

Information Technology University of the Punjab
FoE Midterm Examination – Spring 2025
SE301T – Operating Systems (B)

Name: _____ Solution _____

Roll No.: _____

Total time: 15 + 75 minutes

Total marks: 60

Date & day: 21st March, 2025 (Friday)

Instructions

- This exam will assess your CLOs as per OBE.**
- It is a **CLOSED BOOK/NOTES** exam.
- There are **TWO** parts of this exam.
- Part I has **FIFTEEN (15) multiple-choice questions**, each with four or five options, indexed as letters, a, b, c... Only one option is correct. You are required to mark the correct option for each MCQ on the provided bubble sheet.
- Part II has **THREE (3) questions/problems** for which you are required to provide solution in the space given for each question.
- If you are caught cheating or assisting others in cheating, your exam will be canceled, and disciplinary action will be taken in accordance with the university's academic honesty and anti-plagiarism policy.

CLOs

- Understand fundamental operating system abstractions such as processes, threads, files, semaphores, IPC abstractions, shared memory regions, etc.
- Develop important algorithms and services provided by the operating systems: Scheduling, memory management, ls, cat, zip, etc.
- Develop multi-threaded applications using the learnt parallel programming concepts

Part I – 1,3,6,7,8,9,12,13 (CLO 1)	Part I – 2,4,5,10,11,14,15 (CLO 2)	Part II – 1 (CLO 2)	Part II – 2 (CLO 2)	Part II – 3 (CLO 1)	Total
/ 8	/ 7	/ 12	/ 13	/ 20	/ 60

Instructor: Mr. Umair Shoaib

Teaching Assistant: Ms. Wardah Binte Naveed

Signature: _____

PART I

- How many processes (including the process calling the first fork) will be created by the following code?
fork();
int rc = fork();
if (rc >= 0)
 fork();
 - 3
 - 4
 - 5
 - 6**
- Which scheduler performs the best for interactive processes?
 - Round-Robin**
 - Shortest-To-Completion First
 - Shortest Job First
 - FIFO
- A pipe created inside a child will be available to:
 - Its parent
 - Its children**
 - Its siblings
 - All of these
 - None of these
- Suppose two processes A and B have been assigned tickets 100 and 300 respectively. In stride scheduling, the pass value of which process will increase more rapidly?
 - A**
 - B
 - The value of stride must be known to determine the increase rate of pass value
 - It depends upon the winning ticket

5. The `yield` system call (`yield()` in Xv6, `sched_yield()` in Linux) makes its caller:
 - a. Running
 - b. Runnable
 - c. Zombie
 - d. Blocked
 - e. None of these
6. On an x86 CPU, when the “`int 64`” instruction executes, the CPU moves the instruction pointer (EIP) to the respective interrupt handler. How does the CPU know where this handler is?
 - a. The address of the interrupt handler is hardwired in the architecture
 - b. The user program must specify the address of the handler before calling the `int 64` instruction
 - c. The CPU checks in the interrupt descriptor table set up earlier by the OS during initialization
 - d. Any of these options may be true based on the implementation
 - e. None of these is true
7. Which of the following is true about the `kill` system call in Linux?
 - a. It can only send signals to child processes of the calling process
 - b. It only terminates processes and cannot send other signals
 - c. It cannot send signals to itself
 - d. It can send signals to any process, given the caller has the necessary permissions
 - e. None of these
8. When “`int 64`” executes, the CPU automatically pushes the values of a few registers onto the stack. Which stack does it push to?
 - a. Runtime stack of the process
 - b. Kernel stack of the process
 - c. Runtime stack of the kernel
 - d. None of these
9. Which of the following statements best describes external fragmentation?
 - a. There is no contiguous memory region large enough to satisfy a segment allocation request
 - b. There is less free space in the memory (contiguous or non-contiguous) than the requested allocation
 - c. The physical memory has been divided into too many equal sized chunks
 - d. The OS cannot allocate memory for a process due to few number of free page frames left
10. What happens as a result of increasing the time slice duration in round-robin scheduling?
 - a. The response time improves
 - b. The response time worsens
 - c. The turnaround time worsens
 - d. The context switch takes up a bigger chunk of the time slice duration
11. In xv6 or Linux (both have similar implementations, so what you know about xv6 is also true for Linux), which of the following statements is true about context switching?
 - a. A context switch can occur when the CPU is executing a process in user mode
 - b. A context switch can only occur when a process makes a system call
 - c. A context switch can only occur when the CPU is executing in kernel mode
 - d. A context switch can only be triggered by the process calling `yield` (or `sched_yield` in Linux) in user space
12. To return from the kernel space to the user space, which x86 assembly instruction is used?
 - a. `ret`
 - b. `iret`
 - c. `trapret`
 - d. `forkret`
13. In an x86 system, how many bytes of storage does `malloc (sizeof(float) * 128)` allocate?
 - a. 128
 - b. 256
 - c. 512
 - d. 1024
14. For a 20-bit address space with 2KB pages, how many entries would be there in the page table?
 - a. 2^{20}
 - b. 2^{10}
 - c. 2^{11}
 - d. 2^9
 - e. Cannot be determined with the information provided
15. In a linear page table, the page number is the same as:
 - a. The index of the page table entry
 - b. The Physical Frame Number (PFN)
 - c. The decimal equivalent of the first 20 bits of the PTE
 - d. None of these

