

NAME : PUTHUMBAKU HIMAKESH

ROLL NO: CB.SC.U4CSE24741

SUBJECT: OPERATING SYSTEM (23CSE214)

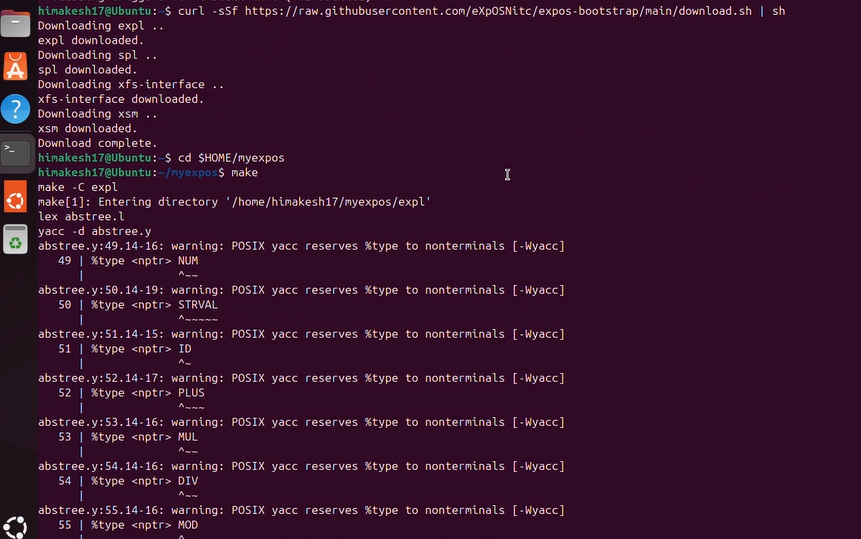
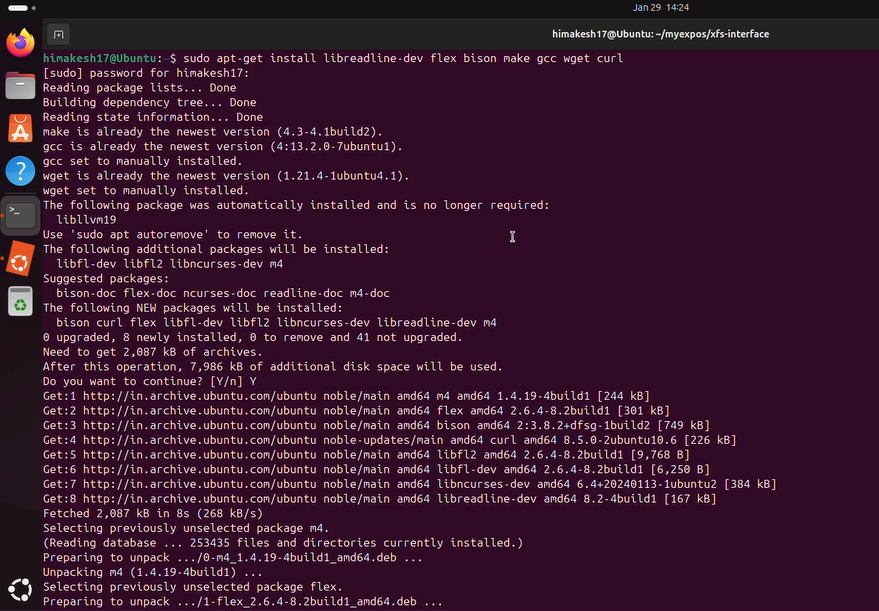
# **STAGE-1**

1. Install prerequisites such as gcc, make, readline, flex/lex, bison/yacc

sudo apt-get install libreadline-dev flex bison make gcc wget curl

1. Download the complete eXpOS support package from the official GitHub repositories

curl -sSf https://raw.githubusercontent.com/eXpOSNitc/expos-bootstrap/main/download.sh | sh



1. Creating myexpos directory



1. Make to build all the components
   1. **spl** in $HOME/myexpos/spl folder
   2. **expl** in $HOME/myexpos/expl folder
   3. **xfs-interface** in $HOME/myexpos/xfs-interface folder
   4. **xsm** in $HOME/myexpos/xsm folder

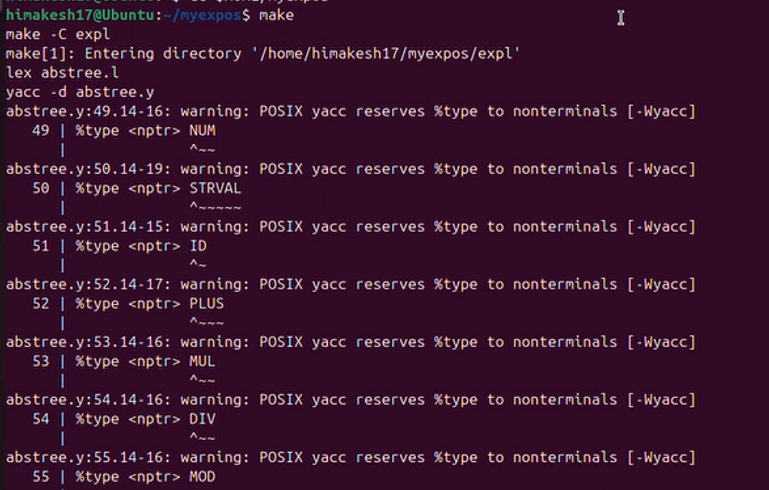
expl – converts your program into machine code

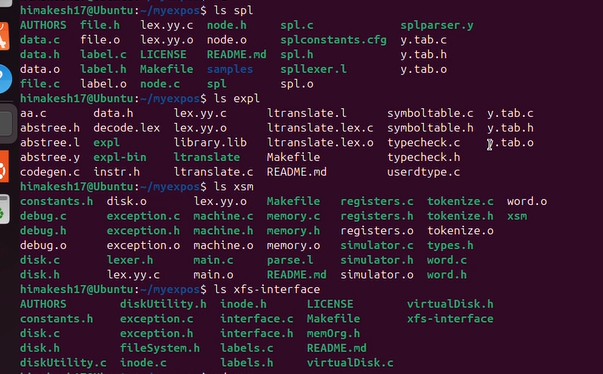
spl – converts OS code into machine code

xfs-interface – to manage file and disk

xsm – runs the compiled machine code

test – *Check if things work*



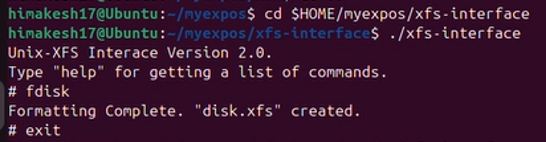


STAGE-2

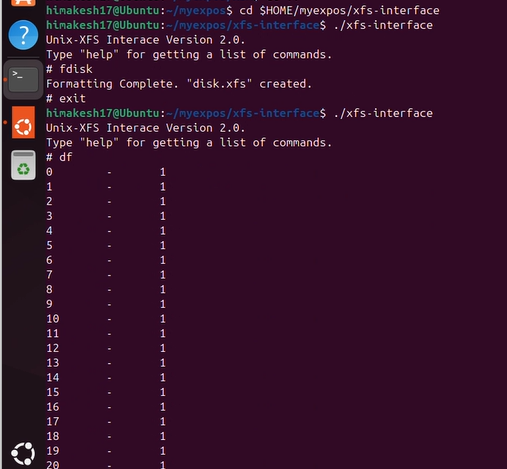
1. Run the XFS Interface



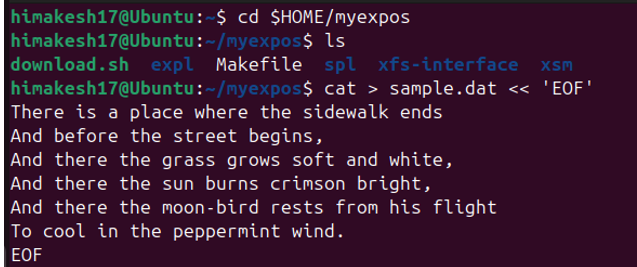
1. *fdisk - prepares (formats) the virtual disk so eXpOS can use it*

**

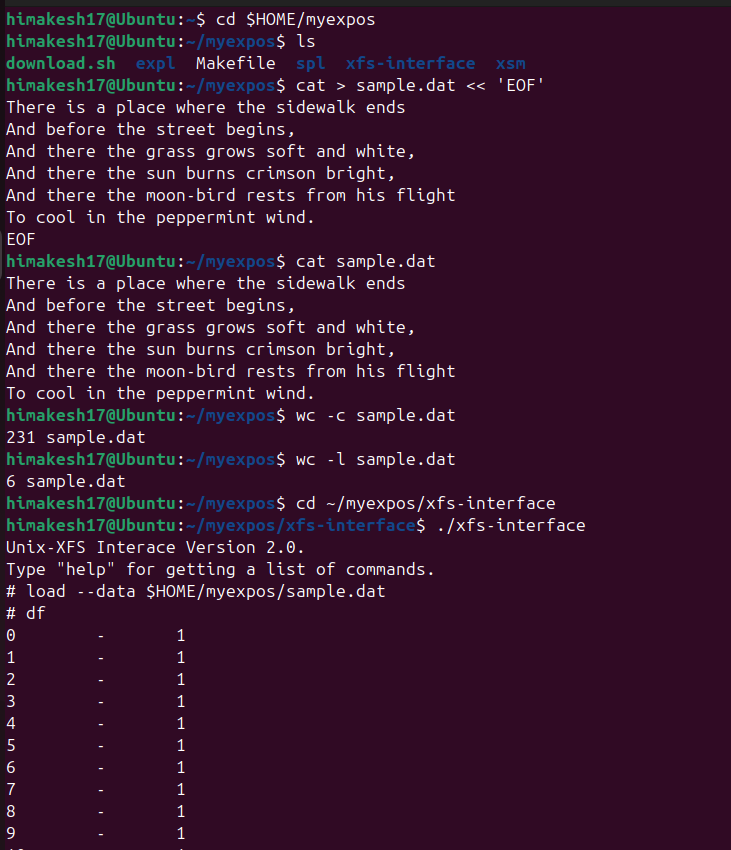
*df - shows how much disk space is used and free*



