
print sum;
```

#### Output:



```
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ ./xfs-interface  
Unix-XFS Interface Version 2.0.  
Type "help" for getting a list of commands.  
# load --os $HOME/myexpos/spl/spl_progs/sumofsquare.xsm  
# exit  
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm  
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm  
2870  
Machine is halting.  
dell@dell-Vostro-3590:~/myexpos/xsm$
```

## Stage 5: XSM Debugging

In this stage, I have written an SPL program with a **breakpoint** statement.

The breakpoint statement translates to the BEKP machine instruction and is used for debugging. Suppose the XSM machine is run in the Debug mode on encountering the BRKP instruction. In that case, the machine simulator will suspend the program execution and allow you to inspect the values of the registers, memory, os data structures, etc. Execution resumes only after you instruct the simulator to proceed.

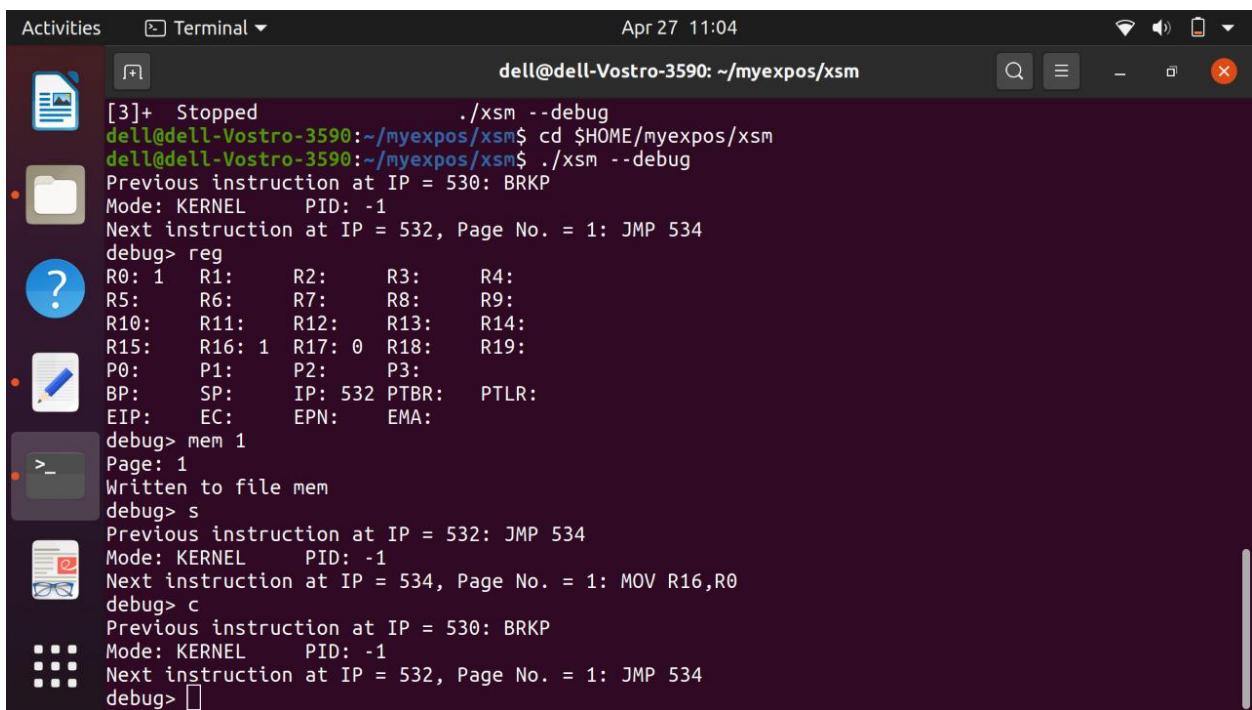
1. SPL code to generate odd numbers from 1 to 10.

```
alias counter R0;
counter = 0;
while(counter <= 10) do
    if(counter%2 != 0) then
        breakpoint;
    endif;
    counter = counter + 1;
endwhile;
```

2. I compiled the program using the SPL compiler.
3. Loaded the compiled xsm code as OS startup code into the XSM disk using the XFS interface.
4. Run the machine in debug mode.  

```
cd $HOME/myexpos/xsm
./xsm --debug
```
5. The Machine pauses after the execution of the first BRKP instruction.

## Output -



A screenshot of a Linux desktop environment, specifically Ubuntu, showing a terminal window. The terminal window is titled 'Terminal' and has the command 'dell@dell-Vostro-3590: ~/myexpos/xsm'. The terminal output shows a debugger session:

```
[3]+ Stopped                  ./xsm --debug
dell@dell-Vostro-3590:~/myexpos/xsm$ cd $HOME/myexpos/xsm
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug
Previous instruction at IP = 530: BRKP
Mode: KERNEL      PID: -1
Next instruction at IP = 532, Page No. = 1: JMP 534
debug> reg
R0: 1   R1:     R2:     R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 1  R17: 0  R18:    R19:
P0:     P1:     P2:     P3:
BP:     SP:     IP: 532 PTBR:    PTLR:
EIP:    EC:     EPN:     EMA:
debug> mem 1
Page: 1
Written to file mem
debug> s
Previous instruction at IP = 532: JMP 534
Mode: KERNEL      PID: -1
Next instruction at IP = 534, Page No. = 1: MOV R16,R0
debug> c
Previous instruction at IP = 530: BRKP
Mode: KERNEL      PID: -1
Next instruction at IP = 532, Page No. = 1: JMP 534
debug>
```

## Stage 6: Running a user program

In this stage, we will be running our first user program, squares.xsm.

*File - expl\_progs/squares.xsm*

This is the user program that we want to run, which calculates the square of the first 5 natural numbers. This code will be loaded as an INIT program by the OS startup code.

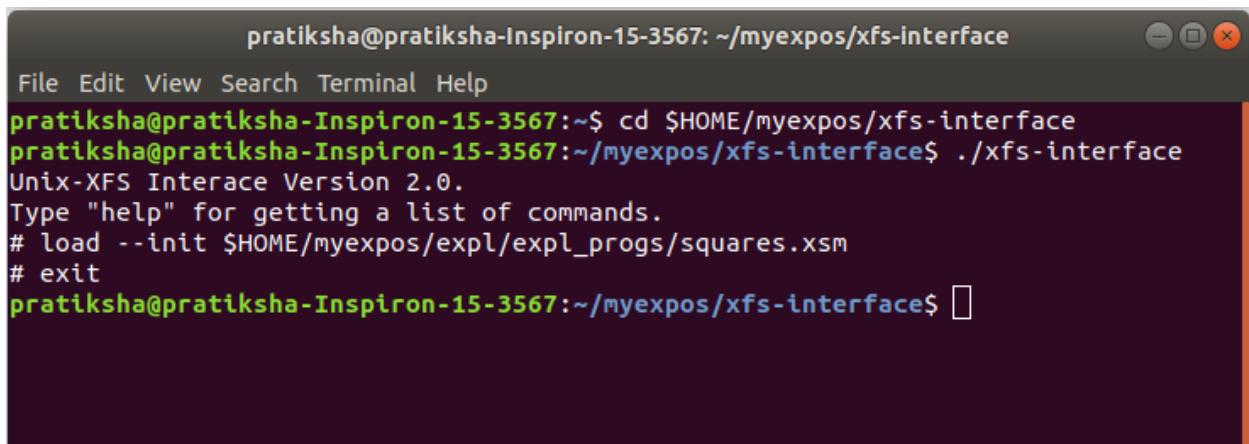
*File- spl\_progs/haltprog.spl*

This spl program has only a halt statement and will be used as interrupt handler 10 and also the exception handler. At the end of the user program, an INT 10 instruction is used to give control back to os and invoke the software interrupt handler 10. Its default action is to halt the machine.

*File- spl\_progs/os\_startup.spl*

This is the os startup code that loads the rest of the program from the disk to memory. It sets up the page table for the INIT program, sets the value for PTBR and PTLR registers, and also sets the stack. Once everything is set, the code transfers the control to the INIT program using IRET instruction.

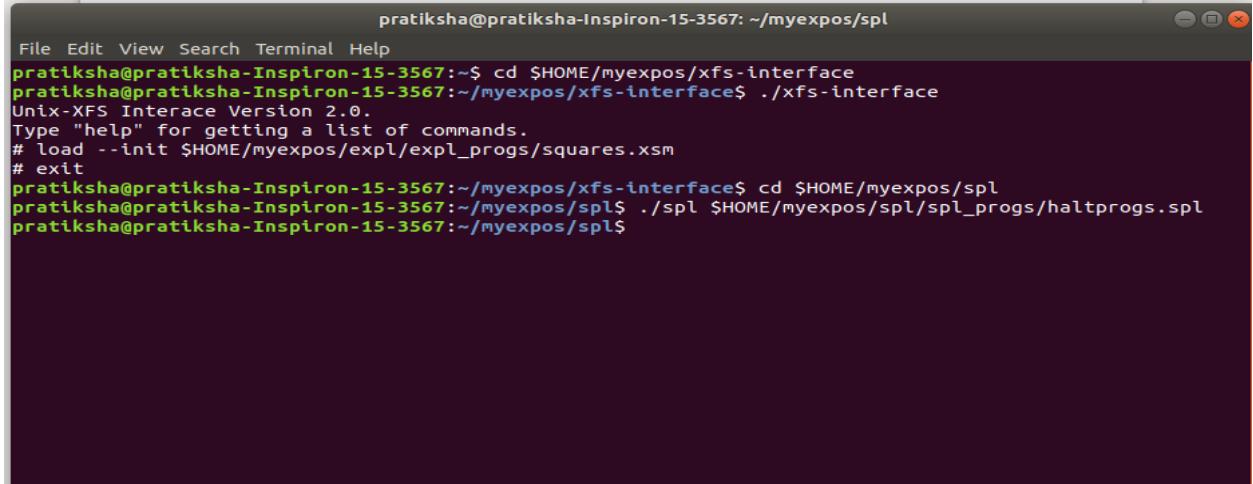
Loading the file squares.xsm to the xsm disk as the INIT program using the XFS interface.



The screenshot shows a terminal window titled "pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xfs-interface". The terminal displays the following command sequence:

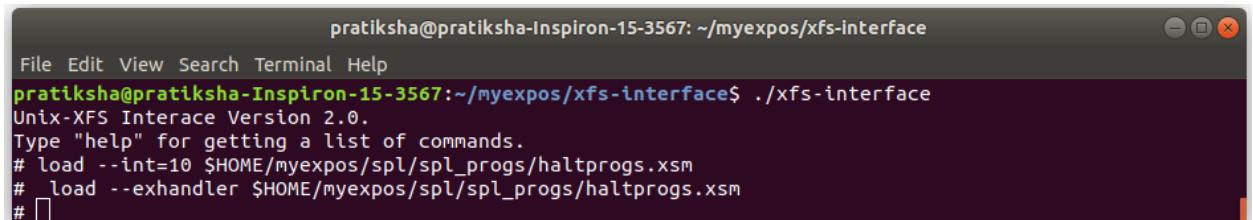
```
pratiksha@pratiksha-Inspiron-15-3567:~$ cd $HOME/myexpos/xfs-interface
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interface Version 2.0.
Type "help" for getting a list of commands.
# load --init $HOME/myexpos/expl/expl_progs/squares.xsm
# exit
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$
```

## Compiling the haltprogs.spl after creating it.