PART II

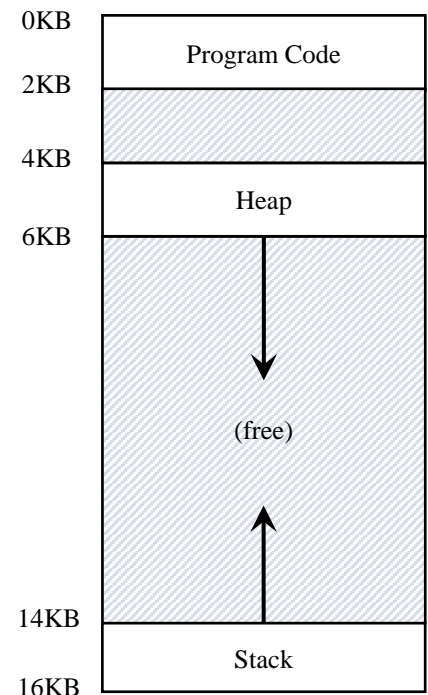
- The adjacent figure shows the address space of a process. This address space is to be mapped to a 256KB physical memory using both segmentation and paging.

In case of segmentation, the base, size, direction of growth and top bits in virtual address for code, heap and stack segments are specified as follows:

Segment	Base	Size	Direction	Bits
Code	36K	2KB	Positive	00
Heap	128K	2KB	Positive	01
Stack	200K	2KB	Negative	11

Similarly, for paging with a page size of 1KB, the pages in the address space are mapped to physical frames as follows (note that the total allocated space in case of paging (7KB) is more than the total allocated space in segmentation (6KB)):

Page Number	Frame Number
0	33
1	34
4	192
5	193
6	194
14	244
15	245



For the virtual addresses given below, find the corresponding physical addresses for both techniques. You should first check if the translation would be valid, if not, you can stop and state that the address is out of bounds in case of segmentation, and not mapped to a physical frame in case of paging.

Virtual address	Physical address (Segmentation)	Physical address (Paging)
6277	133253, out of bounds	<u>1100 0010 00 1000 0101</u>
15219	203635	<u>1111 0100 11 0111 0011</u>
1998	38862	<u>0010 0010 11 1100 1110</u>

SEGMENTATION:

6277:

Binary = 01 1000 1000 0101 (lies in heap)

Physical address = $128k + (1000\ 1000\ 0101)_b = 128k + 2181 = 133253$

This is out of bounds, as heap segment range is 128k – 130k or 131072 – 133120

15219:

Binary: 11 1011 0111 0011 (lies in stack)

Physical address = $200k - (16k - 15219) = 200k - 1165 = 203635$

This is in bounds, as stack segment range is 198k – 200k or 202752 – 204800

1998:

Binary: 00 0111 1100 1110 (lies in code)

Physical address = $36k + (0111\ 1100\ 1110)_b = 38862$

This is in bounds, as code segment range is 36k – 38k or 36864 – 38912

PAGING:

6277:

Binary: 01 10 00 1000 0101 (VPN = 6)

Physical address: 1100 0010 00 1000 0101 (concatenating binary of frame number, 194, and offset)

{Space for Part II – 1}

15219:

Binary: 11 10 11 0111 0011 (VPN = 14)

Physical address: 1111 0100 11 0111 0011 (concatenating binary of frame number, 244, and offset)

1998:

Binary: 00 01 11 1100 1110 (VPN = 1)

Physical address: 0010 0010 11 1100 1110 (concatenating binary of frame number, 34, and offset)

Rubric for evaluation:

Total marks for segmentation: 7.5 (2.5 for each conversion)

For each VA to PA translation in segmentation, marks are to be awarded as follows:

1 for identifying the segment.