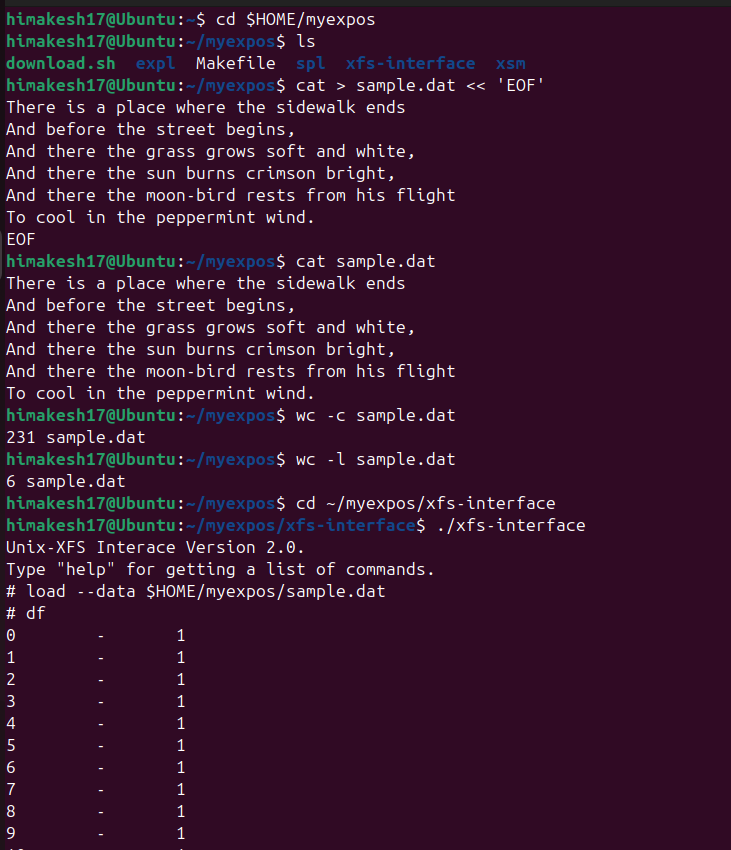
1. Creating a file with sample data and save the file as $HOME/myexpos/sample.dat



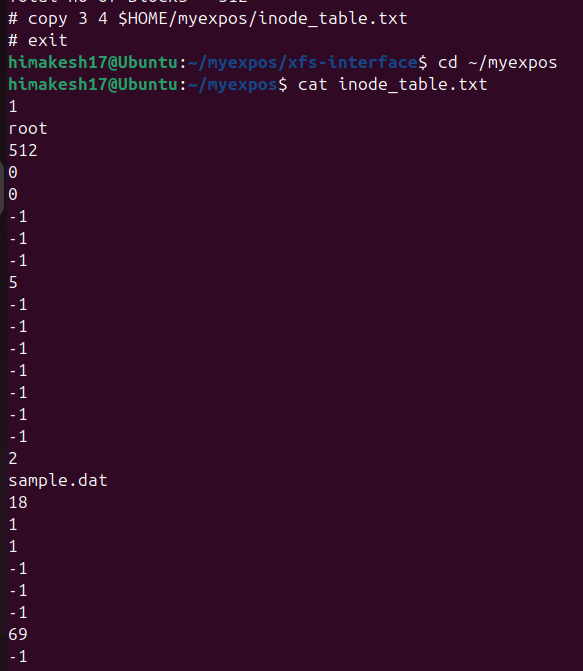
1. Load this data file $HOME/myexpos/sample.dat to the XFS disk



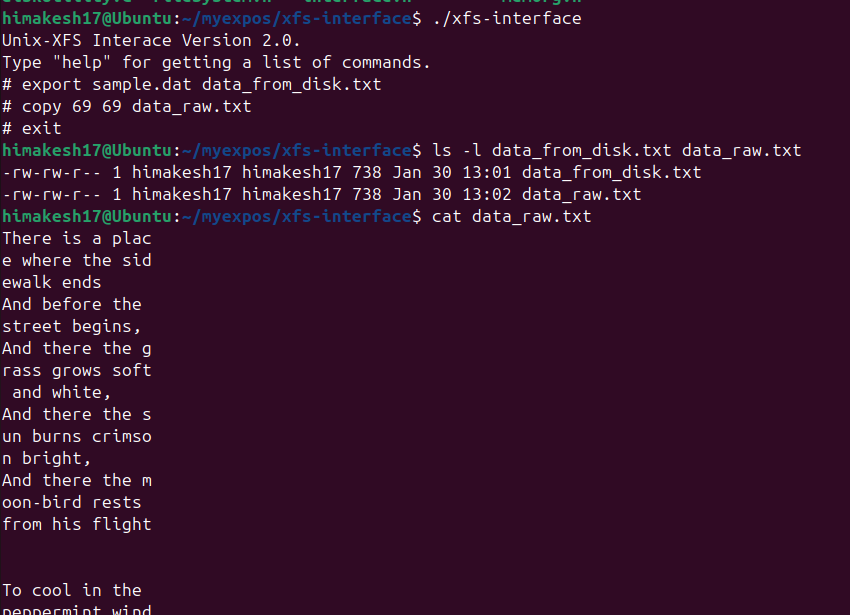
1. One free disk block is allocated
2. New inode entry is created to sotore file name, size, user id and data block
3. An entry for this file will be made in the Root File also



Inode table is stored in disk blocks 3 and 4

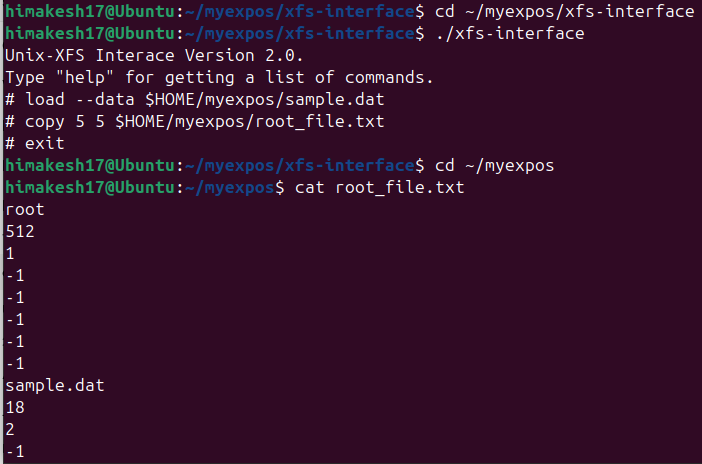


Copy the data blocks from the XFS disk and display it as a UNIX file



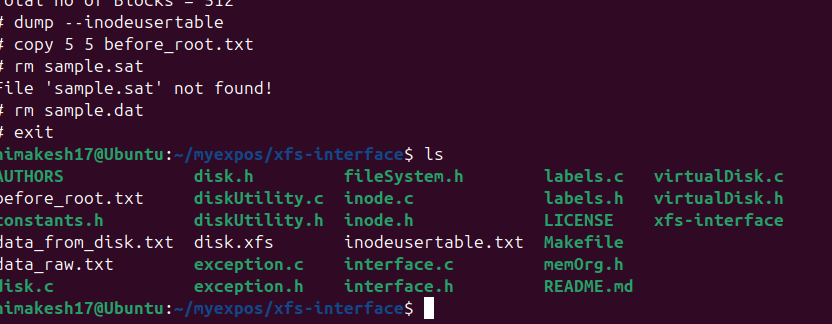
Assignment 1

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.



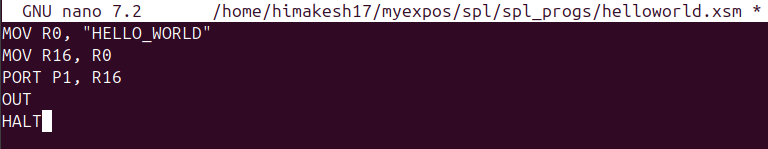
Assignment 2

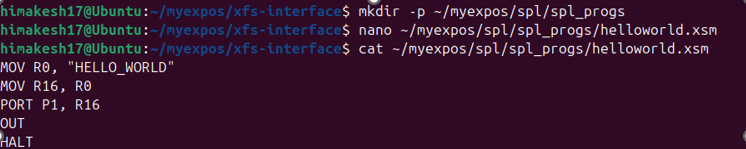
Delete the sample.dat from the XSM machine using xfs-interface and note the changes for the entries for this file in *inode table, root file and disk free list*.