```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/spl
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ cd $HOME/myexpos/xfs-interface
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interface Version 2.0.
Type "help" for getting a list of commands.
# load --init $HOME/myexpos/expl/expl_progs/squares.xsm
# exit
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ cd $HOME/myexpos/spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/haltprogs.spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$
```

## Loading the compiling code as INT 10 from xfs-interface.



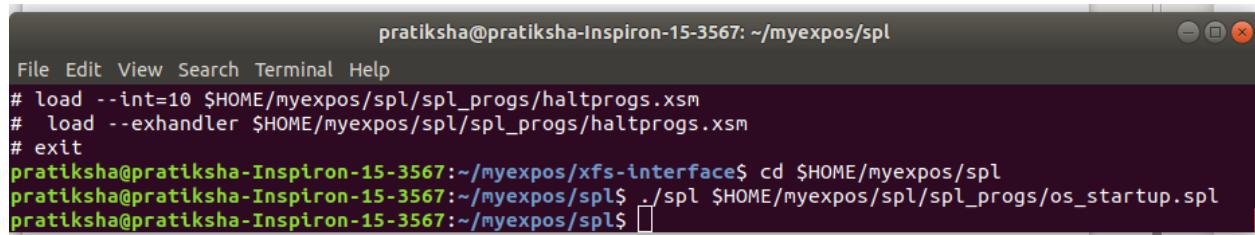
```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xfs-interface
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interface Version 2.0.
Type "help" for getting a list of commands.
# load --int=10 $HOME/myexpos/spl/spl_progs/haltprogs.xsm
# load --exhandler $HOME/myexpos/spl/spl_progs/haltprogs.xsm
#
```

OS startup code:

```
loadi(65,7);
loadi(66,8);
loadi(22,35);
loadi(23,36);
loadi(2, 15);
loadi(3, 16);
PTBR = PAGE_TABLE_BASE;
PTLR = 3;
[PTBR+0] = 65;
[PTBR+1] = "0100";
[PTBR+2] = 66;
[PTBR+3] = "0100";
[PTBR+4] = 76;
```

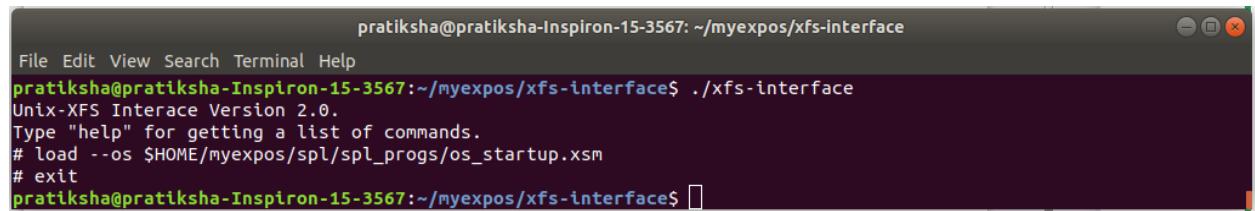
```
[PTBR+5] = "0110";
[76*512] = 0;
SP = 2*512;
ireturn;
```

Saving the OS startup code as \$HOME/myexpos/spl/spl\_progs/os\_startup.spl. Compiler this file using SPL compiler.



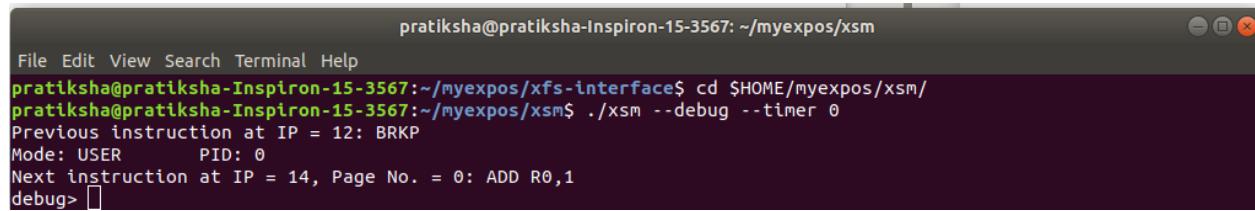
```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/spl
File Edit View Search Terminal Help
# load --int=10 $HOME/myexpos/spl/spl_progs/haltprogs.xsm
# load --exhandler $HOME/myexpos/spl/spl_progs/haltprogs.xsm
# exit
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ cd $HOME/myexpos/spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/os_startup.spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$ 
```

This generates a file \$HOME/myexpos/spl/spl\_progs/os\_startup.xsm. Load this file as os startup code to disk.xfs using xfs interface.



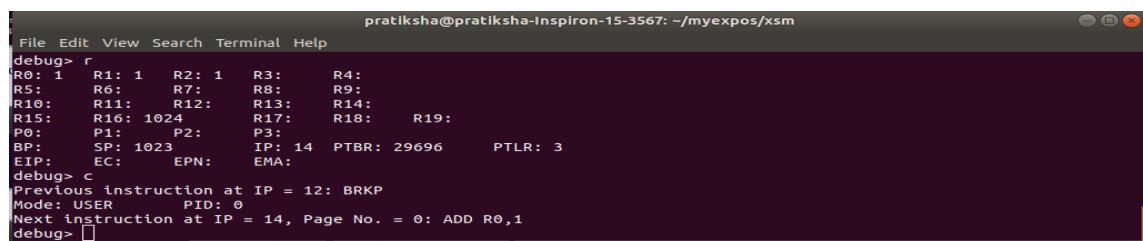
```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xfs-interface
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# load --os $HOME/myexpos/spl/spl_progs/os_startup.xsm
# exit
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ 
```

Running the machine in debug mode:



```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm/
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ ./xsm --debug --timer 0
Previous instruction at IP = 12: BRKP
Mode: USER      PID: 0
Next instruction at IP = 14, Page No. = 0: ADD R0,1
debug> 
```

Viewing R1 in each iteration:-



```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> r
R0: 1    R1: 1    R2: 1    R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:   R11:   R12:   R13:   R14:
R15:   R16: 1024   R17:   R18:   R19:
P0:     P1:     P2:     P3:
BP:   SP: 1023   IP: 14   PTBR: 29696   PTLR: 3
EIP:   EC:     EPN:     EMA:
debug> c
Previous instruction at IP = 12: BRKP
Mode: USER      PID: 0
Next instruction at IP = 14, Page No. = 0: ADD R0,1
debug> 
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> r
R0: 2   R1: 4   R2: 1   R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 1024    R17:    R18:    R19:
P0:     P1:     P2:     P3:
BP:     SP: 1023    IP: 14   PTBR: 29696    PTLR: 3
EIP:    EC:     EPN:     EMA:
debug> c
Previous instruction at IP = 12: BRKP
Mode: USER      PID: 0
Next instruction at IP = 14, Page No. = 0: ADD R0,1
debug> 
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> r
R0: 3   R1: 9   R2: 1   R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 1024    R17:    R18:    R19:
P0:     P1:     P2:     P3:
BP:     SP: 1023    IP: 14   PTBR: 29696    PTLR: 3
EIP:    EC:     EPN:     EMA:
debug> c
Previous instruction at IP = 12: BRKP
Mode: USER      PID: 0
Next instruction at IP = 14, Page No. = 0: ADD R0,1
debug> 
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> r
R0: 4   R1: 16  R2: 1   R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 1024    R17:    R18:    R19:
P0:     P1:     P2:     P3:
BP:     SP: 1023    IP: 14   PTBR: 29696    PTLR: 3
EIP:    EC:     EPN:     EMA:
debug> c
Previous instruction at IP = 12: BRKP
Mode: USER      PID: 0
Next instruction at IP = 14, Page No. = 0: ADD R0,1
debug> 
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> r
R0: 5   R1: 25  R2: 1   R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 1024    R17:    R18:    R19:
P0:     P1:     P2:     P3:
BP:     SP: 1023    IP: 14   PTBR: 29696    PTLR: 3
EIP:    EC:     EPN:     EMA:
debug> c
Machine is halting.
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ 
```

**Its explanation:-**

The files are loaded into the following disk blocks:-

Block no.	Contents
0-1	bootstrap/OS start code
...	...
7-8	init/login code
...	...
15-16	Exception Handler
...	...
35-36	interrupt 10 Routine:Exit
...	...

The ROM code (present on page 0) on the machine loads the OS startup code from disk block 0 to memory page 1 and sets P to 512, which is the first instruction on page 1. memory layout:-

Physical page number	contents
0	Rom code
1	Page for OS start up code
2-3	exception handler
...	...
22-23	Interrupt 10 Routine: Exit
...	...
58	Page Tables*
...	...
65-66	Init/Login code
...	...

The OS startup program loads the disk blocks to their corresponding page numbers.

### **Page table for INIT program**

Each page table entry for a logical page is of 2 words. The first word is the physical page number and the second word contains a series of flags such as reference bit, valid/invalid bit, write permission bit, and dirty bit.

Memory address	Physical page number	flags
PTBR	65	0100
PTBR+2	66	0100
PTBR+4	76	0110

Assignment: Change virtual memory model such that code occupies logical pages 4 and 5 and the stack lies in logical page 8. You will have to modify the user program as well as the os startup code.

Output -

```
Activities Terminal ▾ dell@dell-Vostro-3590: ~/myexpos/xsm Apr 30 17:57
# load --int=10 $HOME/myexpos/spl/spl_progs/haltprog.xsm
# load --exhandler $HOME/myexpos/spl/spl_progs/haltprog.xsm
# load --library $HOME/myexpos/expl/library.lib
# exit
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug
Previous instruction at IP = 12: BRKP
Mode: USER      PID: 0
Next instruction at IP = 14, Page No. = 0: ADD R0,1
debug> s
Previous instruction at IP = 14: ADD R0,1
Mode: USER      PID: 0
Next instruction at IP = 16, Page No. = 0: JMP 2050
debug> s
Previous instruction at IP = 16: JMP 2050
Mode: USER      PID: 0
Next instruction at IP = 2050, Page No. = 4:
debug> s
Previous instruction at IP = 2050:
Mode: KERNEL     PID: 0
Next instruction at IP = 1024, Page No. = 2: HALT
debug> s
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$
```

```
Activities Terminal ▾ dell@dell-Vostro-3590: ~/myexpos/xsm Apr 30 17:59
Previous instruction at IP = 16: JMP 2050
Mode: USER      PID: 0
Next instruction at IP = 2050, Page No. = 4:
debug> r
R0: 2    R1: 1    R2: 1    R3:      R4:
R5:      R6:      R7:      R8:      R9:
R10:     R11:     R12:     R13:     R14:
R15:     R16: 1024      R17:     R18:     R19:
P0:      P1:      P2:      P3:
BP:      SP: 1023      IP: 2050      PTBR: 29696      PTLR: 3
EIP:      EC:      EPN:      EMA:
debug> s
Previous instruction at IP = 2050:
Mode: KERNEL     PID: 0
Next instruction at IP = 1024, Page No. = 2: HALT
debug> r
R0: 2    R1: 1    R2: 1    R3:      R4:
R5:      R6:      R7:      R8:      R9:
R10:     R11:     R12:     R13:     R14:
R15:     R16: 1024      R17:     R18:     R19:
P0:      P1:      P2:      P3:
BP:      SP: 1023      IP: 1024      PTBR: 29696      PTLR: 3
EIP: 2050      EC: 2      EPN:      EMA: 2050
debug> s
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$
```

## **Stage 7: (ABI and XEXE Format)**

An application binary interface (ABI) is the interface between a user program and the kernel.

In this stage, we will rewrite the user program and OS startup code of Stage 6 in compliance with expos ABI.

The INIT program must be modified to comply with the XEXE executable format.

## Modified init program code

```
0  
2056  
0  
0  
0  
0  
0  
0  
0  
MOV R0, 1  
MOV R2, 5  
GE R2, R0  
JZ R2, 2074  
MOV R1, R0  
MUL R1, R0  
BRKP  
ADD R0, 1  
JMP 2058  
INT 10
```

## Modified OS startup program code

```
//Loads library code from disk blocks 13 and 14 to physical pages 63 and 64  
loadi(63,13);  
loadi(64,14);
```

```
//Loads INIT program from 7,8 blocks to 65,66 physical pages  
loadi(65,7);  
loadi(66,8);
```

```
//Loads interrupt routine 10 from 35,36 blocks to 22,23 physical pages
loadi(22,35);
loadi(23,36);

//Loads exception handler from 15,16 blocks to 2,3 physical pages
loadi(2,15);
loadi(3,16);
PTBR = PAGE_TABLE_BASE;
PTLR = 10;

//Library
[PTBR+0] = 63;
[PTBR+1] = "0100";
[PTBR+2] = 64;
[PTBR+3] = "0100";

//Heap
[PTBR+4] = 78;
[PTBR+5] = "0110";
[PTBR+6] = 79;
[PTBR+7] = "0110";

//Code
[PTBR+8] = 65;
[PTBR+9] = "0100";
[PTBR+10] = 66;
[PTBR+11] = "0100";
[PTBR+12] = -1;
[PTBR+13] = "0000";
[PTBR+14] = -1;
[PTBR+15] = "0000";

//Stack
[PTBR+16] = 76;
[PTBR+17] = "0110";
[PTBR+18] = 77;
[PTBR+19] = "0110";

//2nd entry of header of executable contain entry point
[76*512] = [65*512 + 1];
```

```
SP = 8*512;
```

```
ireturn;
```

1. Compile and load the modified OS startup Code.

