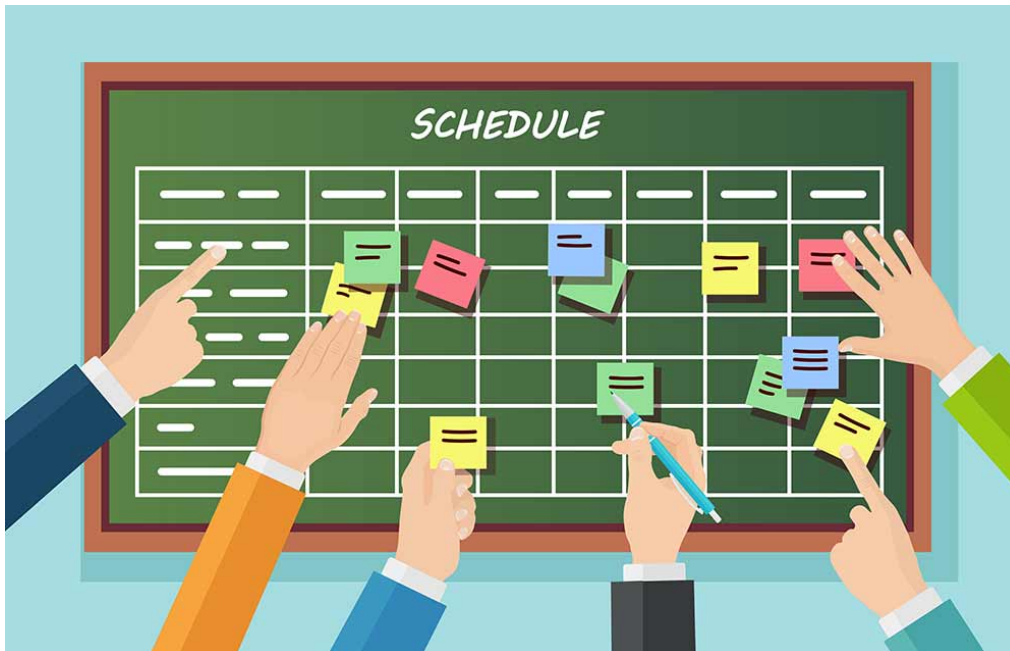


FIXED PRIORITY PREEMPTIVE SCHEDULER

<https://github.com/DevSheth/SchedulINxv6>



Sheth Dev Yashpal
Michael Mervin

Team 19 : Operators
CS 3500

Problem Statement

- The task is to implement a fixed priority scheduler for xv6. Currently it only maintains an array of processes and schedules any Runnable process while constantly iterating.
- First task is to define a prio value in the proc structure in order to keep track of the process's priority level. Also, the scheduler should now schedule processes as per the priorities defined.
- Second task is to enable users to get the prio value for a process and also set it to a desired value if needed.
- The last task is to implement a feedback function in order to dynamically change priorities for processes based on their runtime.

Solution Design

- For the first task we just have to find the proc structure in the xv6 implementation and modify it to include prio.
- Next we change the implementation of the scheduler function to support priority based scheduling.
- We need to add syscalls for users to change the kernel state variables and also yield on receiving a setpriority() call.
- For the last part, in order to implement the feedback function we need to maintain the time for which each process runs and use that to dynamically update the priority.
- The scheduler function on every iteration calls the context switch function to switch to the scheduled process in the user space. Once the process is done it returns back to the same point. We can use this to time the execution time of the scheduled process and hence update its priority. We can use the inbuilt time function cmos_read() function for timing.

What's Done

- We have updated the proc structure to contain the attribute prio, which signifies the current priority of the process.
- We have updated the scheduler to choose to run the process with the highest priority.
- The system call `getpriority()` which returns the priority of the calling process has been implemented. The `setpriority()` call has also been implemented however it's having some bug which we need to fix relating to underflow and overflow.

DESIGN APPROACH (1/2)

Proc.h

1)The proc structure is modified to contain **prio** attribute.

prio values are modified and used in the functions of proc.c

Proc.c

1)The allocproc() function is modified to initialise **prio** attribute of newly created process

2)The scheduler is modified to update **prio** values based on running time and also chooses the process with minimum **prio** value

3)Functions **SET_prio** and **GET_prio** which set the priority values and obtain the priority value of the calling process.

Syscalls.c,make file etc.