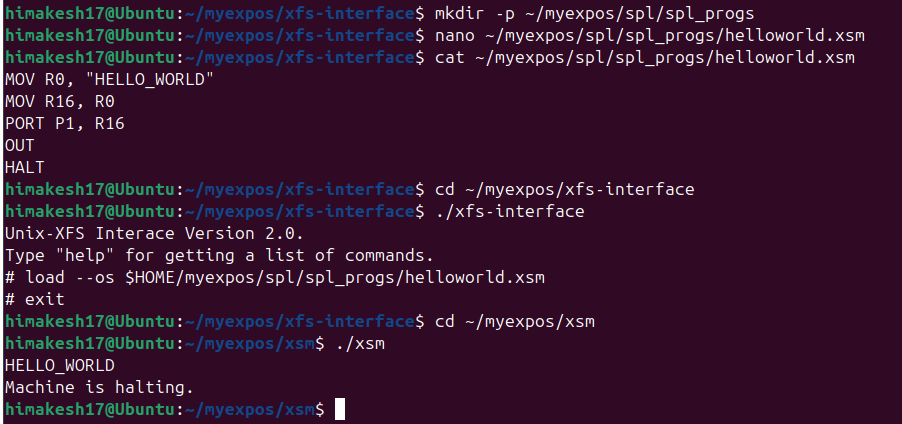
STAGE-3

1. Create the assembly program to print "HELLO\_WORLD". The assembly code to print "HELLO\_WORLD" :

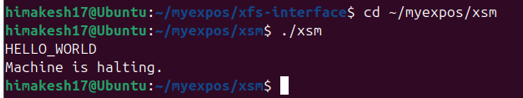




1. Load the file as OS Startup code to disk.xfs using XFS-Interface.



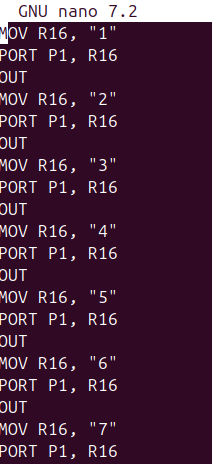
output



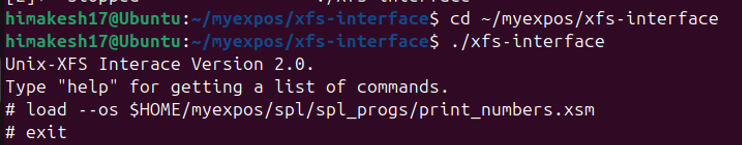
**Assignment 1**

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

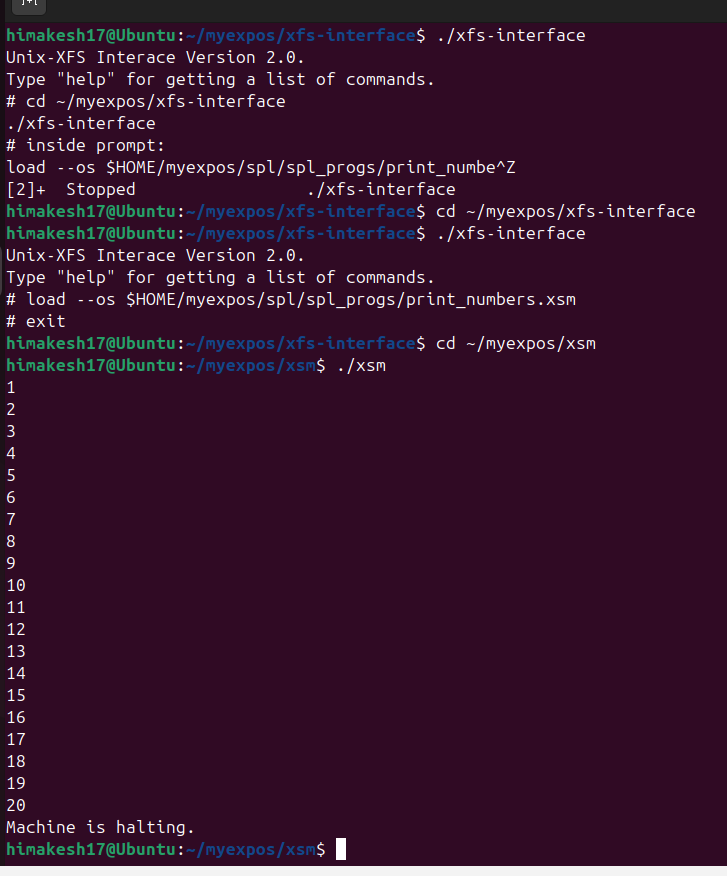
1. Create the assembly program to print 1 to 20 "print\_numbers". The assembly code to print :



1. Load the file as OS Startup code to disk.xfs using XFS-Interface

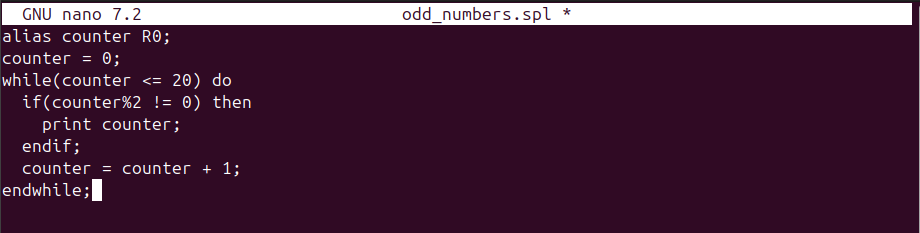


1. Run the machine



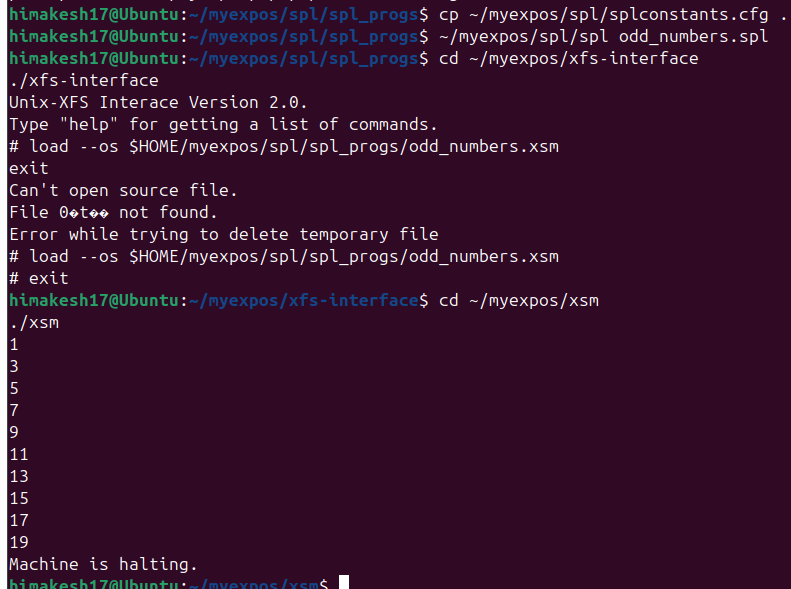
STAGE-4

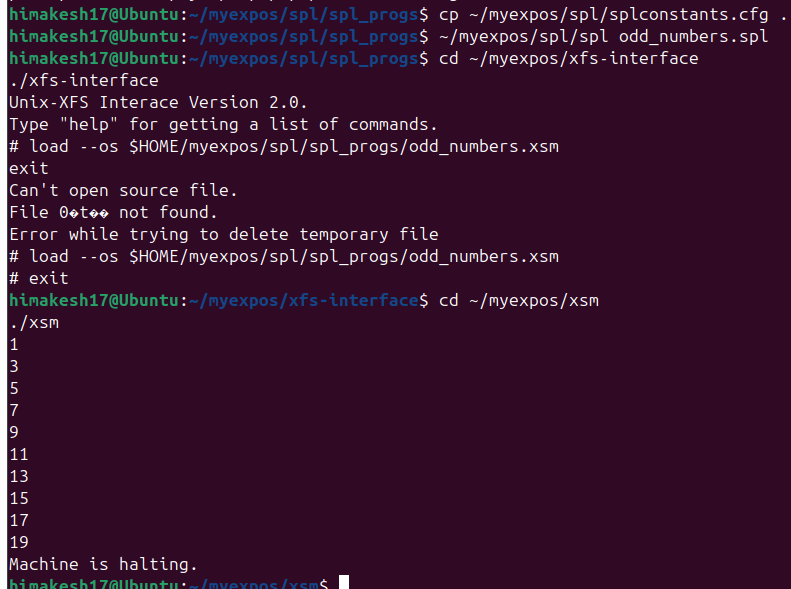
1. Create the program to print odd numbers form 1 to 20 using SPL





1. Save the file and compile this SPL program using the commands and load the file generated by the SPL compiler

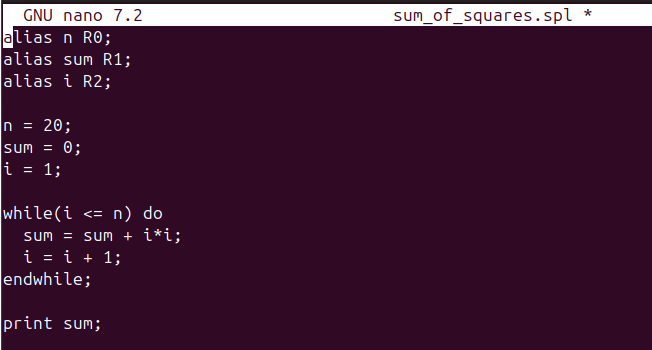




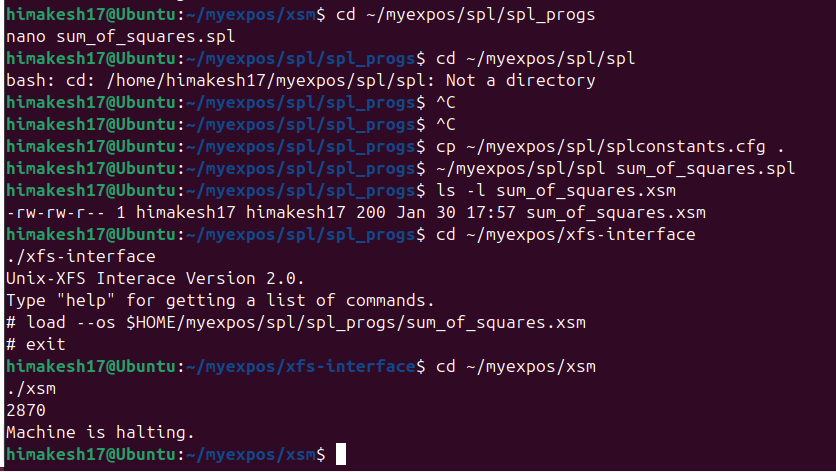
Assignment 1

write the spl program to print sum of squares of the first 20 natural numbers. load it using xfs interface and run the in the machine.

