Project 2

Group no. 20
1.Sai Milind Tammisetti
2. Nikita Kokne

Q1. 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.

Sol: XV6 kernel makes use of non co-operative approach. The problem with co-operative approach is that once the process enters into an infinite loop the only way to resort back to normal state is that you have to reboot your system. Whereas in non-co operative approach, a timer device is programmed to raise an interrupt every so many milliseconds; when the interrupt is raised, the currently running process is halted, and a pre-configured interrupt handler in the kernel runs. At this point, the kernel has regained control of the CPU, and thus can do what it pleases; stop the current process, and start a different one. The addition of a timer interrupt also gives the kernel the ability to run again on a CPU even if processes act in a non-cooperative fashion. Thus, this hardware feature is essential in helping the kernel maintain control of the machine.

The following part of XV6 code is used to enable interrupts,
sti();

Q2. 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()?

Sol: When fork is called the parent process is currently in the running state and the parent process creates a new process (child process) structure by calling allocproc() and fills it with the values of the original process(parent process value) and maps the same page tables. So the scheduler basically always runs the parent process first as it is in the running state and the newly created child process is set to runnable.

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(the one where parent is running) and will cause the child process to run first always.

Q3. 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.

Sol: The following piece of code in the scheduler function of proc.c is responsible for saving the context of old process and loading the context of new process,

```
p = hpr;
c->proc = p;
switchuvm(p);
p->state = RUNNING;
swtch(&(c->scheduler), p->context);
switchkvm();
```

switchuvm sets up a task state segment SEG_TSS that instructs the hardware execute system calls and interrupts on the process's kernel stack. Swtch saves the current registers and loads the saved registers of the target kernel thread (proc->context) into the x86 hardware registers, including the stack pointer and instruction pointer.