1)System calls **setprio** and **getprio** have been added.

The sys calls getprio and setprio directly call functions **GET_prio** and **SET_prio**.

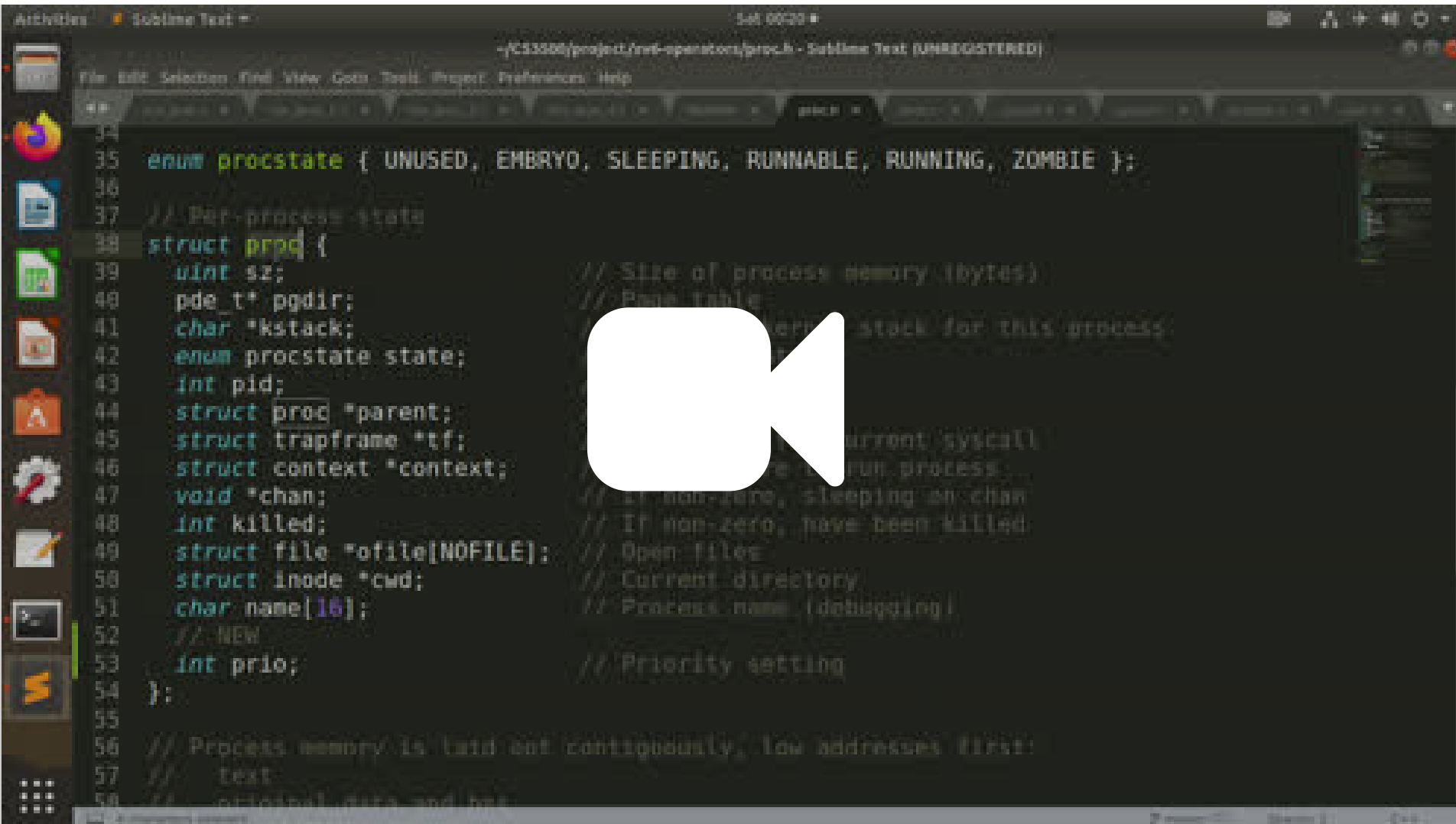
DESIGN APPROACH (2/2)

Level	Status	Approach
1	Completed	The proc structure is updated to contain the attribute prio, which signifies the current priority of the process. The for loop in the scheduler is also modified to choose the process with highest priority (lowest prio value).
2.	Completed	The system call getpriority has been created to return the prio attribute of the calling process. The system call set priority has also been created which sets the prio attribute of the calling process and also calls the sched() function which causes rescheduling.
3.	Completed	The scheduler now maintains the clock ticks before a process has been scheduled and the clock ticks after control returns to the scheduler. If the difference between these quantities is zero then priority of the current process is increased (prio value reduced), if not the priority of the current process is reduced.