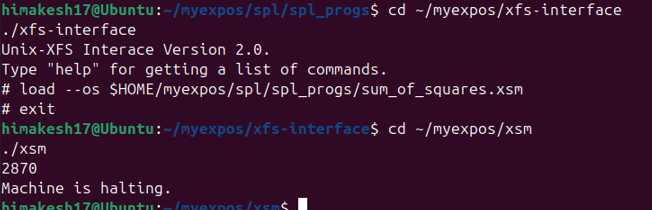
1. Create the program to print the sum of squares of the first 20 natural numbers



1. Save the file and compile this SPL program using commands

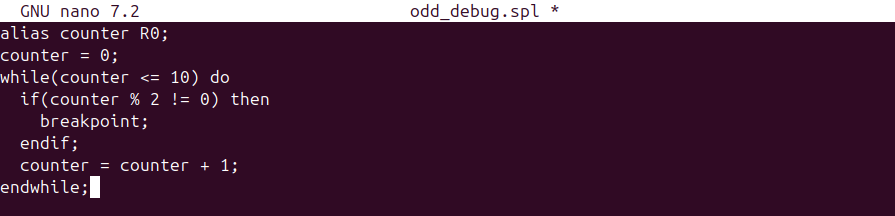


1. Load the file generated by the SPL compiler

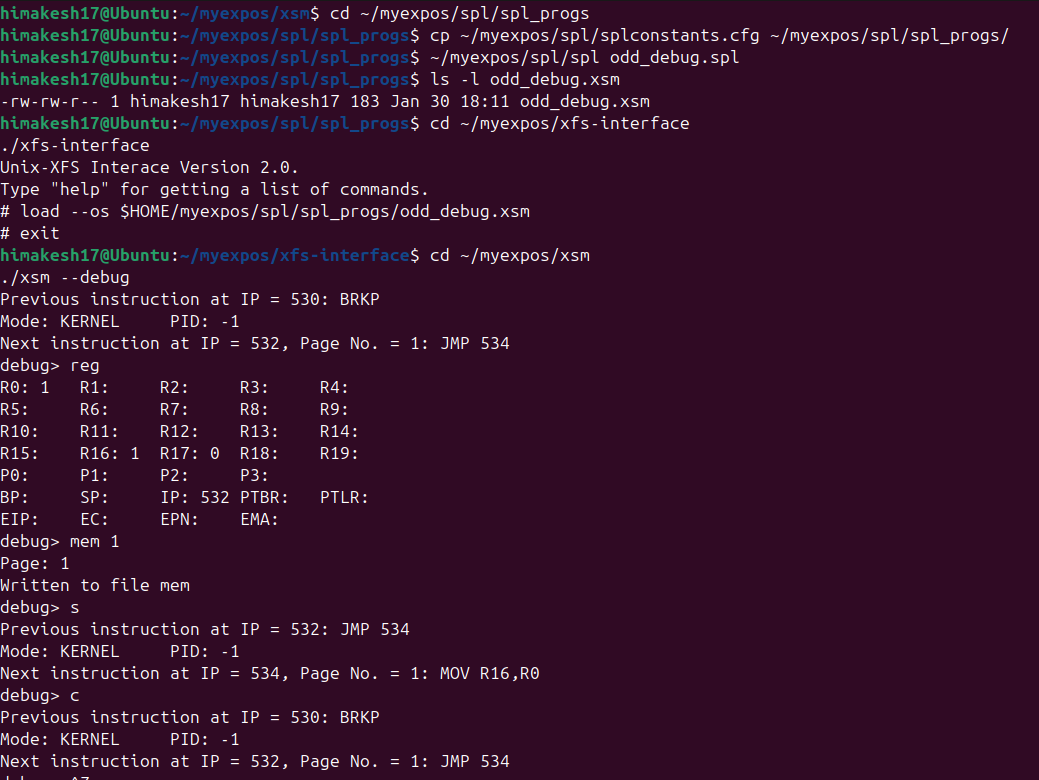


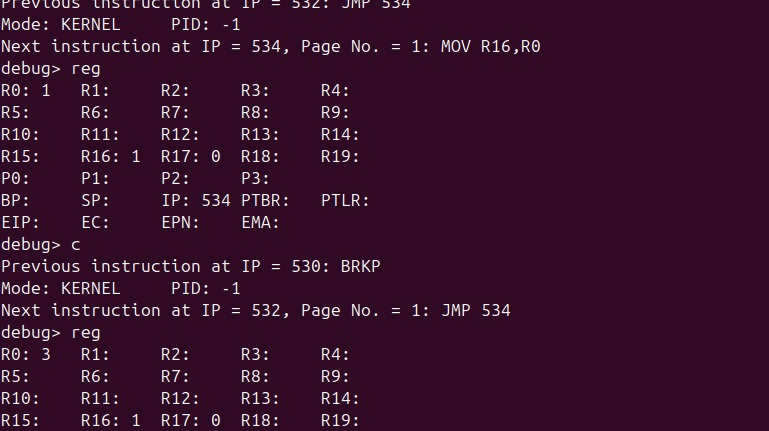
STAGE-5

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



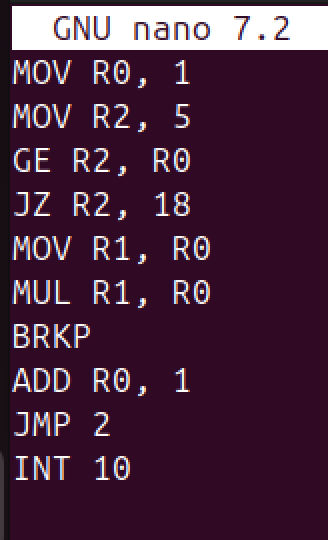
1. Compile the program using the spl compiler
2. Load the compiled xsm code
3. Run the machine in debug mode



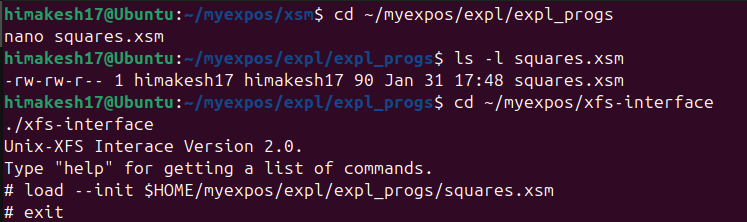


STAGE – 6

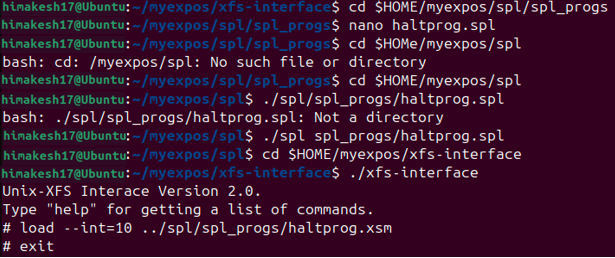
1. The following code illustrates the INIT program used in this stage. It computes squares of first 5 numbers. The value of Register R1 during each iteration will hold the result.