1 for physical address

0.5 for checking valid/invalid

Total marks for paging: 4.5 (1.5 for each conversion)

For each VA to PA translation in paging, marks are to be awarded as follows:

0.5 for identifying the Virtual page number

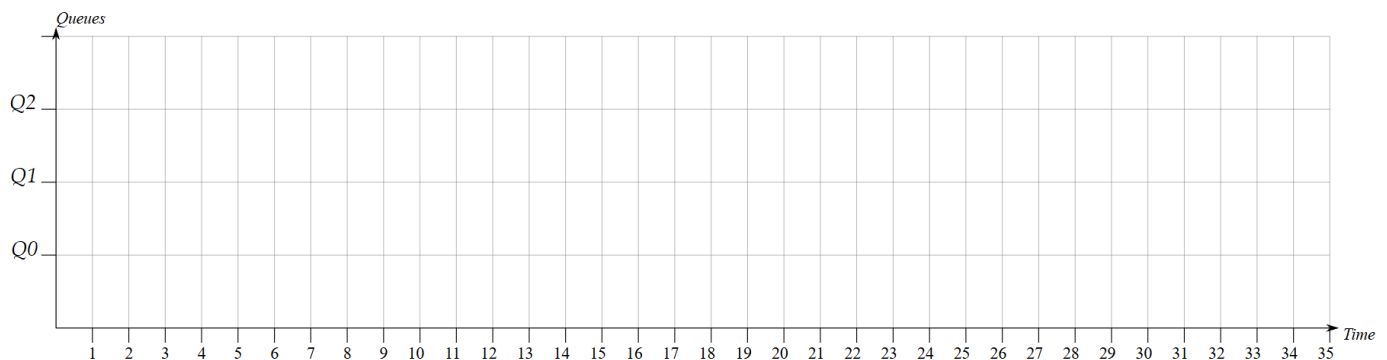
1 for physical address by concatenating corresponding frame number and offset

2. Consider the following MLFQ parameters:
- Number of queues: 3 (Q2, Q1, Q0)
 - Allotment time in Q2 (highest priority queue): 2 ms
 - Allotment time in Q1 (highest priority queue): 4 ms
 - Allotment time in Q0 (highest priority queue): 4 ms
 - Quantum (time slice) in all queues: 1 ms
 - Priority boost every 25ms

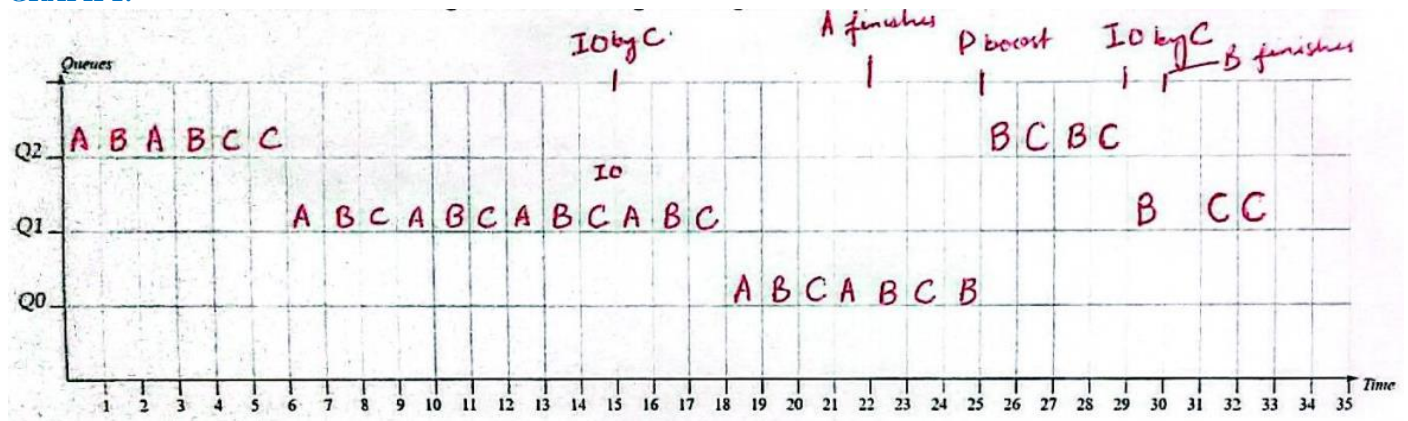
Determine the schedule and the average turnaround and response times for the following workload:

Job	Arrival time	Total runtime	I/O
A	0	8	None
B	1	12	None
C	4	12	2 ms I/O every 5 ms

Use the grid provided below to plot the schedule. It would be better to draw the schedule in a rough space first and then copy it here once finalized, to avoid cutting and over-writing on this grid.



GRAPH 1:



C issues first IO at 15 ms.

A finishes at 22 ms.

Priority is boosted at 25 ms. After priority boost, either of B or C may run. In Graph 1, B is made to run. In Graph 2 (below), C is made to run.