MOST IMPORTANT CHALLENGE

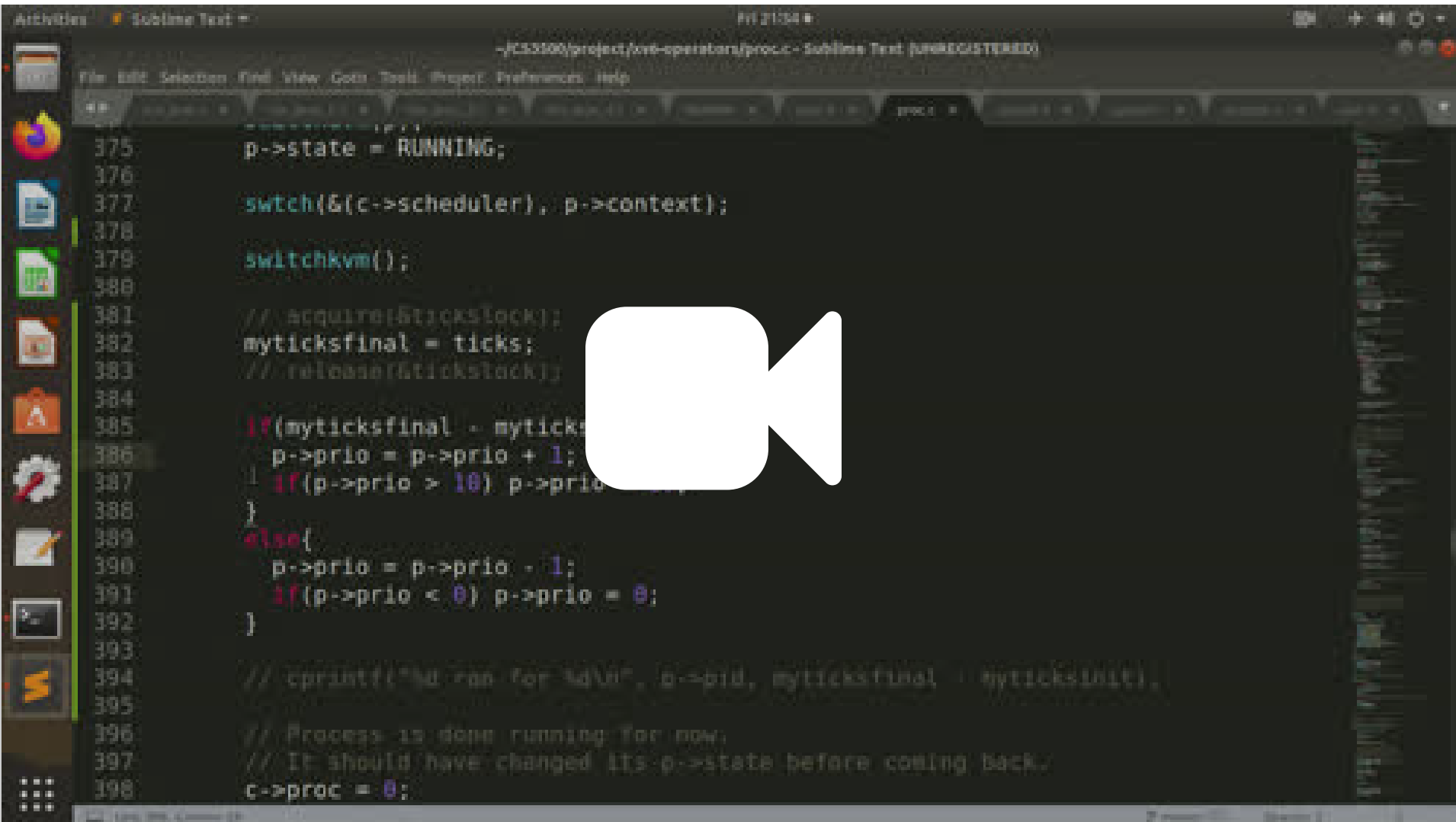
- Creating user space programs to demonstrate the working of the scheduler was challenging, the difficulty was mainly due to timer interrupts after every clock tick. This makes the behavior of the processes to be slightly unpredictable since the some other process maybe scheduled even without there being any explicit `yield()` or `setpriority()` calls.
- Understanding the exact functionality of the scheduler and other functions for managing the processes was also challenging.
- Also to figure out threshold for the processes to implement the feedback was a challenge. We initially thought we should use `cmos` function but then resorted to ticks and for that too we have to build multiple programs and test and finalize the threshold.