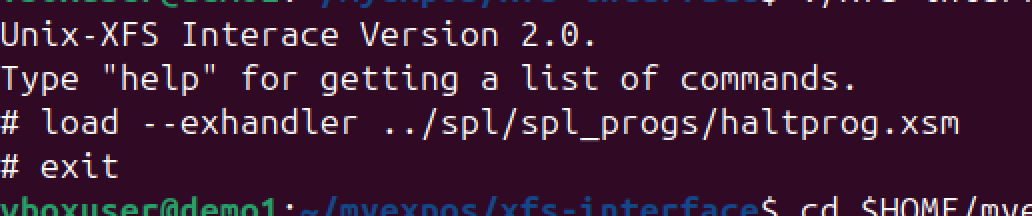


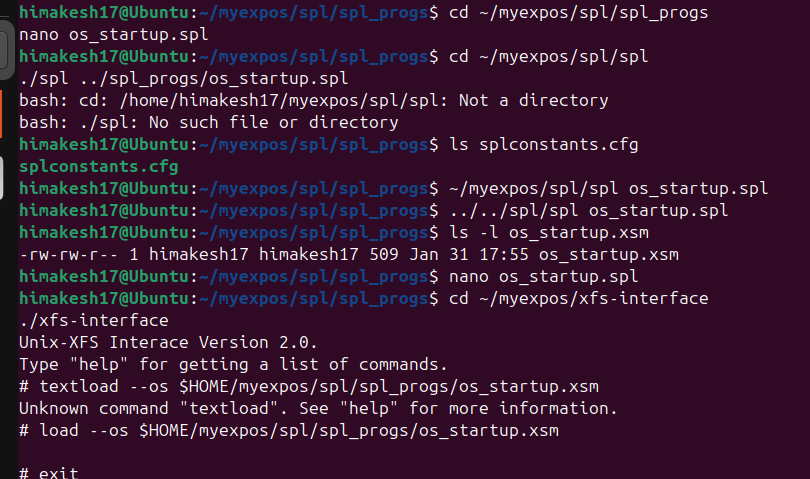
2. Load this file to the XSM disk as the INIT program using XFS interface.# load --init $HOME/myexpos/expl/expl\_progs/squares.xsm



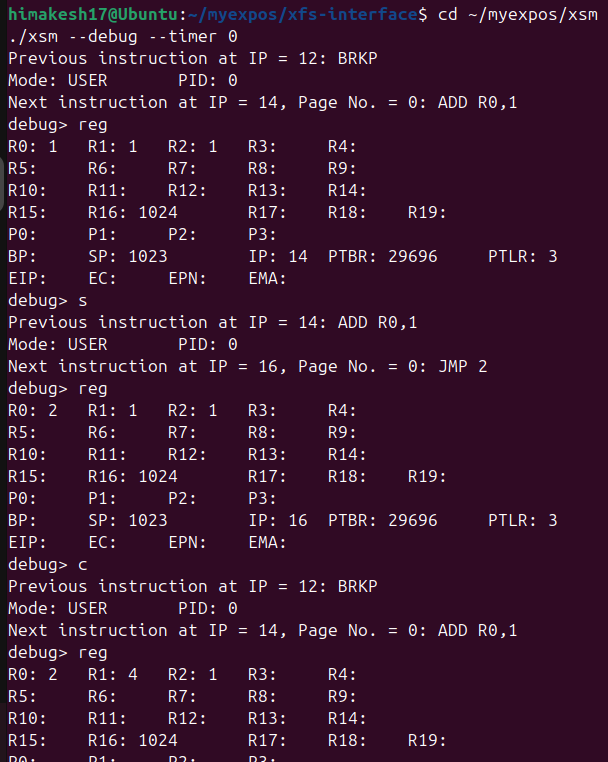
1. Create a file haltprog.spl with a single halt satatment
2. Compile the program
3. Load the compiled code as INT 10 from the xfs-interface

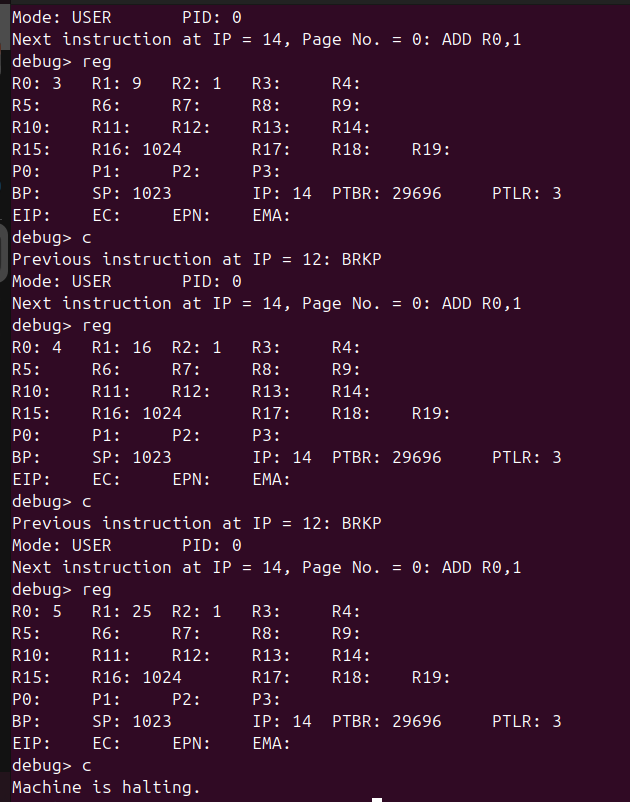






OUTPUT:

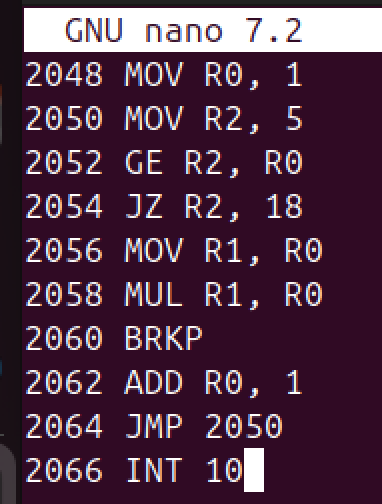




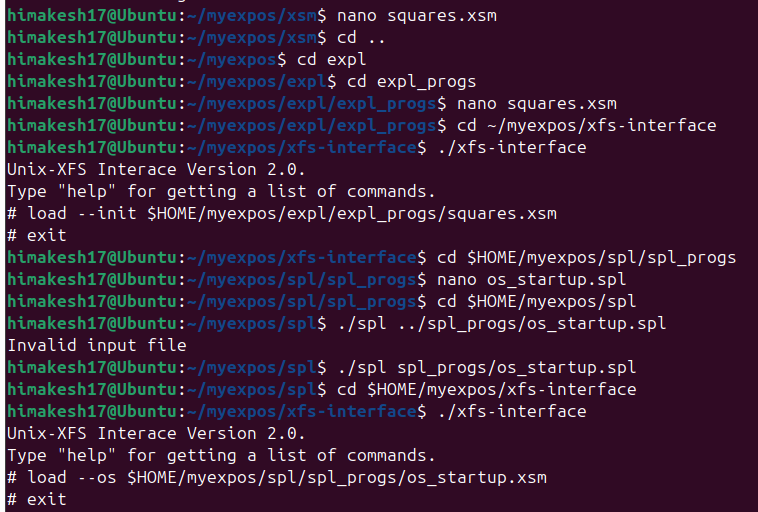
ASSIGNMENT-1

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.

