# Midterm Discussion

CSCE 313 Spring 2018

Tanzir Ahmed

# True/False Questions

1. Some parts of the Operating System (e.g., utilities, libraries) run in User Mode, while other parts in Kernel Mode
2. To save User programs and the Kernel from each other's malicious/buggy access, memory protection is always performed – both in User mode and Kernel mode
3. CPU checks for Interrupts after executing every instruction
4. A process is kicked out voluntarily for System calls and involuntarily for Interrupts
5. Disabling Interrupts means that Interrupts are deferred instead of completely blocked
6. Each Interrupt comes with a number and the CPU uses that number as index into the Interrupt Vector Table
7. Handling Interrupt is an example of User to Kernel mode switch
8. Context-switch overhead is an important OS performance metric
9. A parent process and its child processes share the same address spaces
10. Because of Direct Memory Access (DMA), we can overlap CPU and I/O operations

# TRUE/FALSE Questions

11. The concept of Multi-programming does not work without DMA
12. A system call for an I/O operation only returns (i.e., back to user) when DMA generates an interrupt to indicate I/O finish
13. While performing system call, the requesting process waits in the I/O device queue
14. Once I/O finishes, the waiting process goes directly from “blocked” to “running” state
15. Each I/O device has a separate wait queue
16. In Multi-level Feedback Queue (MFQ), the highest priority queue has the smallest Round-Robin (RR) time quantum
17. CPU-bound processes stay high up among the MFQ levels
18. Small RR quantum is good for I/O-bound processes, while large quantum is good for CPU bound processes
19. Mid-term scheduler acts under memory pressure to evict some waiting processes to disk
20. Response time is a user-centric performance metric while throughput is a system-wide metric

# TRUE/FALSE Questions

21. Virtual Memory system uses page fault as a mechanism to make memory available to users on demand
22. Being in the User mode, the CPU ignores a Kernel mode instructions and continue onto the next one
23. RR scheduler is the worst when all jobs are equal
24. While handling Interrupts in a multi-CPU system, Interrupts are disabled in all CPUs
25. Manipulating page table entries is an example of privileged operation done in Kernel

# Short Questions

26. [5 pts] What is output of the following program assuming the file "foobar.txt" contains "foobar":

```
char buf[512]={0}; // buffer initialized to all NULL
int fd = open ("foobar.txt", O_RDONLY); // foobar.txt contains "foobar"
int nbytes = read(fd, buf, 512);
cout << "NB = " << nbytes << ",  BUF = " << buf << endl;
```

NB = 6, BUF=foobar

Output becomes interesting if not for the first line (initializing with 0)

# Short Questions

27. [2 pts] Why do we need hardware support to handle an Interrupt?

The CPU has only 1 set of PC, SP, EFALGS. Any other piece of code will require these. Thus that would be unable to save the original interrupted process's PC, SP, EFLAGS

28. [3 pts ] Why other interrupts are masked while handling one interrupt?

This is not absolute necessity – there are ways to handle interrupts recursive (i.e., 1 interrupt arriving while processing another). However, that makes Interrupt handling more complicated.

However, with the algorithm that we saw in class, it is necessary that further Interrupts disabled while processing one. For instance, our algorithm saves PC, SP, EFLAGS at the bottom of the Kernel Interrupt Stack. This would be overwritten with the current Interrupt's PC, SP etc when a new Interrupt comes. Also, if you are in the middle of storing the PC, SP etc in KIS and another Interrupt comes, you would again have the KIS at an undesirable state (some containing the original Interrupted process's data, others the Interrupt handler's data)

# Short Questions

29. Define Synchronous and Async Exceptions with examples

Sync: Happening from inside, i.e., instruction of the current program (i.e., divide by zero, page fault)

Async: Happens from outside (e.g., Interrupts)