```
cd $HOME/myexpos/spl
```

```
./spl $HOME/myexpos/spl/spl_progs/os_startup.spl
```

```
# load --library ..../expl/library.lib
```

```
# load --os $HOME/myexpos/spl/spl_progs/os_startup.xsm
```

```
# exit
```

2. Load the modified user program.

```
# load --init $HOME/myexpos/expl/expl_progs/squares.xsm
```

3. Run the machine in debug mode.

```
cd $HOME/myexpos/xsm/
```

```
./xsm --debug --timer 0
```

Activities Terminal ▾ dell@dell: ~ /myexpos/xsm Apr 29 21:33

```
EIP: EC: EPN: EMA:  
debug> s  
Previous instruction at IP = 2070: ADD R0,1  
Mode: USER PID: 0  
Next instruction at IP = 2072, Page No. = 4: JMP 2058  
debug> s  
Previous instruction at IP = 2072: JMP 2058  
Mode: USER PID: 0  
Next instruction at IP = 2058, Page No. = 4: MOV R2,5  
debug> s  
Previous instruction at IP = 2058: MOV R2,5  
Mode: USER PID: 0  
Next instruction at IP = 2060, Page No. = 4: GE R2,R0  
debug> s  
Previous instruction at IP = 2060: GE R2,R0  
Mode: USER PID: 0  
Next instruction at IP = 2062, Page No. = 4: JZ R2,2074  
debug> r  
R0: 3 R1: 4 R2: 1 R3: R4:  
R5: R6: R7: R8: R9:  
R10: R11: R12: R13: R14:  
R15: R16: 4096 R17: 2056 R18: R19:  
P0: P1: P2: P3:  
BP: SP: 4095 IP: 2062 PTBR: 29696 PTLR: 10  
EIP: EC: EPN: EMA:  
debug>
```

Activities Terminal ▾ dell@dell-Vostro-3590: ~ /myexpos/xsm Apr 30 00:38

```
Mode: USER PID: 0  
Next instruction at IP = 2058, Page No. = 4: MOV R2,5  
debug> s  
Previous instruction at IP = 2058: MOV R2,5  
Mode: USER PID: 0  
Next instruction at IP = 2060, Page No. = 4: GE R2,R0  
debug> reg  
R0: 4 R1: 9 R2: 5 R3: R4:  
R5: R6: R7: R8: R9:  
R10: R11: R12: R13: R14:  
R15: R16: 4096 R17: 2056 R18: R19:  
P0: P1: P2: P3:  
BP: SP: 4095 IP: 2060 PTBR: 29696 PTLR: 10  
EIP: EC: EPN: EMA:  
debug> c  
Previous instruction at IP = 2068: BRKP  
Mode: USER PID: 0  
Next instruction at IP = 2070, Page No. = 4: ADD R0,1  
debug> reg  
R0: 4 R1: 16 R2: 1 R3: R4:  
R5: R6: R7: R8: R9:  
R10: R11: R12: R13: R14:  
R15: R16: 4096 R17: 2056 R18: R19:  
P0: P1: P2: P3:  
BP: SP: 4095 IP: 2070 PTBR: 29696 PTLR: 10  
EIP: EC: EPN: EMA:
```

Assignment: Change the user program to compute cubes of the first five numbers.

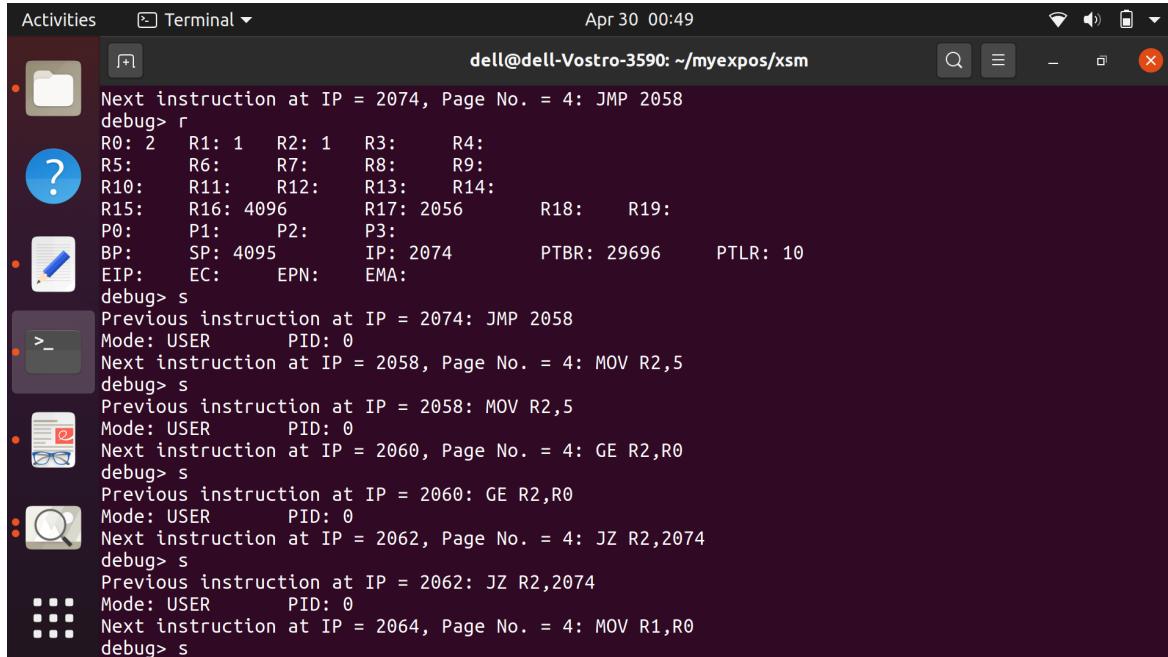
Code

0

2056

```
0  
0  
0  
0  
0  
0  
MOV R0,1  
MOV R2,5  
GE R2,R0  
JZ R2,2074  
MOV R1,R0  
MUL R1,R0  
MUL R1,R0  
BRKP  
ADD R0,1  
JMP 2058  
INT 10
```

## R1 stores the cube



The screenshot shows a terminal window titled "Terminal" with the command "dell@dell-Vostro-3590: ~/myexpos/xsm". The terminal displays assembly code and a debugger session:

```
Activities Terminal Apr 30 00:49 dell@dell-Vostro-3590: ~/myexpos/xsm
debug> r
R0: 2    R1: 1    R2: 1    R3:    R4:
R5:      R6:    R7:    R8:    R9:
R10:   R11:   R12:   R13:   R14:
R15:   R16: 4096    R17: 2056    R18:    R19:
P0:      P1:    P2:    P3:
BP:    SP: 4095    IP: 2074    PTBR: 29696    PTLR: 10
EIP:    EC:    EPN:    EMA:
debug> s
Previous instruction at IP = 2074: JMP 2058
Mode: USER      PID: 0
Next instruction at IP = 2058, Page No. = 4: MOV R2,5
debug> s
Previous instruction at IP = 2058: MOV R2,5
Mode: USER      PID: 0
Next instruction at IP = 2060, Page No. = 4: GE R2,R0
debug> s
Previous instruction at IP = 2060: GE R2,R0
Mode: USER      PID: 0
Next instruction at IP = 2062, Page No. = 4: JZ R2,2074
debug> s
Previous instruction at IP = 2062: JZ R2,2074
Mode: USER      PID: 0
Next instruction at IP = 2064, Page No. = 4: MOV R1,R0
debug> s
```

Activities Terminal ▾ Apr 30 00:49

dell@dell-Vostro-3590: ~/myexpos/xsm