1. Modify user Code



1. Reload INIT into disk



1. Modify 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 = 9;**

**[PTBR+8] = 65;**

**[PTBR+9] = "0100";**

**[PTBR+10] = 66;**

**[PTBR+11] = "0100";**

**[PTBR+16] = 76;**

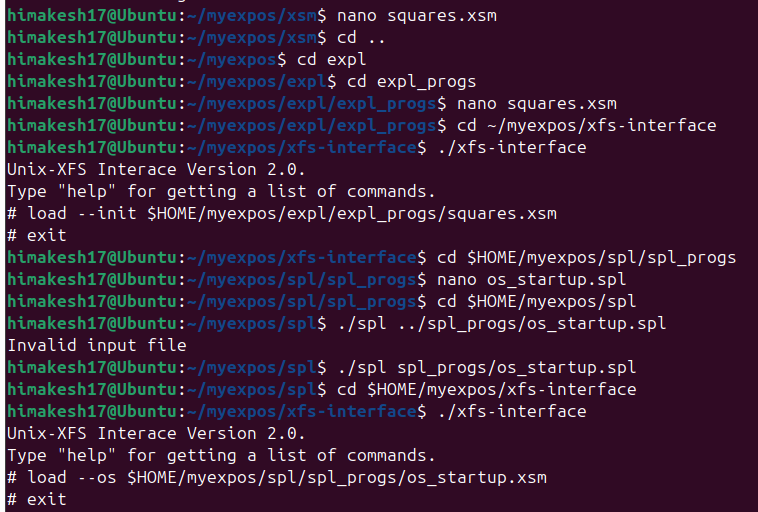
**[PTBR+17] = "0110";**

**[76\*512] = 2048;**

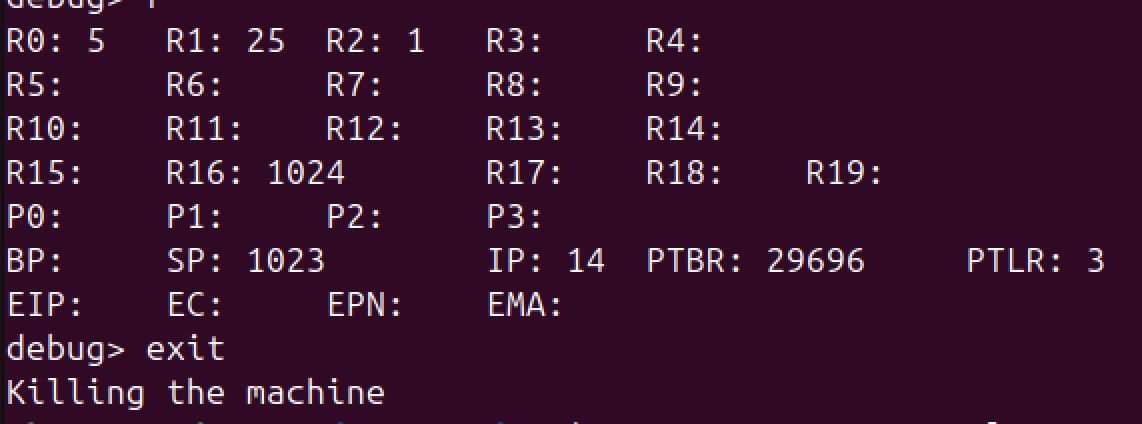
**SP = 76\*512;**

**ireturn;**

1. Recompile OS startup

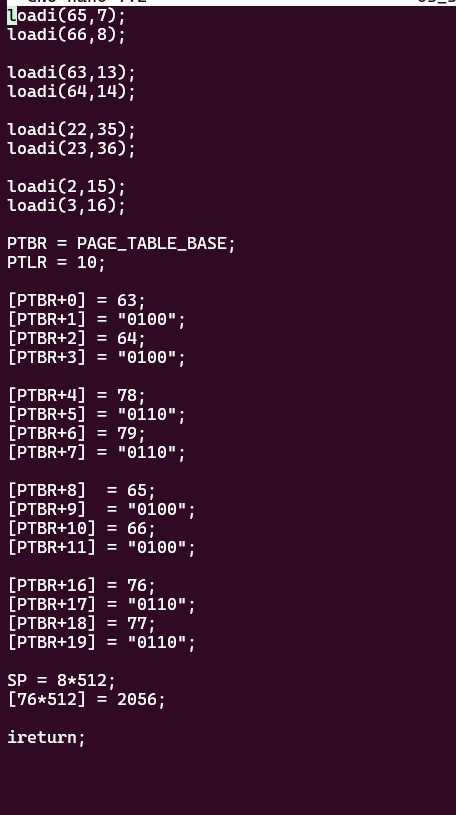


1. Run in Debug

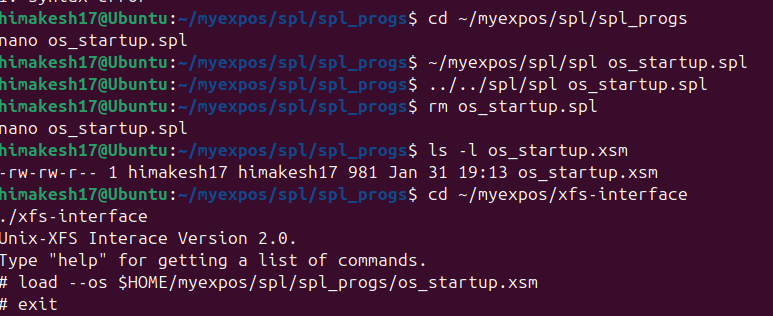


STAGE-7

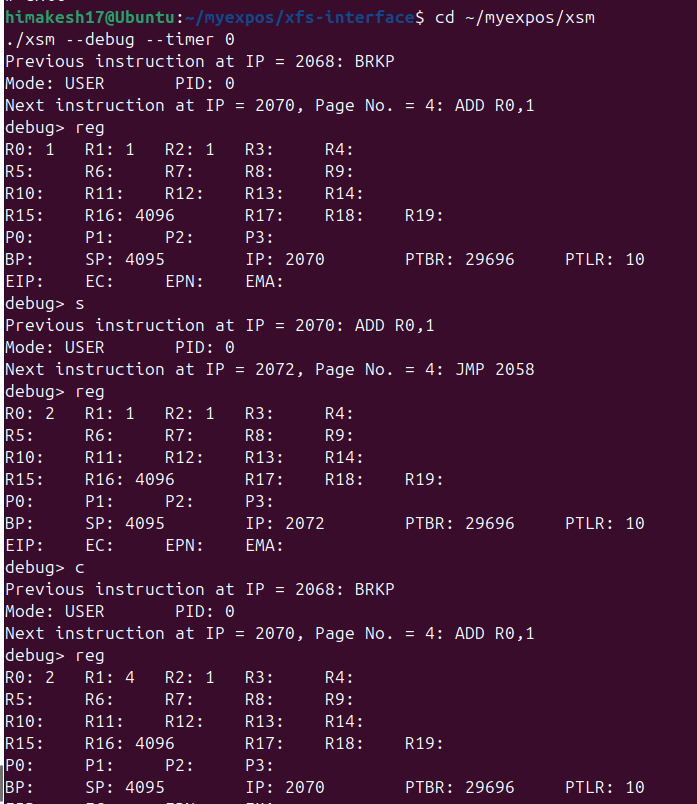
1. Compile and load the modified OS startup code

