Design Document

# User Programs

# CC – 109 – OS Lab

# Group

Ananth Murthy [ananth.murthy@iiitb.org](mailto:ananth.murthy@iiitb.org)

# Preliminaries

While I did not work with my assigned group members, I did co-ordinate with some of my fellow classmates: Chandan Yeswanth, Kartik S and Yashvanth Kondi.

## References

For references, apart from the course material, we referred to the following online repositories:

<https://code.google.com/p/pintosof4p/source/browse/trunk/pintos/src/userprog/syscall.c>

<https://github.com/Jenso/PintOS/blob/master/src/userprog/syscall.c>

<https://github.com/rtwilson/Pintos-Project-2/blob/master/src/userprog/syscall.c>

## Wait

process\_wait() was implemented trivially by inserting an infinite loop in the function, as suggested in the ‘Suggested Order of Implementation’.

## Argument Passing

Argument passing was implemented by setting PHYS\_BASE to PHYS\_BASE – 12, also as suggested in the ‘Suggested Order of Implementation’. This is a workaround to make sure page fault errors do not occur. However, this does not let a program actually use the command line arguments passed to it. In other words, ‘echo’ command on shell cannot actually print the argument passed to it.

# System Calls

Four system calls were handled, READ, WRITE, EXEC and EXIT; as these were needed to make the most minimal shell work. The ‘echo.c’ and ‘shell.c’ programs given in the examples folder were used. ‘echo.c’ was modified to act as a simple ‘Hello World’ program since argument passing was not implemented.

## Data Structures

No new data structures were defined, nor were any changed. No global or static variables were added.

## Algorithms

After extracting the ‘syscall number’ from the stack (from f->esp) and using switch(),

### Read

For reading user input, the input\_getc() function was used, and a single character value was retuned by writing it into the f->eax ( eax register, for the interrupt frame).

### Write

The file descriptor, size of buffer and pointer to buffer were read from the f->esp (stack pointer for the interrupt frame). Write has been implemented only for file descriptor 1, i.e. standard output. In this case, write can be implemented with either putbuf()or by using printf()(defined in kernel). I have used printf(), since putbuf()cannot handle escape sequences and special characters.

### Exec

The exec()system call has been implemented by calling process\_execute(), with the required arguments.

### Exit

The exit()system call has been implemented by calling thread\_exit().

## Synchronization

No new code has been added for synchronization. Thus, the output of an exec()call in the shell program (i.e the return status) sometimes gets printed to the screen before the output of a user program which was called in the exec().

## Rationale

The above design decisions were made under severe time constraints due to the presence of consecutive project demos for several other courses. Thus, most functionality has been implemented in the quickest way possible.

Thus, it is quite possible that the implementations maybe incomplete, and may not handle all the error cases.

# Survey Questions / Feedback