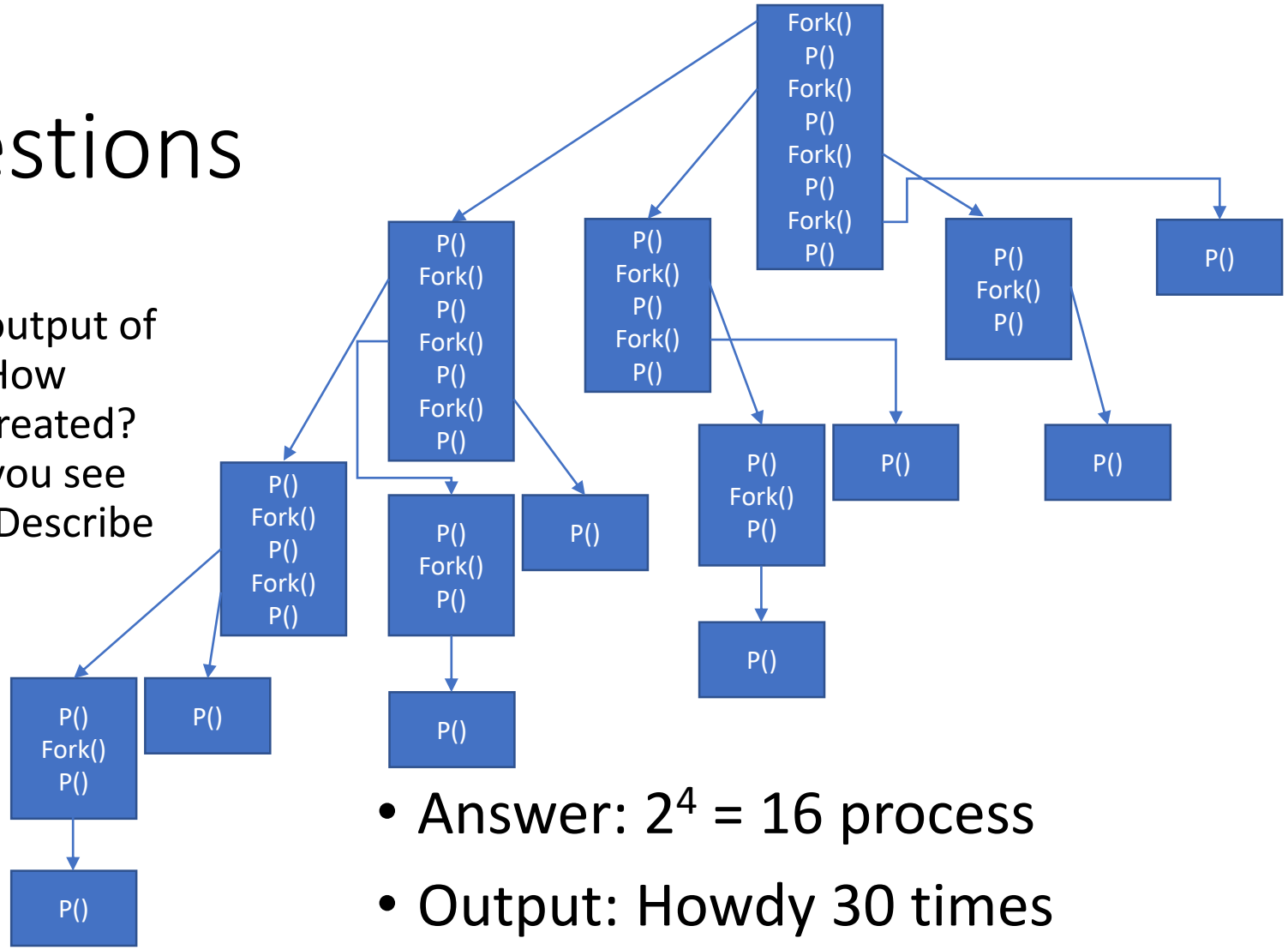
30. What is Kernel Interrupt Stack? Why does it need to be in Kernel memory?

The place where Interrupted processes info (PC, SP, EFLAGS) are stored. This needs to be in the Kernel because if it were in the user memory, right after the Interrupt Handler returns, another thread (with access to the User Interrupt Stack) could manipulate it so that the Kernel returns to a wrong address.

