Q1: 9 Q2: 4+0 Q3: 2 Q4: 3 Q5: 9

Mid-Semester Examination
CS330: Operating Systems
Department of Computer Science and Engineering

Total: 20

Total marks: 50 **Duration: 90 minute**

Roll No: 210355 **Signature:** Divyansh

The exam is open books, notes, printed materials. No electronic device should be used during the examination. Consulting/discussing with anyone during the exam is strictly prohibited. *Unless specified explicitly, assume 64-bit X86 architecture with multiple CPUs.*
Best of luck!

1. Select the correct answer(s) for the following questions. Note that, you have to *mark all correct choices* to get the credits (no partial marks). No explanation required. $3 \times 5 = 15$

   (a) Which of the following C/assembly statement(s) *will never* result in a user-to-OS mode context switch? Assume that registers `r1` and `r2` are general purpose registers and no external interrupts occur during execution of the statements.

   A. `malloc(1024);`
   B. `MOV r1, r2; // Move the value in register r1 to register r2`
   C. `ADD r1, r2; // r1 = r1 + r2`
   D. `PUSH r1; // Push the value of r1 to the stack`
   E. `RET; // Return from the current function`

   2 (B)

   (b) Consider the following program in Unix/Linux system.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<fcntl.h>
5 int main(){
6           int i, arfd[5];
7           for(i=0; i<5; ++i){
8               char buf[16];
9               if(i % 2)
10                  arfd[i] = dup(arfd[i-1]);
11              else
12                  arfd[i] = open("xyz.txt", O_RDWR);
13              read(arfd[i], buf, 16);
14          }
17        return 0;
18 }
```

   Assuming 'xyz.txt' is a file in the current directory of size 1MB and the standard file descriptors are inherited (not created), which of the following option(s) is/are *false*?

   A. The file position for `arfd[0]` after the `for` loop will be 80
   B. The file position for all FDs in `arfd` after the for loop will be 32
   C. The file position of `arrfd[1]` after the `for` loop will be 32
   D. Total number of file objects created by the program is 3
   E. Total number of file objects created by the program is 5

   3

   (c) Which of the following statement(s) is/are *false*?

   A. Memory allocated using `mmap` or `brk` system calls can be freed using the `free` library call
   B. If a program invokes `malloc(1024)`, a `mmap` or `brk` system call will always be required
   C. Allocation of 1GB memory using mmap() will always increase the physical memory usage of the system by 1GB
   D. Allocation of 1GB memory using malloc() will always increase the physical memory usage of the system by 1GB
   E. Consider that, two processes, P_1 and P_2 are using 1GB and 2GB virtual memory, respectively, and both invoke the `fork` system call. Considering a typical `fork` implementation (as explained in the class), the total physical memory copied (to create the child processes) for P_1 can be grater than P_2.

   5

(d) Consider a scenario where because of a OS bug, *some part of the code segment* of a user process P is modified by the OS. Which of the following statements is/are *true* in this scenario? (Assume there are no other bugs in the OS and no other user process kills P)

A. P may be terminated by the OS because of an illegal memory access
B. P may be terminated by the OS because of an illegal instruction (invalid opcode)
C. P may successfully complete its execution
D. P can be abruptly terminated by the hardware without notifying the OS
E. If P is abruptly terminated, both the hardware and OS must be involved in terminating P

(e) Consider the following code snippet consisting of a function `valloc` in the OS.

```
int valloc(struct pcb *p)
{
   int retval = -1;
   //kprint is printf equivalent in the OS

   kprint("%p\n", &valloc); //L1
   kprint("%p\n", &retval); //L2

   // Some logic
   return retval;
}
```

Which of the following statement(s) is/are *false* regarding the behavior of OS mode execution. Assume a Unix/Linux like OS. Assume that, system call handler for a given system call S refers to the entire system call handling code for S in the OS.

A. The value printed in `L1` *will be the same* for all invocations of `valloc`.
B. The value printed in `L2` *will be the same* for all invocations of `valloc`.
C. Given that `valloc` is invoked only from a particular system call handler, the value printed in `L2` will be same for all invocation of `valloc`.
D. Given that `valloc` is invoked only from a particular system call handler, the value printed in `L2` will be same for all invocation of `valloc` for any given process.
E. The value printed in `L2` *will be the same* when `valloc` is invoked from different system call handlers for any given process.

2. True or False with justification. Justification should be a precise argument supporting your verdict. No marks without proper reasoning. $2 \times 5 = 10$

(a) If a file is duplicated five times (using the dup system call), all the file descriptor values returned from the dup system call will always be in sequence. (check)

False.

It might be possible that any initial file descriptor (for example 0) is closed before calling these five dup calls. So the 1st fd returned by the dup will the initially closed one because it searches for latest free fd. After that it can have 4 consecutive fds but that does not ensure all 5 to be in a sequence.

(b) System call arguments can be passed using the user stack of the process invoking the system call.

True

We do that in read argument where buffer initialized in user stack is passed as an argument.

(c) A user program can modify the stack pointer register (SP) in the middle of its execution and point the SP to an address in the heap and correctly complete the execution.

False.

With such functionality user function can change the pointer to kernel stack and access the privileged memory.

0

(d) Assuming that the OS is part of the process address space (split-address mode), the OS can invoke a function in the user code segment (by using assembly instructions like `call` or `jump`).

True,

Since OS shares the address space with the user process it move the programme counter to any code instruction in user code segment.

2

(e) A process currently executing in OS mode can not be descheduled in non-preemptive scheduling.

False

OS can deschedule any process even in non-preemptive scheduling since it can have the charge back during CPU execution via any trap, interrupts.

0

3. Consider the successful execution of the following program in the Unix/Linux system.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main(void){
  for(int i=0; i<3; ++i){
     if(i % 2 == 0)
        fork();
  }
  printf("%d\n", getpid());
  exit(0);
}
```

Answer the following questions regarding the execution behavior of the program considering the start of the execution from the `main` function. (No explanation required) $2 \times 5 = 10$

(a) What will be the minimum and maximum number of PCBs *actively used* during the execution?

1, 4

(b) What will be the minimum number of user mode to OS mode context switches?

7

(c) What will be the minimum number of OS mode to user mode context switches?

7

(d) What will be the number of kernel stacks *allocated* during the execution?

4

(e) What will be the minimum number of scheduler invocations assuming a non-preemptive scheduler?

6

4. Consider a single-CPU system with MLFQ scheduling scheme where five CPU bound batch processes and twenty short/interactive processes are always in the ready queue. The time quantum is 10ms and there are five priority levels (P_1 (lowest), P_2, ... , P_5 (highest)). Round-Robin (RR) is employed within any given priority level where new processes are added to the tail of the queue and the process in the head of the queue is scheduled. All processes except the batch processes always remain in the priority level P_5. [1+2+2=5]

(a) Given that, all batch processes starve, which priority level they will be in? [1]

P_1

(b) A periodic priority boost scheme with a boost interval B is employed to guarantee at least 10% of the CPU time for the batch processes, calculate the value of B? Assume that priority boosting takes place when selecting the next process where the processes from the lower priority levels are queued at the tail of P_5.(Provide brief explanation) [2]

Let's that the 5 batch processes ran their time quantums and Now after B ms when they are back to the queue tail they will start running after all the interactive proces.

do, $\frac{5 \times 10ms}{B + 20 \times 10ms} = \frac{10}{100} \Rightarrow B = 500ms.$

(c) In the previous question, what is the maximum time that any batch process remains in P_5. (Provide brief explanation) [2]

the last running batch proces will ran in 240-250ms interval (taking it last) and when it boosted it will wait for other 24 to run ~~it its~~ then its turn will come, hence 500ms.

5. Consider the following OS pseudocode implementing the `fork()` system call. [5+5=10]

```
1. do_fork(PCB *parent)
2. {
3.     PCB *child = create_new_pcb(parent);  // Allocate child PCB + Copy the parent into the child
4.     add_child(parent, child);             // Create parent and child relation
5.     adjust_child_pid(child);              // Assign a new PID, set the PPID correctly
6.     alloc_kstack(child);                  // Allocate kernel stack for child
7.     copy_kstack(child, parent);           // Copy the parent kernel stack to the child
8.     save_exec_state(child);               // Save the CPU state in child PCB
9.     add_to_ready_queue(child);            // Child added to scheduler ready queue
10.    return child->pid;                    // Parent returns
11. }
```

Assume that `do_fork()` is invoked from a generic system call handler (e.g., `do_syscall()`). Further, assume that all functions invoked from `do_fork()` perform the operations mentioned in the comment correctly. For example, assume that `create_new_pcb` never fails to allocate the child PCB and copy the parent to child. Therefore, ignore the lack of error handling for invocation of different functions within `do_fork`.

(a) The above implementation is error-prone. Point out the errors (what, when and why) in a precise manner. 5

(a) The process does not return the 0 pid for the child process. While returning from the the 'do-fork()' function. As the program needs that pid to figure out which is child process and which is parent process.

(b) What will you change in the above code to address the errors? 5

(b) The return from this function can be tackled separately for the child process by returning 0 to the address of child pid and returning child's pid to the address of parent.