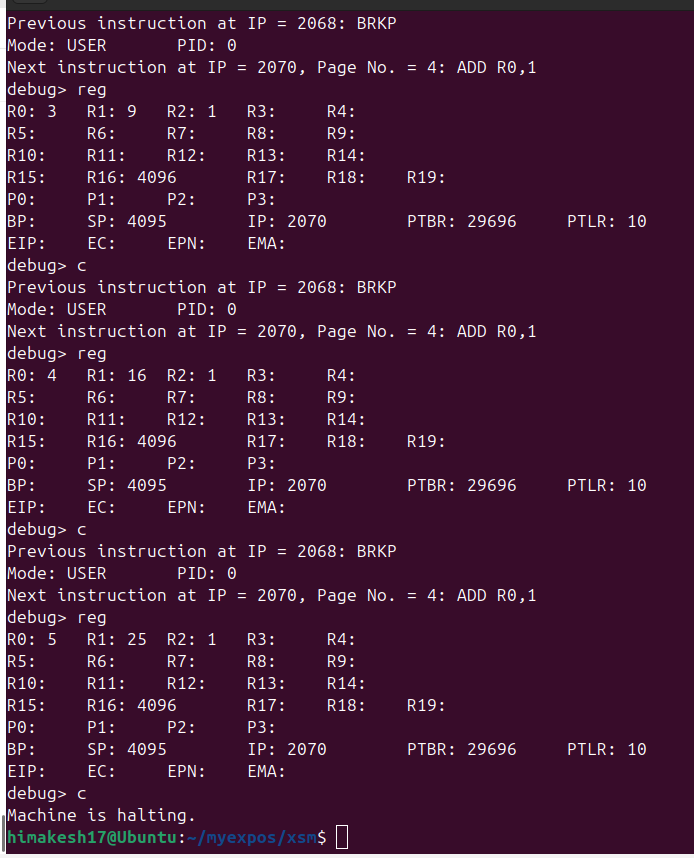


1. Load the modified user program



1. Run the machine in debug Mode

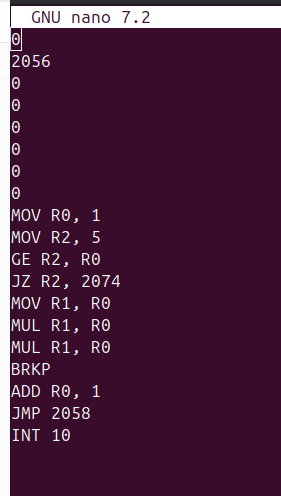




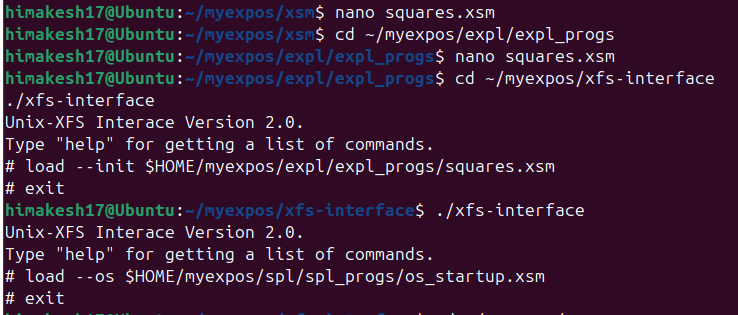
ASSIGNMENT

Change the user program to compute cubes of the first five numbers

1. Modify INIT program



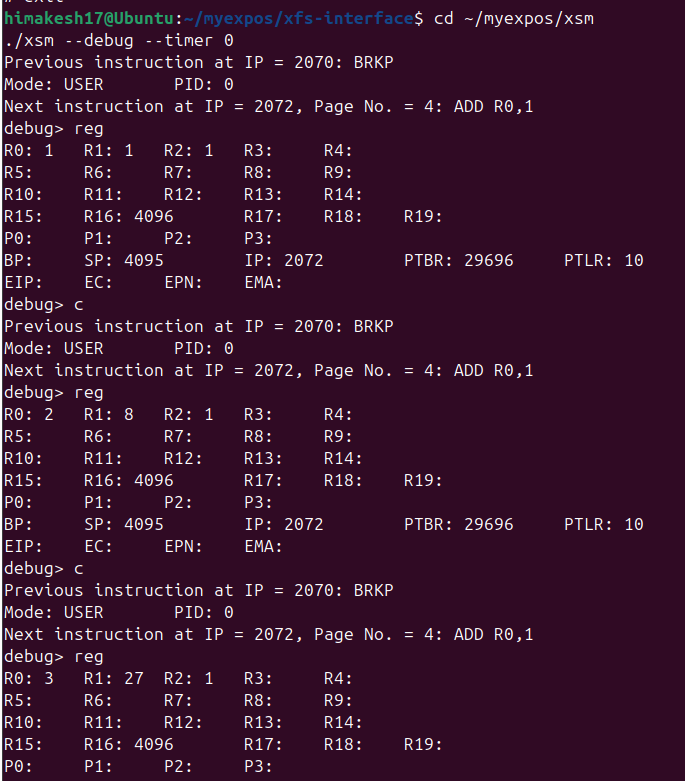
1. Reload INIT program

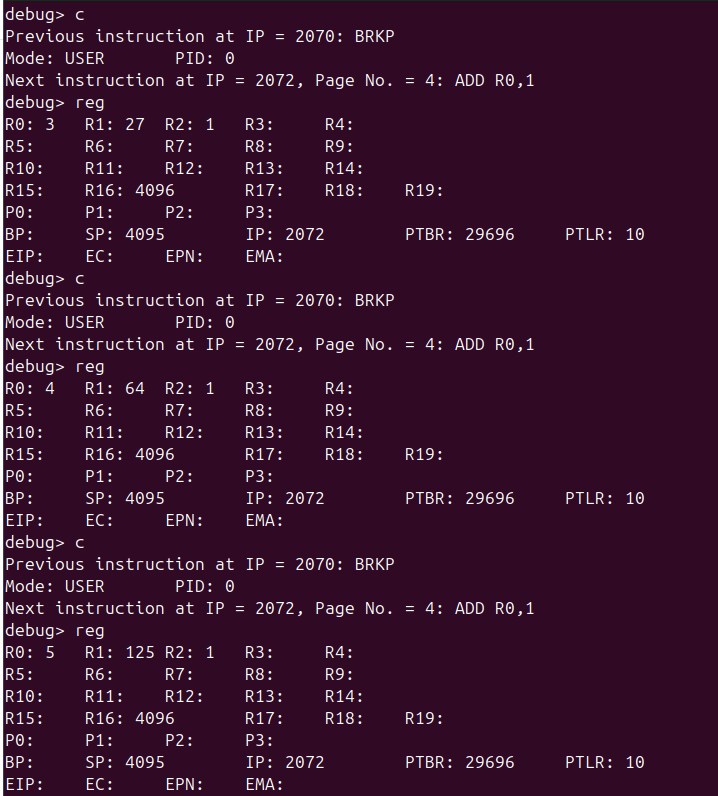


1. Run the program,



OUTPUT



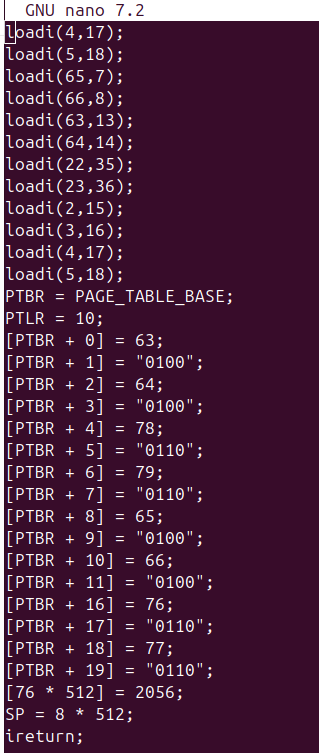


STAGE-8

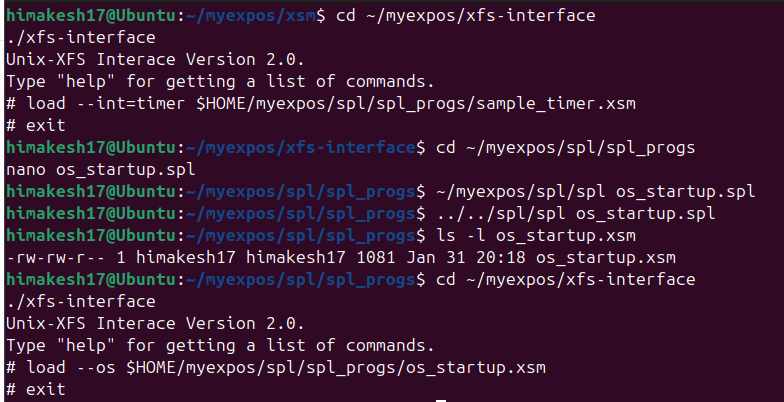
Running XSM *with timer enabled* means:

* The machine will generate a **timer interrupt automatically**
* The interrupt occurs **after a fixed number of instructions**
* Control is transferred to the **timer interrupt handler** in the OS

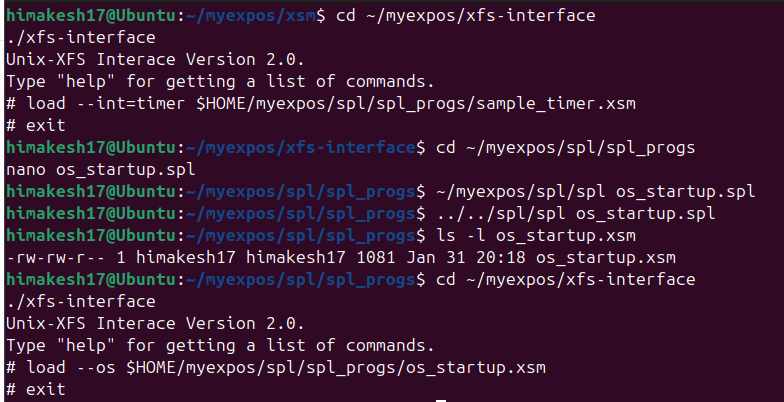
1. OS\_startup sets PTBR for INT
2. Timer interrupt handler upadates ptbr during scheduling
3. Debugging often involves checking PTBR
4. Save this file in your UNIX machine as $HOME/myexpos/spl/spl\_progs/sample\_timer.spl



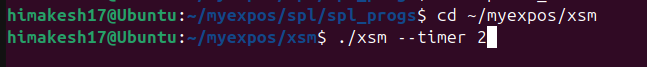
1. Compile the program using the SPL compiler
2. Load the compiled XSM code as the time interrupt into XSM disk



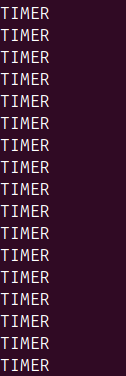
1. Recompile and reload the OS Startup code



1. Run the XSM machine with timer enabled

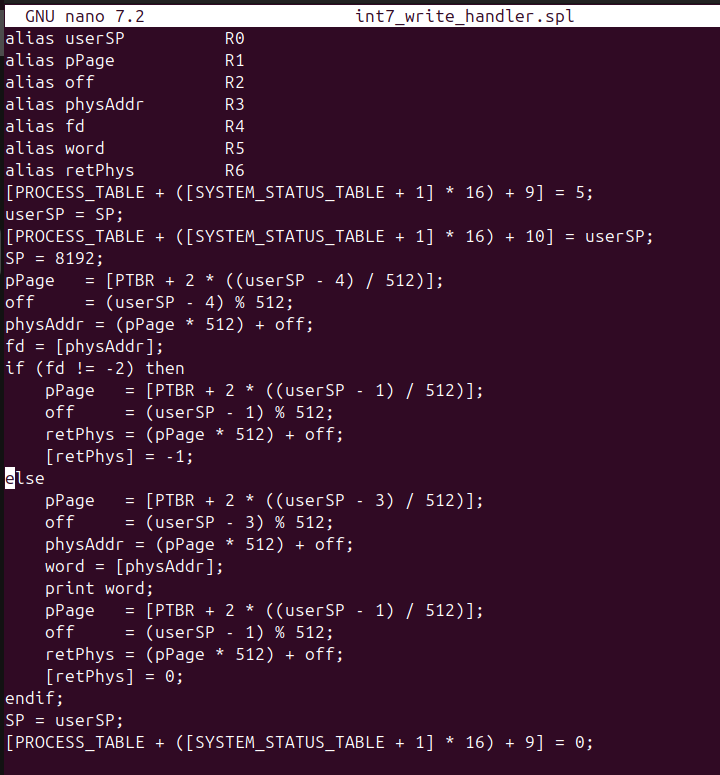


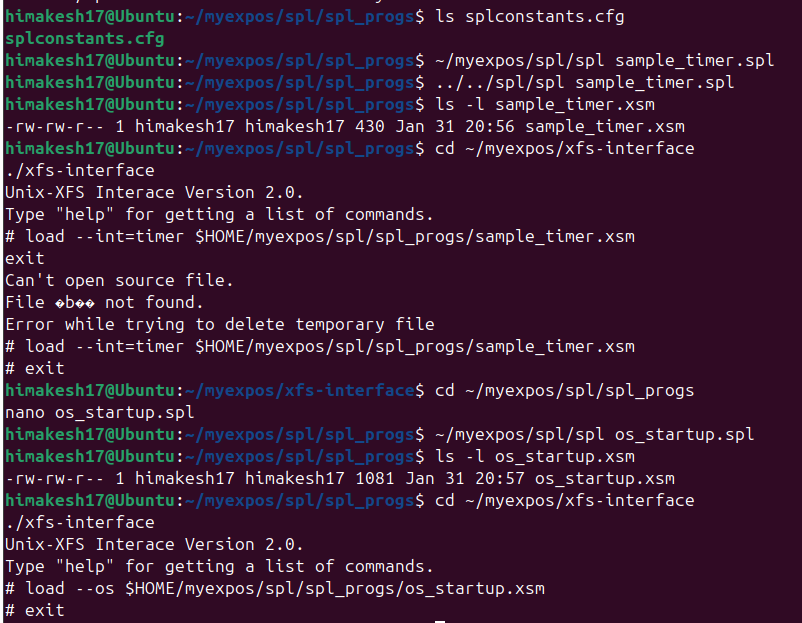
OUTPUT

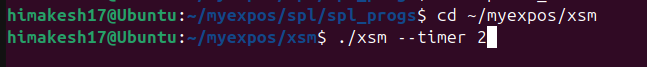


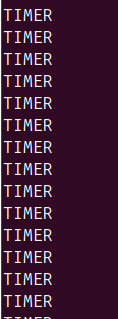
STAGE-9

1. OS STARTUP CODE
2. Allocate user area
3. Set process ID (stored in process Table)
4. Update system status table
5. TIMER INTERRUPT
6. Save user stack pointer
7. Switch to kernel stack
8. Save user register
9. Print message
10. Restore user context
11. Return to user mode