CODE DEMO - 1



```
34
35 enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37 // Per-process state
38 struct prpq {
39     uint sz; // Size of process memory (bytes)
40     pde_t* pgdir; // Page table
41     char *kstack; // Kernel stack for this process
42     enum procstate state;
43     int pid;
44     struct proc *parent;
45     struct trapframe *tf; // Current syscall
46     struct context *context; // Context to run process
47     void *chan; // If non-zero, sleeping on chan
48     int killed; // If non-zero, have been killed
49     struct file *ofile[NOFILE]; // Open files
50     struct inode *cwd; // Current directory
51     char name[16]; // Process name (debugging)
52     // NEW
53     int prio; // Priority setting
54 };
55
56 // Process memory is laid out contiguously, low addresses first:
57 //   text
58 //   original data and bss
```

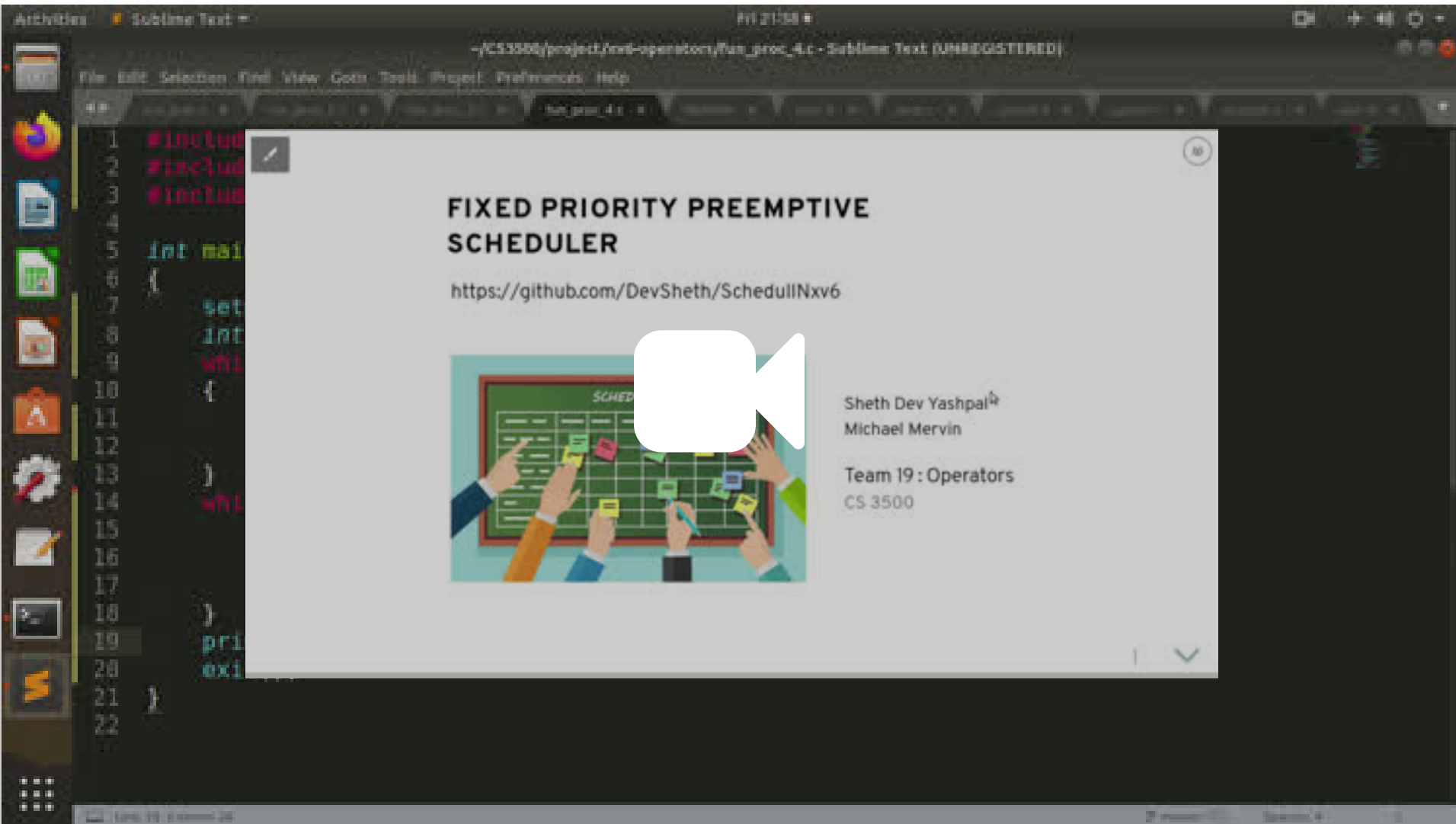

CODE DEMO - 2



The screenshot shows a Sublime Text editor window with a dark theme. The title bar indicates the file path is `~/CS3500/project/kvm-operators/proc.c - Sublime Text (UNREGISTERED)`. The code is