# Longer Questions

31. [10 pts] What is the output of the following program? How many processes will be created? How many times would you see the message “Howdy”? Describe using a process diagram.

```
for (int i=0; i<4; i++){
    fork();
    cout << "Howdy" << endl;
}
```




- Answer: $2^4 = 16$ process
- Output: Howdy 30 times

# Longer Questions

32.[10 pts] What will the file “test.out” contain after running the following:

```
char *fname = "test.out";
int fd1 = open(fname, O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IWUSR); //opens in read/write mode
write(fd1, "abcdef", 6);
int fd3 = dup(fd1); /* Allocates descriptor */
lseek (fd1, 0, SEEK_SET); // seeks from the beginning of the file
write(fd3,"wxyz", 4);
write (fd1, "pq", 2);
```

Answer: File will contain “wxyzpq”

# Longer Questions

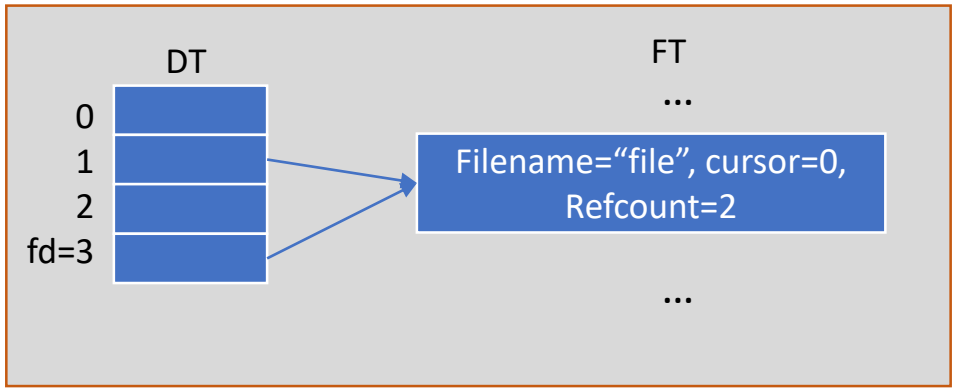
33. [20 pts] For the program below:

A. Draw the DT and FT after executing line 3, 5, 8

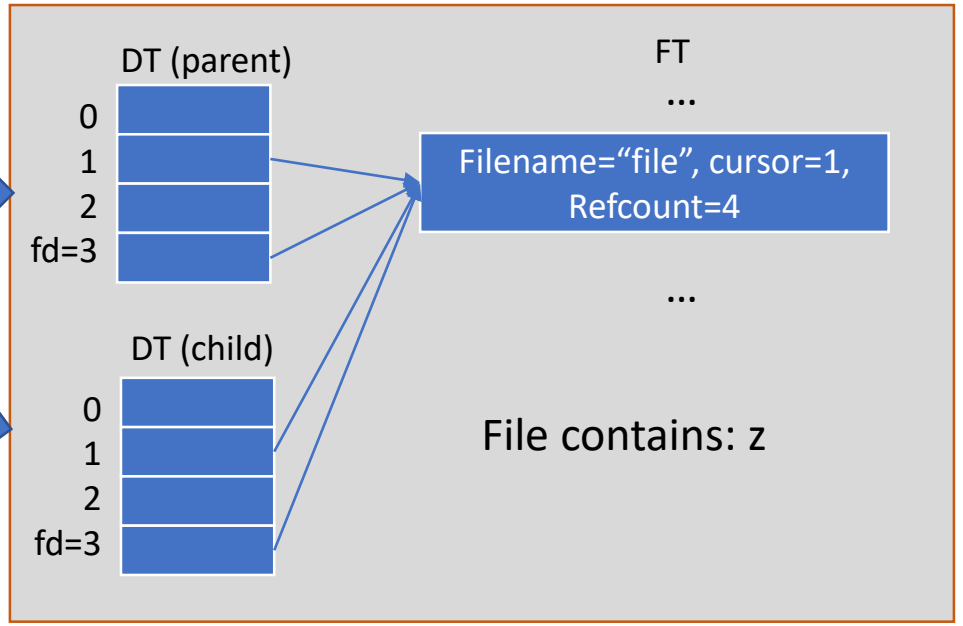
B. What does the file contain at the end? What is at the STDOUT

```
1 char c='a';
2 int fd = open ("file", O_WRONLY|O_CREAT, S_IRWXU);
// opens in write mode
3 dup2 (fd, 1);              // draw tables
4 if (!fork ())
5         cout << 'z';       // draw tables
6 else{
7         wait (0);
8         write (fd, &c, 1); // draw tables
9 }
```