```
Next instruction at IP = 2062, Page No. = 4: JZ R2,2074
debug> s
Previous instruction at IP = 2062: JZ R2,2074
Mode: USER      PID: 0
Next instruction at IP = 2064, Page No. = 4: MOV R1,R0
debug> s
Previous instruction at IP = 2064: MOV R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2066, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2066: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2068, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2068: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: BRKP
debug> r
R0: 2    R1: 8    R2: 1    R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:   R11:   R12:   R13:   R14:
R15:   R16: 4096   R17: 2056     R18:     R19:
P0:     P1:     P2:     P3:
BP:     SP: 4095   IP: 2070     PTBR: 29696    PTLR: 10
EIP:    EC:     EPN:     EMA:
debug> 
```

Activities Terminal ▾ Apr 30 00:50

dell@dell-Vostro-3590: ~/myexpos/xsm

```
Next instruction at IP = 2062, Page No. = 4: JZ R2,2074
debug> s
Previous instruction at IP = 2062: JZ R2,2074
Mode: USER      PID: 0
Next instruction at IP = 2064, Page No. = 4: MOV R1,R0
debug> s
Previous instruction at IP = 2064: MOV R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2066, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2066: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2068, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2068: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: BRKP
debug> r
R0: 3    R1: 27   R2: 1    R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:   R11:   R12:   R13:   R14:
R15:   R16: 4096   R17: 2056     R18:     R19:
P0:     P1:     P2:     P3:
BP:     SP: 4095   IP: 2070     PTBR: 29696    PTLR: 10
EIP:    EC:     EPN:     EMA:
debug> 
```

Activities Terminal ▾ dell@dell-Vostro-3590: ~/myexpos/xsm Apr 30 00:50

```
Next instruction at IP = 2062, Page No. = 4: JZ R2,2074
debug> s
Previous instruction at IP = 2062: JZ R2,2074
Mode: USER      PID: 0
Next instruction at IP = 2064, Page No. = 4: MOV R1,R0
debug> s
Previous instruction at IP = 2064: MOV R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2066, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2066: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2068, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2068: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: BRKP
debug> r
R0: 4    R1: 64   R2: 1    R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 4096    R17: 2056    R18:     R19:
P0:     P1:     P2:     P3:
BP:     SP: 4095    IP: 2070    PTBR: 29696    PTLR: 10
EIP:    EC:     EPN:     EMA:
debug>
```

Activities Terminal ▾ dell@dell-Vostro-3590: ~/myexpos/xsm Apr 30 00:51

```
Next instruction at IP = 2064, Page No. = 4: MOV R1,R0
debug> s
Previous instruction at IP = 2064: MOV R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2066, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2066: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2068, Page No. = 4: MUL R1,R0
debug> s
Previous instruction at IP = 2068: MUL R1,R0
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: BRKP
debug> s
Previous instruction at IP = 2070: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2072, Page No. = 4: ADD R0,1
debug> r
R0: 5    R1: 125  R2: 1    R3:     R4:
R5:     R6:     R7:     R8:     R9:
R10:    R11:    R12:    R13:    R14:
R15:    R16: 4096    R17: 2056    R18:     R19:
P0:     P1:     P2:     P3:
BP:     SP: 4095    IP: 2072    PTBR: 29696    PTLR: 10
EIP:    EC:     EPN:     EMA:
debug>
```

## Stage 8) Handling timer Interrupt

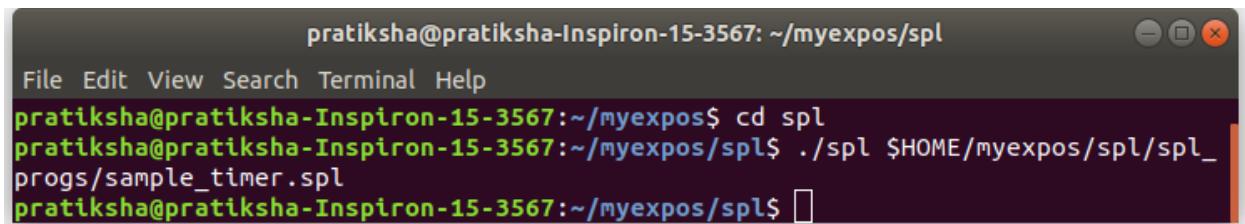
If the xsm simulator is run with the timer set to some value – say 20, then every time the machine completes the execution of 20 instructions in user mode, the timer device will send a hardware signal to interrupt the machine execution. The machine will push the IP value of the user-mode instruction to the stack and pass control to the timer interrupt handler at physical address 2048.

Modification in OS startup code:

OS startup code has to be modified to load the timer interrupt routine from disk blocks 17 and 18 to memory pages 4 and 5.

```
loadi(4, 17);
loadi(5, 18);
```

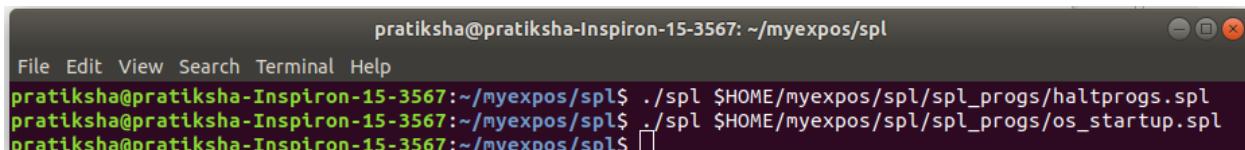
Creating a file name sample\_timer.spl that prints the “TIMER” and returns it to the user program. Then compiling it. (here the user program is the same as in the previous stage)



A screenshot of a terminal window titled "pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/spl". The window shows the following command sequence:

```
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos$ cd spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/sample_timer.spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$
```

Also, compiling the os\_startup and os\_startup to avoid any kind of compiling error.



A screenshot of a terminal window titled "pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/spl". The window shows the following command sequence:

```
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/haltprogs.spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/os_startup.spl
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/spl$
```

In xfs-interface, loading the process squares.xsm as init process.

Loading haltprogs.xsm with int 10.

Loading haltprogs.xsm as the interrupt handler.

Loading the os\_startup.xsm in the xfs disk.

Load the library that is present in expl as library.lib.

Loading the sample\_timer.xsm for int=timer.

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos$ cd xfs-interface
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interace Version 2.0.
Type "help" for getting a list of commands.
# load --init $HOME/expos-roadmap/stage8/expl_progs/squares.xsm
File /home/pratiksha/expos-roadmap/stage8/expl_progs/squares.xsm not found.
# load --init $HOME/myexpos/expl_progs/squares.xsm
File /home/pratiksha/myexpos/expl_progs/squares.xsm not found.
# load --init $HOME/myexpos/expl/expl_progs/squares.xsm
# load --int==10 $HOME/myexpos/spl/spl_progs/haltprogs.xsm
# load --exhandler $HOME/myexpos/spl/spl_progs/haltprogs.xsm
# load --os $HOME/myexpos/spl/spl_progs/os_startup.xsm
# load --library $HOME/myexpos/expl/library.lib
# load --int=timer $HOME/myexpos/spl/spl_progs/sample_timer.xsm
# exit
```

As the user program here will print the squares of first five natural numbers.

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ cd $HOME/myexpos/xsm/
pratiksha@pratiksha-Inspiron-15-3567:~/myexpos/xsm$ ./xsm --debug --timer 2
TIMER
TIMER
TIMER
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> r
R0: 1    R1: 1    R2: 1    R3:      R4:
R5:      R6:      R7:      R8:      R9:
R10:     R11:     R12:     R13:     R14:
R15:     R16: TIMER    R17: 2056      R18:      R19:
P0:      P1: TIMER    P2:      P3:
BP:      SP: 4095      IP: 2070      PTBR: 29696      PTLR: 10
EIP:      EC:      EPN:      EMA:
debug> c
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> c
TIMER
TIMER
TIMER
TIMER
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> r
R0: 2    R1: 4    R2: 1    R3:      R4:
R5:      R6:      R7:      R8:      R9:
R10:     R11:     R12:     R13:     R14:
R15:     R16: TIMER    R17: 2056      R18:      R19:
P0:      P1: TIMER    P2:      P3:
BP:      SP: 4095      IP: 2070      PTBR: 29696      PTLR: 10
EIP:     EC:      EPN:      EMA:
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> c
TIMER
TIMER
TIMER
TIMER
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> r
R0: 3    R1: 9    R2: 1    R3:      R4:
R5:      R6:      R7:      R8:      R9:
R10:     R11:     R12:     R13:     R14:
R15:     R16: TIMER    R17: 2056      R18:      R19:
P0:      P1: TIMER    P2:      P3:
BP:      SP: 4095      IP: 2070      PTBR: 29696      PTLR: 10
EIP:     EC:      EPN:      EMA:
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> c
TIMER
TIMER
TIMER
TIMER
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> r
R0: 4    R1: 16   R2: 1    R3:      R4:
R5:      R6:      R7:      R8:      R9:
R10:     R11:     R12:     R13:     R14:
R15:     R16: TIMER    R17: 2056      R18:      R19:
P0:      P1: TIMER    P2:      P3:
BP:      SP: 4095      IP: 2070      PTBR: 29696      PTLR: 10
EIP:     EC:      EPN:      EMA:
debug> c
```