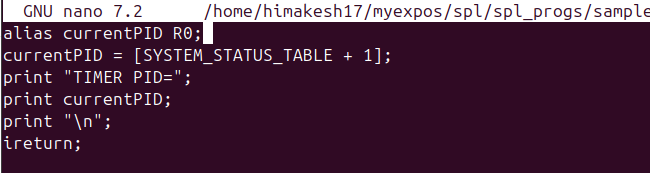


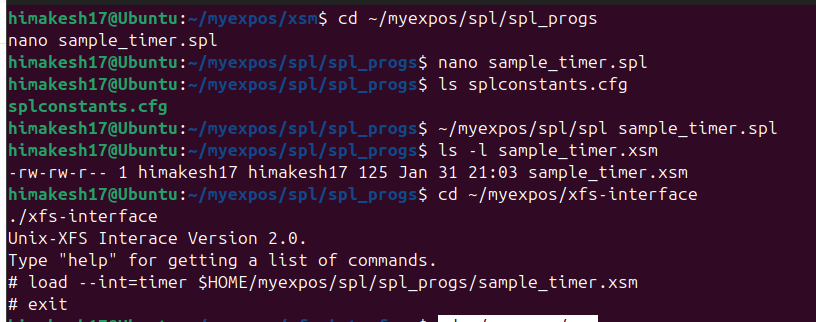


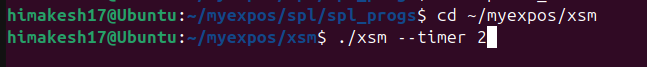


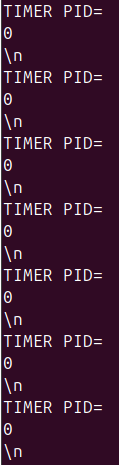
ASSIGNMENT 1

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.







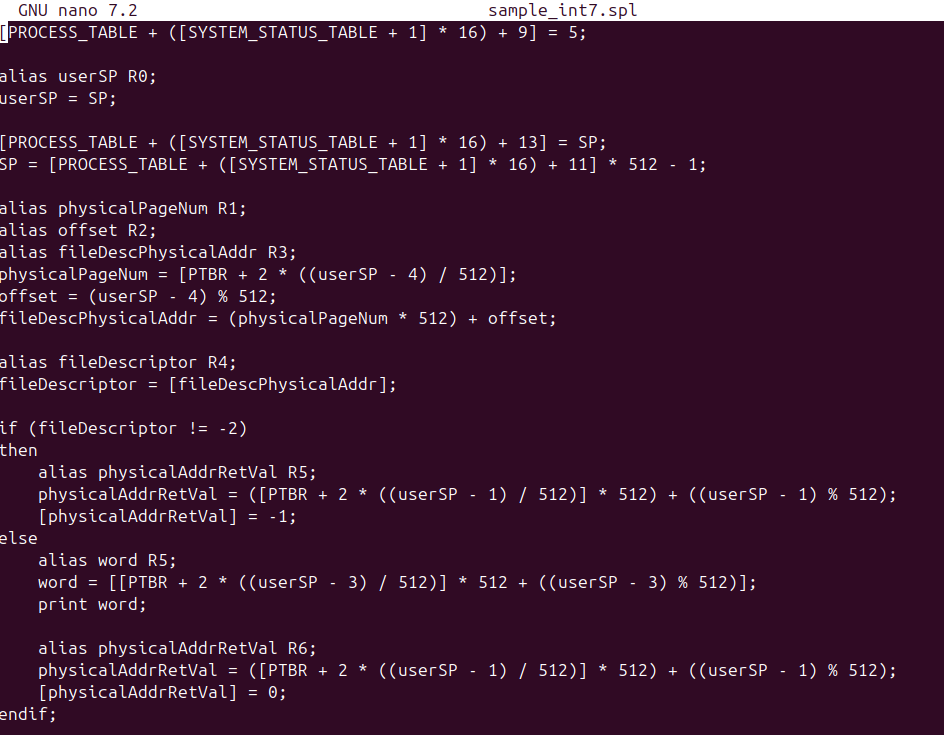


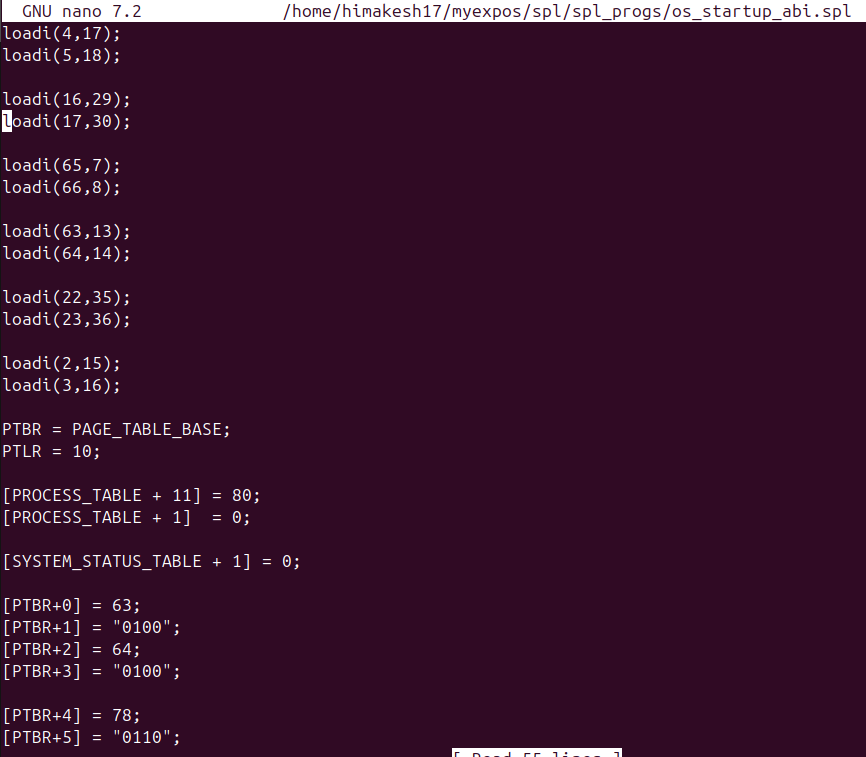
STAGE-10

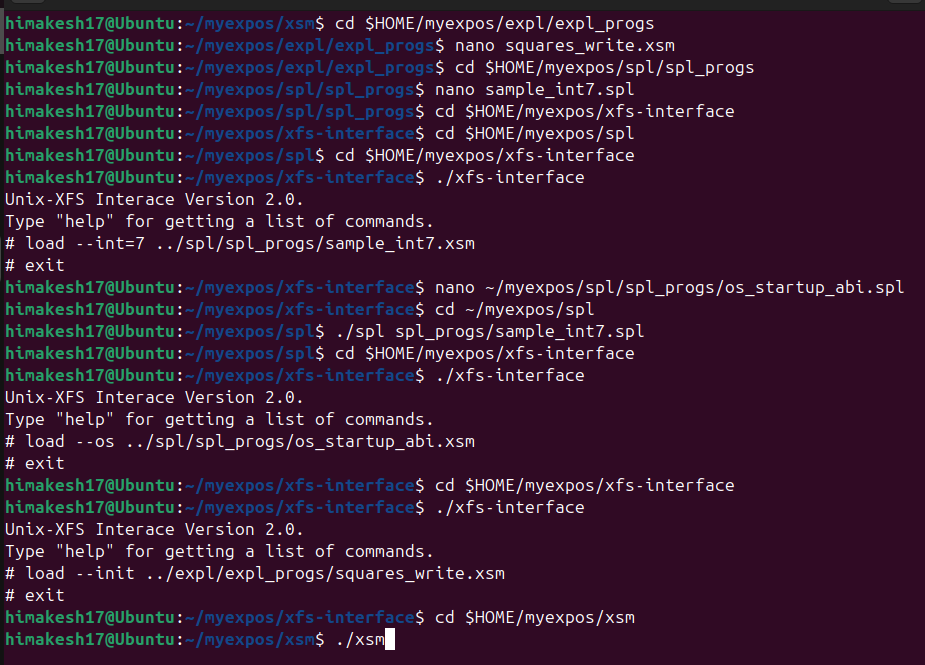
This stage is about **printing output from a user program using a system call instead of debug mode**, and **implementing the kernel-side handler for that system call**.

* 1. The **user program** invokes the **write system call** (system call number **5**) using **INT 7**
  2. The system switches from **user mode to kernel mode**
  3. The **INT 7 handler (kernel code)**:
     + 1. Reads arguments from the **user stack**
       2. Writes the given word to the **terminal**
       3. Stores a return value
  4. Control then returns safely back to the **user program**









OUTPUT:

