**OS Project Demo 2**

**Team 09- Kiran Hiranandani, Akshay Shinde**

1. **Does xv6 kernel use cooperative approach or non-cooperative approach to gain control while a user process is running? Explain how xv6’s approach works using xv6’s code.**

* xv6 kernel uses non-cooperative approach for gaining control when a user process is in running stage.
* Problem with cooperative approach is that it runs in an infinite loop and the only way to get back to normal is to reboot the whole system.
* There are 3 processes which are performed by the CPU in a loop -
* Reading all the instructions.
* Adding or incrementing the program counter.
* Repeating the above 2 steps.
* The kernel handles these events in a non-destructive manner by using a Timer Interrupt implemented in the *trap.c* file.
* At this point, kernel has gained control of the CPU and does the following :
  1. Saving the processor’s registers for resuming in the future.
  2. Switching to the kernel mode
  3. Using the interrupt handlers
  4. Return the control to the scheduler for normal handling.Thus, this hardware feature is essential in helping the kernel maintain control of the machine.

The xv6 code is used to enable interrupts : sti();

2. **After fork() is called, why does the parent process run before the child process in most of the cases? In what scenario will the child process run before the parent process after fork()?**

The parent process runs before the child process in most of the cases is because :

* fork() system call is called by parent process before the child process. fork() is a system call which saves the current user context from resuming back.
* When the *fork()* system call is done the system reverts to user mode and switches the context of the current process. i.e. the parent process which called the *fork()*.
* Hence the parent resumes to operate making itself run instead of the child process most of the times.We can make the child process run first by setting the child process state to running after being forked by the parent process and calling yield function which basically gives up the CPU for one scheduling round and will cause the child process to run first always.
* The child will run before the parent in these scenarios –

The parent yields a cycle by calling the function *yield()*

3. **When the scheduler de-schedules an old process and schedules a new process, it saves the context (i.e., the CPU registers) of the old process and load the context of the new process. Show the code which performs these context saving/loading operations. Show how this piece of code is reached when saving the old process’s and loading the new process’s context.**

* Below is the code for saving the context of the old process and loading it to the new process:

p = highprio;

c->proc = p;

switchuvm(p);

p->state = RUNNING;

swtch(&(c->scheduler), p->context);

switchkvm();

* Swtch saves the current registers and loads the saved registers of the target kernel thread into the x86 hardware registers, including the stack pointer and instruction pointer.