```
pratiksha@pratiksha-Inspiron-15-3567: ~/myexpos/xsm
File Edit View Search Terminal Help
debug> c
TIMER
TIMER
TIMER
TIMER
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> r
R0: 5    R1: 25   R2: 1    R3:     R4:
R5:    R6:     R7:     R8:     R9:
R10:   R11:   R12:   R13:   R14:
R15:   R16: TIMER    R17: 2056    R18:     R19:
P0:    P1: TIMER    P2:     P3:
BP:    SP: 4095     IP: 2070     PTBR: 29696    PTLR: 10
EIP:   EC:     EPN:     EMA:
debug> c
```

Page table for the corresponding process:

Memory Address	Physical Page Number	Flags
PTBR	63	0100
PTBR + 2	64	0100
PTBR + 4	78	0110
PTBR + 6	79	0110
PTBR + 8	65	0100
PTBR + 10	66	0100
PTBR + 12	-1	0000
PTBR + 14	-1	0000
PTBR + 16	76	0110
PTBR + 18	77	0110

## Stage 9: Handling Kernel Stack

eXpOS requires that when the OS enters an interrupt handler that runs in kernel mode, the interrupt handler must switch to a different stack. This requirement is to prevent user level “hacks” into the kernel through the stack

To isolate the kernel from the user stack, the OS kernel must maintain two stacks for a program - **a user stack and a kernel stack**. In eXpOS, one page called the user area page is allocated for each process.

Whenever there is a transfer of program control from the user mode to the kernel during interrupts (or exceptions), the interrupt handler will change the stack to the kernel stack of the program.

OS Startup code

```
//Loads library code from disk blocks 13 and 14 to physical pages 63 and 64  
loadi(63,13);  
loadi(64,14);
```

```
//Loads INIT program from 7,8 blocks to 65,66 physical pages  
loadi(65,7);  
loadi(66,8);
```

```
//Loads interrupt routine 10 from 35,36 blocks to 22,23 physical pages  
loadi(22,35);  
loadi(23,36);
```

```
//Loads exception handler from 15,16 blocks to 2, 3 physical pages  
loadi(2,15);  
loadi(3,16);
```

```
//Loads timer interrupt routine from 17,18 blocks to 4,5 physical pages  
loadi(4, 17);  
loadi(5, 18);
```

```
PTBR = PAGE_TABLE_BASE;  
PTLR = 10;
```

```
//Library  
[PTBR+0] = 63;  
[PTBR+1] = "0100";
```

```
[PTBR+2] = 64;
[PTBR+3] = "0100";

//Heap
[PTBR+4] = 78;
[PTBR+5] = "0110";
[PTBR+6] = 79;
[PTBR+7] = "0110";

//Code
[PTBR+8] = 65;
[PTBR+9] = "0100";
[PTBR+10] = 66;
[PTBR+11] = "0100";
[PTBR+12] = -1;
[PTBR+13] = "0000";
[PTBR+14] = -1;
[PTBR+15] = "0000";

//Stack
[PTBR+16] = 76;
[PTBR+17] = "0110";
[PTBR+18] = 77;
[PTBR+19] = "0110";

//2nd entry of header of executable contain entry point
[76*512] = [65*512 + 1];

SP = 8*512;

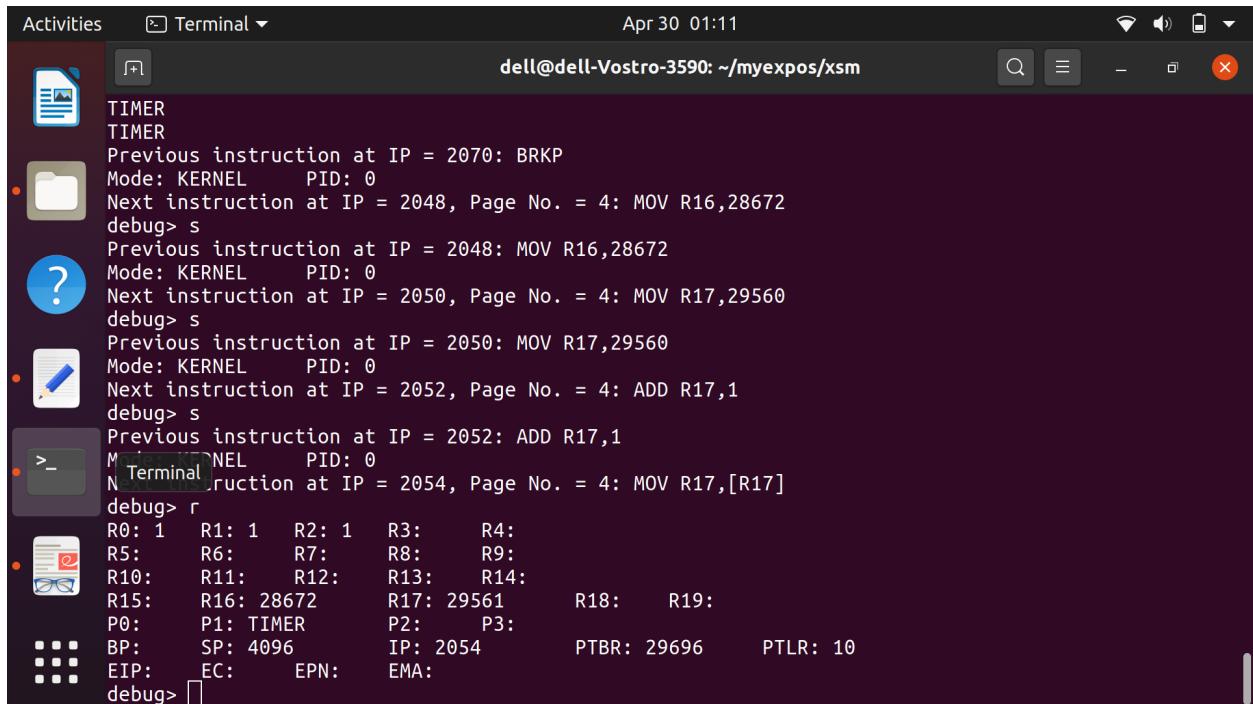
//Process table
//Setting User page area to physical page 80 (first available free page)
//Setting PID to 0
//PROCESS_TABLE = 28672
[PROCESS_TABLE + 11] = 80;
[PROCESS_TABLE + 1] = 0;

//Second field of System status table set to PID 0
[SYSTEM_STATUS_TABLE + 1] = 0;

ireturn;
```

## Timer Program Code

```
//PID obtained from System status table and stored in corresponding process table entry  
[PROCESS_TABLE + ([SYSTEM_STATUS_TABLE + 1] * 16) + 13] = SP;  
  
// Setting SP to UArea Page number * 512 - 1  
SP = [PROCESS_TABLE + ([SYSTEM_STATUS_TABLE + 1] * 16) + 11] * 512 - 1;  
  
//Saving the user context  
backup;  
  
print "TIMER";  
  
//Restoring the user context  
restore;  
  
//Setting SP to user SP saved in process table.  
SP = [PROCESS_TABLE + ([SYSTEM_STATUS_TABLE + 1] * 16) + 13];  
  
lreturn;
```



The screenshot shows a terminal window on a Linux desktop. The title bar reads "Activities Terminal" and the command line shows the user's session: "dell@dell-Vostro-3590: ~/myexpos/xsm". The terminal displays assembly code and a register dump. The assembly code includes instructions like BRKP, MOV R16,28672, ADD R17,1, and MOV R17,[R17]. The register dump shows values for R0 through R19, P0, P1, P2, P3, BP, SP, IP, PTBR, and PTLR.

