Assignment – 4

# COL331 : Operating Systems

## PART A:

### Task 1 :-

Initially before implementing any system call the program would print the lines showing that the pintos started booting and then it identifies our program t1 ans since process_wait is not yet implemented, so it exits from the program and power off the OS.

### Task 2 :-

In syscall.c, pintos provides us 4 syscall (syscall1, syscall2, syscall3, syscall4) macros which are used to generate the assembly code for trapping into the kernel to use. The syscall macros store the function_number and the arguments are pushed onto the stack in reverse order. Ex. The open function uses the macros syscall2 which first pushes the function_number of open onto the stack and then it pushes the argument which we give to the open function.

### Task 3 :-

When we insert an infinite loop in the wait function, at that time the error message of "System call not implemented" is printed confirming the execution of the syscall_handler.

### Task 4 :-

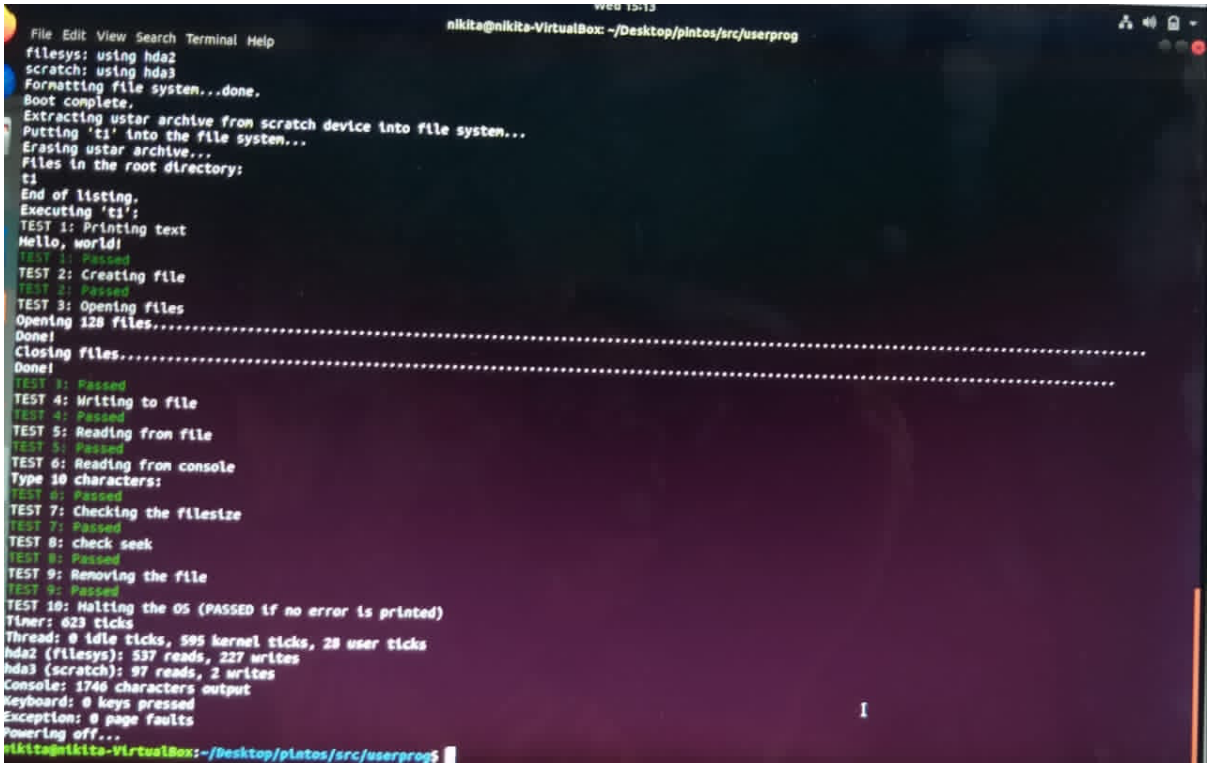
1. **Syscall_handler :**
   The sycall_handler switches the function by looking at the syscall_number of the function which is basically the value stored at the bottom of the stack to which f->esp points. Then to take the arguments of the relevant function I made structs as given in the pdf, so in each struct the id of the system call and the argument of it are stored.
2. **Create :**
   In this at first the char* filename is validated using the already given function validate_user_addr_range() and then file is created using filesys_create function if the addr is valid.
3. **Remove :**
   In this at first the char* filename is validated using the already given function validate_user_addr_range() and then file is removed using filesys_remove function if the addr is valid.
4. **Open :**
   For this parts and some other parts, I have implemented a struct for the file_descriptor which basically stored the fid(file id), file, and list_elem to use the list operations efficiently.
   In the thread data structure we also included a list of file_descriptors as well.
   Now, first the char* file is validated and then an initially file_desc is given space using the function palloc_get_page() and then if we get a valid file_desc then the file_desc should be given the file given as the argument and while giving it the id we will see if the list of the file desciptors is empty or not; if it is empty means this is the first file in this thread hence it is given a value 3, because 0,1,2 are reserved for stdin, stdout, stderr and finally this id is returned
5. **Close :**
   It searches in the file descriptor list for the file_desc object having the same id as given in the arguments and then if closes the file of the object using file_close and removes that file descriptor from the list.
6. **Filesize :**
   It searches in the file descriptor list for the file_desc object having the same id as given in the arguments and then finds the size of the file of that object using file_length.
7. **Read :**
   It validates the buffer and then checks if the fd corresponds to stdin or not, if it is yes then it reads from the stdin else it reads from the file using file_read and return the number of characters read.
8. **Write :**
   It validates the buffer and then checks if the fd corresponds to stdout or not, if it is yes then it writes on the stdout else it writes on the file using file_write and return the number of characters wrote.

9. **Seek :**
   It searches in the file descriptor list for the file_desc object having the same id as given in the arguments and then seeks the position of the file of that object using file_seek.
10. **Halt :**
   It shutdown the OS using the function in shutdown.h

## Task 5 :-

The test are written for seek, remove, filesize and halt in testsycalls.c

**Final output :-**




## Submitted By :-

**Nikita Bhamu**
**2018CS50413**