Documentation----OS Project
Submitted By - Dushyant Goyal, Bhavya Agarwal

We have implemented the following in our project for Operating System -
1. Page Tables using Self-referencing
2. fork() using COW
3. malloc() - no. of bytes taken as input
4. tarfs - open(file name), read(file descriptor), close(file descriptor), opendir(directoryname), readdir(DIR structure), closedir(Dir structure)
5. read(file descriptor, no.of bytes/formatstring, write destination), write(string, no. of bytes, file descriptor)
6. ls, ps, sleep, sh
7. getpid()
8. wait()
9. waitpid()
10. exit()
11. exec()
12. execvp()
13. printf()
14. scanf()
15. sleep()
16. free()
17. #! (parser in libc) works independently but not with SHELL

Some of the test binaries are in bin/ folder which can be run from the SHELL

The starting point of the OS will be the SHELL.

SHELL with PATH, cd, ulimit(we are setting the time for execution limit)

Testing -

Example 1:
[$] bin/test_sleep --- this will create two processes (parent will wait for the child to finish
[$] bin/ps --- this will show the running processes

Example 2:
[$] bin/test_malloc

Example 3:
[$] bin/test_scanf --- binary for testing the scanf

Example 4:
[$] bin/test_segv --- binary for testing the SEGV

Example 5:
[$] bin/clear --- binary for testing the clear

Example 6:
[$] bin/test_fork --- binary for testing test_fork

Example 7:
[$] bin/test_fork_time --- binary for testing test_fork_with timer

Example 8:
[$] bin/ls path--- binary for testing ls . Also tests the readdir,opendir and close

Note- Some of the binaries work when executed independently as a process from the init_process() function in the task_management.c file but breaks when executed from the SHELL;

This is the list of libc files

1. dir.c - it has implementations of readdir, opendir, closedir, open, close, read system calls
2. malloc.c - it has malloc, yield, fork, execve, execvp, getpid, waitpid, wait, sleep, ps_list, kill, clearup and free system calls
3. exit.c - this file has exit system call
4. printf.c - this file breaks the stdout and stderr string in pieces and calls the write system call to print the final string.
5. scanf.c - interface for read system call in order to read from a file or stdin
6. shbang.c - parser for #! files.

List of kernel files -

1. task_management.c - implementation of doFork(), doExec(), doExecvp(), Queues implementation (RunnableQ, WaitQ, DeadQ, AllQ), wait(), waitpid(), sleep(), kill(), getpid(), checkAwake()- checks whether a sleep process is awake,

2. v_mem_manager.c - implementation of COW, deletePageTables, copyPageTables, mapPageTables, createUserStack(), copyUserStack, mmap()

3. task_schedule.c - implementation of scheduler

4. irq.c - implementation of timer interrupt , preemptive based scheduling,
5. isr.c - implementation of INT 80, INT 13, INT 14, INT 0

6. phy_mem.c - implementation of physical pages allocation, free_phy_page(), allocate_free_phy_page()

7. page_table.c - self-referencing based page tables traversal functions

8. tarfs.c - backend for all the tarfs related calls like open, read, close and the parsing of elf files and filling up the pcb with relevant data.

9. dir.c - kernel implementations for opendir, readdir, closedir

10. printf.c - implementation of write system calls

11. irq.c - keyboard and timer interrupts implementation

12. scanf.c - implementation for stdin system calls