```
 TIMER  
 TIMER  
 Previous instruction at IP = 2070: BRKP  
 Mode: KERNEL PID: 0  
 Next instruction at IP = 2048, Page No. = 4: MOV R16,28672  
 debug> s  
 Previous instruction at IP = 2048: MOV R16,28672  
 Mode: KERNEL PID: 0  
 Next instruction at IP = 2050, Page No. = 4: MOV R17,29560  
 debug> s  
 Previous instruction at IP = 2050: MOV R17,29560  
 Mode: KERNEL PID: 0  
 Next instruction at IP = 2052, Page No. = 4: ADD R17,1  
 debug> s  
 Previous instruction at IP = 2052: ADD R17,1  
 Mode: KERNEL PID: 0  
 Next instruction at IP = 2054, Page No. = 4: MOV R17,[R17]  
 debug> r  
 R0: 1 R1: 1 R2: 1 R3: R4:  
 R5: R6: R7: R8: R9:  
 R10: R11: R12: R13: R14:  
 R15: R16: 28672 R17: 29561 R18: R19:  
 P0: P1: TIMER P2: P3:  
 BP: SP: 4096 IP: 2054 PTBR: 29696 PTLR: 10  
 EIP: EC: EPN: EMA:  
 debug>
```

Assignment:- : Print the process id of currently executing process in timer interrupt before returning to user mode. You can look up this value from the System Status Table.

```

//PID obtained from System status table and stored in the corresponding process table entry
[PROCESS_TABLE + ([SYSTEM_STATUS_TABLE + 1] * 16) + 13] = SP;

// Setting SP to UArea Page number * 512 - 1
SP = [PROCESS_TABLE + ([SYSTEM_STATUS_TABLE + 1] * 16) + 11] * 512 - 1;

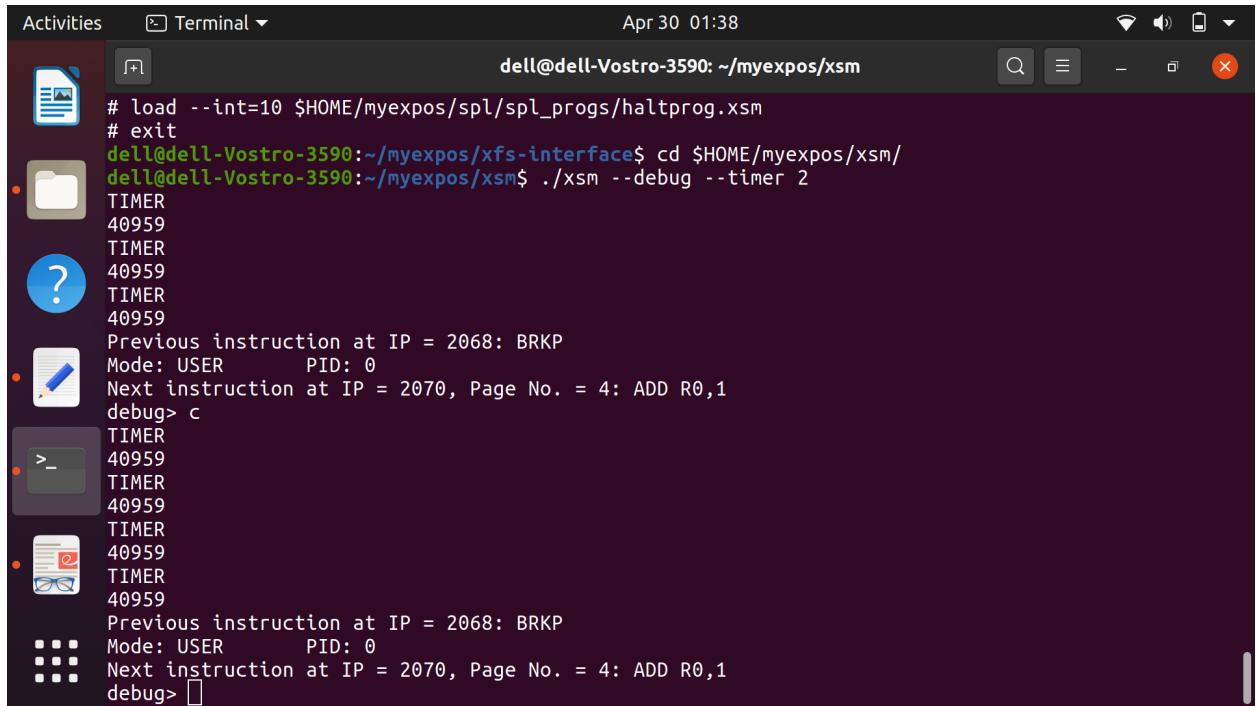
//Saving the user context
backup;

print "TIMER";

//Restoring the user context
restore;
print SP;
//Setting SP to user SP saved in process table.
SP = [PROCESS_TABLE + ([SYSTEM_STATUS_TABLE + 1] * 16) + 13];

Ireturn;

```



The screenshot shows a terminal window titled 'Terminal' with the command 'dell@ dell-Vostro-3590: ~/myexpos/xsm'. The terminal displays assembly code and debugger interactions:

```

# load --int=10 $HOME/myexpos/spl/spl_progs/haltprog.xsm
# exit
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm/
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug --timer 2
TIMER
40959
TIMER
40959
TIMER
40959
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> c
TIMER
40959
TIMER
40959
TIMER
40959
Previous instruction at IP = 2068: BRKP
Mode: USER      PID: 0
Next instruction at IP = 2070, Page No. = 4: ADD R0,1
debug> 

```

## Stage 10: Console output

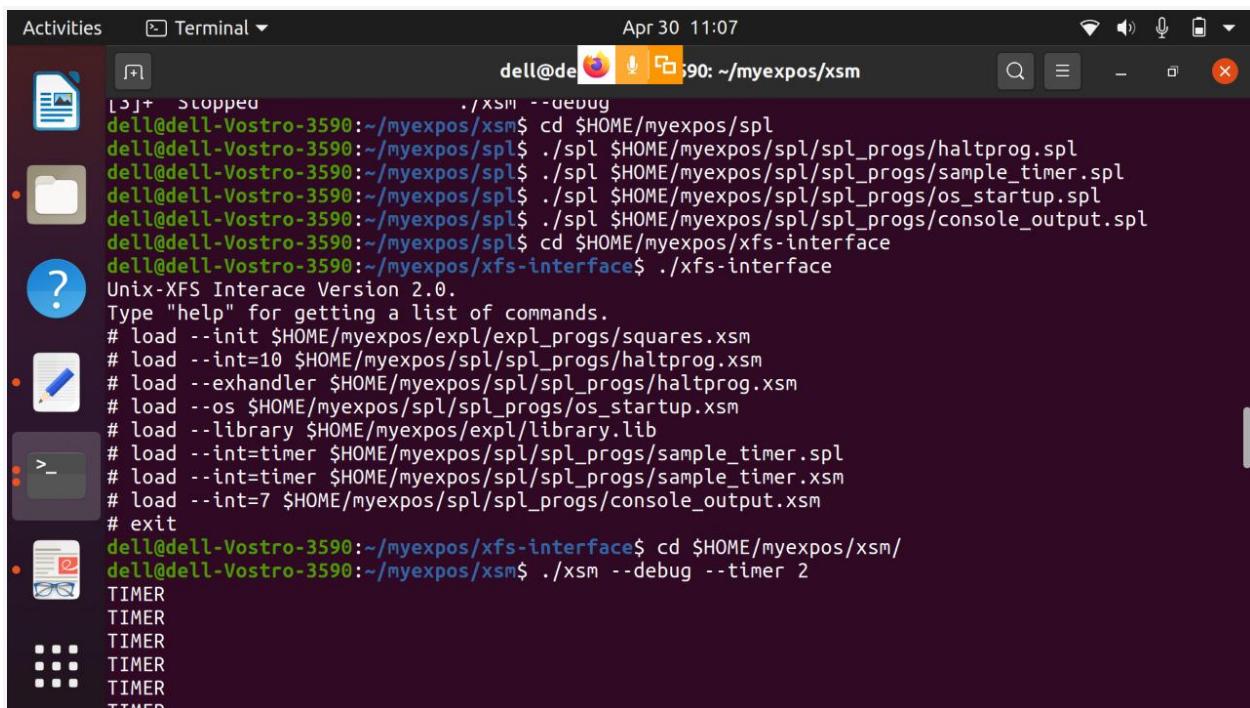
In this stage, we will rewrite the user program to enable it to write to the terminal. The primary data structures we are concerned with here are the system status table, process table, and page table. The write system call is serviced by interrupt routine 7.

A system call is an OS routine that can be invoked from a user program. The OS provides system call routines for various services like writing to a file/console, forking a process, etc. Each system call routine is written inside some software interrupt handler.

### **Console\_output.spl**

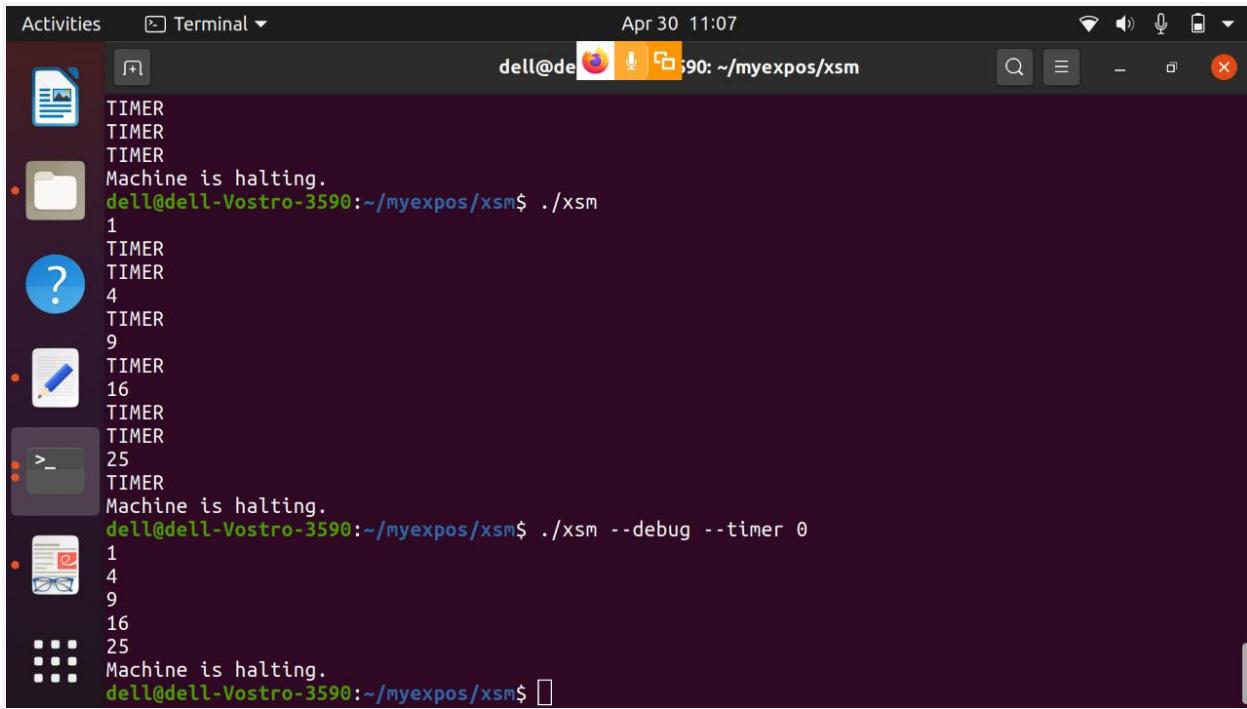
The interrupt 7 routine programs are used for printing to the terminal screen. This program requires 3 arguments- argument 1 is -2 (file descriptor for terminal), argument 2 is the value to be printed, and argument 3 is could be anything (exists solely for the convention). If argument 1 is -2, prints value to the terminal, else, returns -1.

The output of squares.xsm -



The screenshot shows a terminal window titled "Terminal" with the command line interface. The terminal window has a dark background with light-colored text. It displays the following session:

```
dell@dell-Vostro-3590:~/myexpos/xsm$ cd $HOME/myexpos/spl
dell@dell-Vostro-3590:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/haltprog.spl
dell@dell-Vostro-3590:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/sample_timer.spl
dell@dell-Vostro-3590:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/os_startup.spl
dell@dell-Vostro-3590:~/myexpos/spl$ ./spl $HOME/myexpos/spl/spl_progs/console_output.spl
dell@dell-Vostro-3590:~/myexpos/spl$ cd $HOME/myexpos/xfs-interface
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interface Version 2.0.
Type "help" for getting a list of commands.
# load --init $HOME/myexpos/expl/expl_progs/squares.xsm
# load --int=10 $HOME/myexpos/spl/spl_progs/haltprog.xsm
# load --exhandler $HOME/myexpos/spl/spl_progs/haltprog.xsm
# load --os $HOME/myexpos/spl/spl_progs/os_startup.xsm
# load --library $HOME/myexpos/expl/library.lib
# load --int=timer $HOME/myexpos/spl/spl_progs/sample_timer.spl
# load --int=timer $HOME/myexpos/spl/spl_progs/sample_timer.xsm
# load --int=7 $HOME/myexpos/spl/spl_progs/console_output.xsm
# exit
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm/
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug --timer 2
TIMER
TIMER
TIMER
TIMER
TTMEDE
```



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "Terminal" and the date and time are "Apr 30 11:07". The terminal content shows a list of processes and their timer values. The processes are labeled with icons: a document, a folder, a question mark, a pen, and a terminal. The timer values are: 1, 4, 9, 16, 25, and "Machine is halting." The terminal prompt is "dell@dell-Vostro-3590:~/myexpos/xsm\$". The terminal window has a dark background with light-colored text.

```
Activities Terminal ▾ Apr 30 11:07
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm
1
4
9
16
25
TIMER
TIMER
TIMER
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug --timer 0
1
4
9
16
25
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$
```

---

## Question -

Why should we calculate the physical address of userSP-3 and userSP-1 separately instead of calculating one and adding/subtracting the difference from the calculated value?

Answer -

Suppose the physical address corresponding to the logical address in userSP be - say 3000. it may not be the case that 2997 is the physical address corresponding to the logical address userSP-3. Similarly, the physical address corresponding to userSP-1 need not be 2999. The problem is that the stack of a process spreads over two pages, and these two physical pages need not be contiguous. Hence logical addresses can not be calculated in such a way.

---

**Assignment 1:** Write a program to print the first 20 numbers and run the system with a timer enabled.

Program files have been uploaded to google classroom.

Output - First 20 numbers (1,2,3...,20) are printed on the terminal.

Activities Terminal ▾ dell@dell-Vostro-3590: ~/myexpos/xsm Apr 30 16:03

```
5
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$ cd $HOME/myexpos/xfs-interface
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ ./xfs-interface
Unix-XFS Interface Version 2.0.
Type "help" for getting a list of commands.
# load --init $HOME/myexpos/expl/expl_progs/squares.xsm
# load --int=10 $HOME/myexpos/spl/spl_progs/haltprog.xsm
# load --exhandler $HOME/myexpos/spl/spl_progs/haltprog.xsm
# load --os $HOME/myexpos/spl/spl_progs/os_startup.xsm
# load --library $HOME/myexpos/expl/library.lib
# load --int=timer $HOME/myexpos/spl/spl_progs/sample_timer.xsm
# load --int=7 $HOME/myexpos/spl/spl_progs/console_output.xsm
# exit
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm/
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug --timer 0
1
2
3
4
5
6
7
8
9
10
11
```

Activities Terminal ▾ dell@dell-Vostro-3590: ~/myexpos/xsm Apr 30 16:01

```
# load --int=7 $HOME/myexpos/spl/spl_progs/console_output.xsm
# exit
dell@dell-Vostro-3590:~/myexpos/xfs-interface$ cd $HOME/myexpos/xsm/
dell@dell-Vostro-3590:~/myexpos/xsm$ ./xsm --debug --timer 0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Machine is halting.
dell@dell-Vostro-3590:~/myexpos/xsm$ 
```