C language, showing a switch statement for `c->scheduler` and a `switchkvm()` call. It includes comments about acquiring and releasing a tick lock, and adjusting process priority. A large white video camera icon is centered over the code.

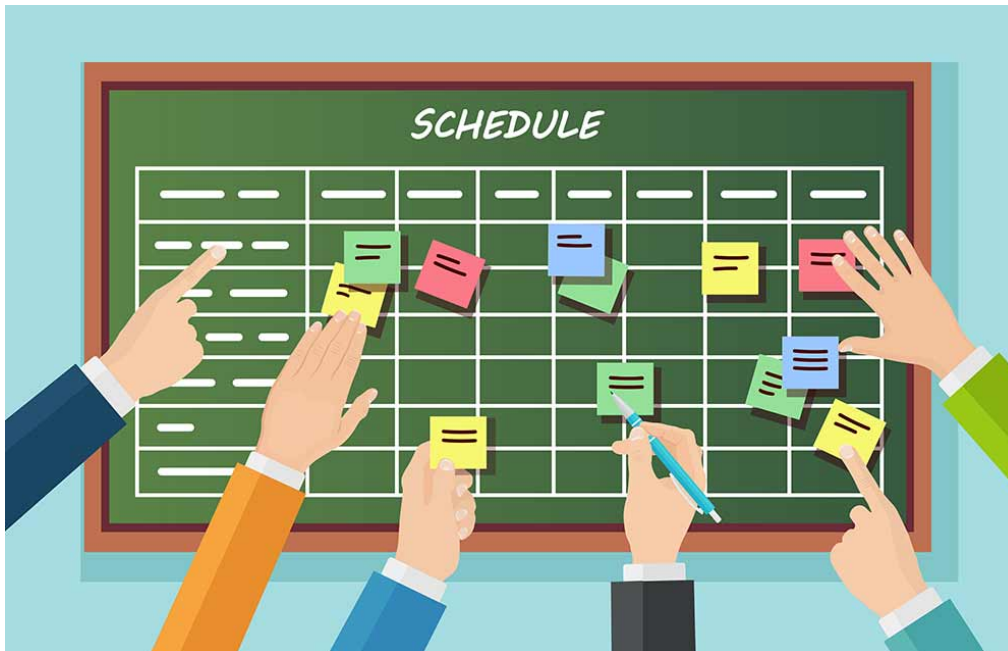
```
375     p->state = RUNNING;
376
377     switch(&(c->scheduler), p->context);
378
379     switchkvm();
380
381     // acquire(&tickslock);
382     myticksfinal = ticks;
383     // release(&tickslock);
384
385     if(myticksfinal - myticks
386         p->prio = p->prio + 1;
387     | if(p->prio > 10) p->prio = 10;
388     }
389     else{
390         p->prio = p->prio - 1;
391         if(p->prio < 0) p->prio = 0;
392     }
393
394     // cprintf("%d ran for %d\n", p->pid, myticksfinal - myticksinit);
395
396     // Process is done running for now;
397     // It should have changed its p->state before coming back.
398     c->proc = 0;
```

CODE DEMO - 3



"The key is not to prioritize what's on your schedule,
but to schedule your priorities."

- Stephen Covey



Sheth Dev Yashpal

Michael Mervin

Team 19: Operators

CS 3500