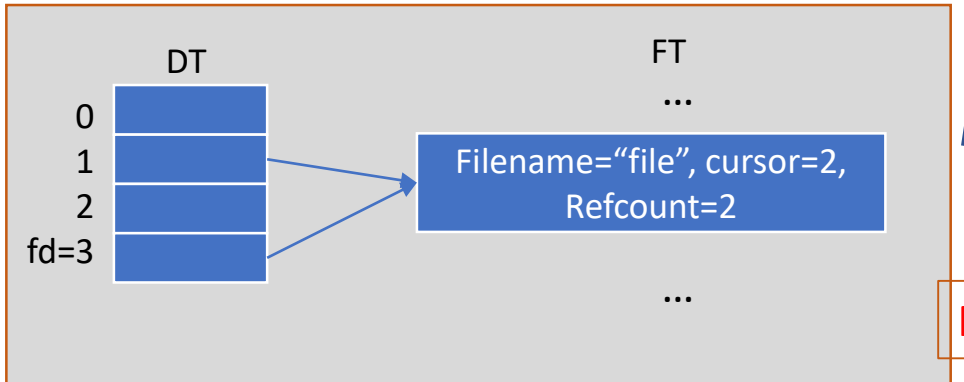
After line 3




After line 5




After line 8




File contains: za, STDOUT does not show anything

# Longer Questions

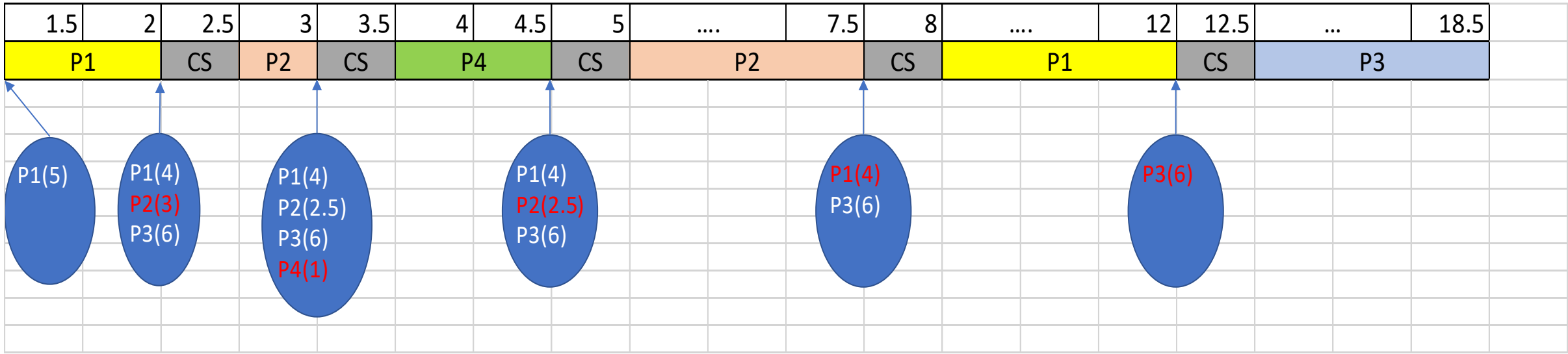
| Process | Arrival Time | Service Time |
|---|---|---|
| P1 | 1 | 5 |
| P2 | 2 | 3 |
| P3 | 2 | 6 |
| P4 | 3 | 1 |

- Question 34:

[20 pts] Assuming 0.5 sec overhead per context switch (round trip) in a 1-CPU-1-core system, schedule the following workload and compute Average Response Time (ART) under:

1. Shortest Remaining Time First (SRTF)
2. Round-Robin (RR) with time quantum=2sec

**SRTF ART = ((12-1) + (7.5-2) + (18.5-2) + (4.5-3))/4 = 34.5/4 = 8.625**



# Longer Questions

- Question 34:

[20 pts] Assuming 0.5 sec overhead per context switch (round trip) in a 1-CPU-1-core system, schedule the following workload and compute Average Response Time (ART) under:

1. Shortest Remaining Time First (SRTF)
2. Round-Robin (RR) with time quantum=2sec

| Process | Arrival Time | Service Time |
|---|---|---|
| P1 | 1 | 5 |
| P2 | 2 | 3 |
| P3 | 2 | 6 |
| P4 | 3 | 1 |

**RR ART = ((17.5-1) + (13.5-2) + (20-2) + (9.5-3))/4 = 54.5/4 = 13.625**