C issues second IO at 29 ms (28 ms in Graph 2).

B finishes at 30 ms.

31st ms is idle (in Graph 1) as IO by C has not completed its 2 ms yet. In contrast, in Graph 2, as IO gets to complete its 2ms period coincidentally with B finishing, C runs immediately when B finishes. As a result, both graphs have a difference of 1 ms in turnaround times for this workload (specifically C).

C ends at 33 ms in Graph 1. In Graph 2, C finishes at 32 ms.

Turnaround time in Graph 1: $\frac{(22-0)+(30-1)+(33-4)}{3} = 26.67$

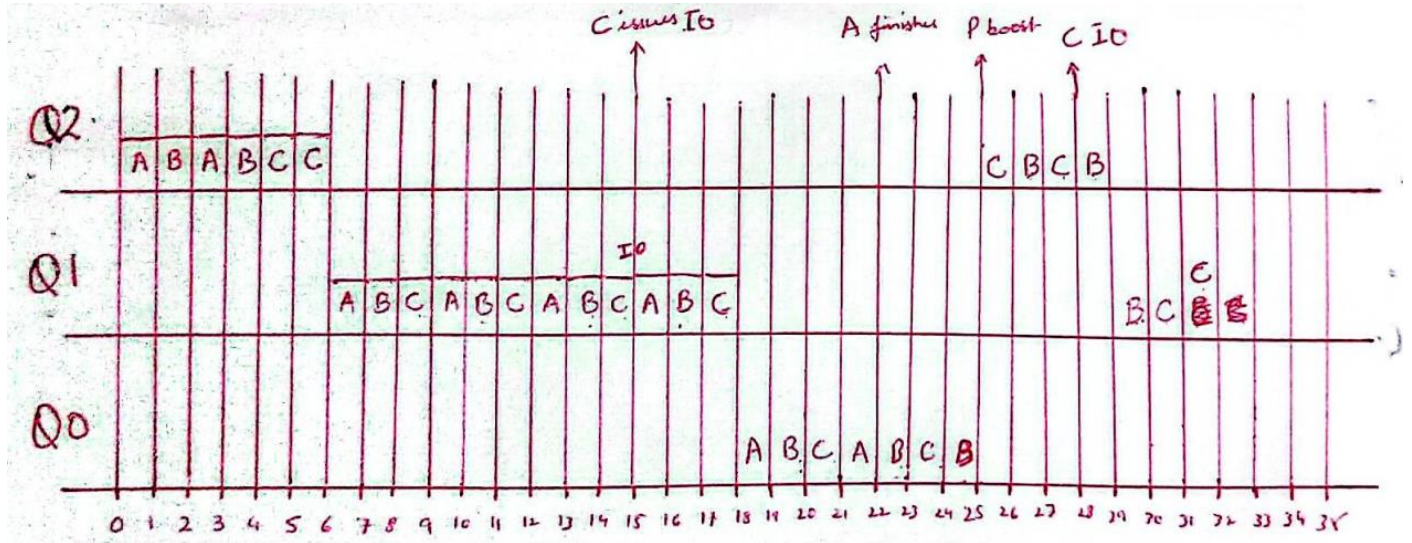
Response time in Graph 1: $\frac{(0-0)+(1-1)+(4-4)}{3} = 0$

{Space for Part II – 2}

Turnaround time in Graph 2: $\frac{(22-0)+(30-1)+(32-4)}{3} = 26.33$

Response time in Graph 2: $\frac{(0-0)+(1-1)+(4-4)}{3} = 0$

GRAPH 2:



Rubric for evaluation:

Apart from the above two graphs, there are other possibilities as well; for example, when priorities change from Q2 to Q1 at time 6 ms, the scheduler could have validly chosen to run in B -> A -> C order or any other order as all processes are in Q2 and RUNNABLE at 6ms, so full marks are to be awarded for any such valid variations in solution.

Graph: 9 marks

Turnaround time: 2 marks

Response time: 2 marks

Division of graph marks:

2 marks for tracking allotment time inside queues and changing process priorities.

2 marks for monitoring process runtime and ending all processes at the right time.

2 marks for round-robin inside queues.

1 mark for C's IO.

2 marks for priority boost at correct time.

3. Write a program that performs the following operations:
- The parent process creates a pipe.
 - It then creates two children using `fork`.
 - Child 1 closes its read end of the pipe and duplicates the pipe's write end to the `STDOUT_FILENO`.
 - Child 1 then uses one of the `exec` variants to execute the `echo` command with "Hello world!" as argument string. (`echo` prints its arguments to standard output. In bash, the command would be: `echo Hello world!`)
 - Child 2 closes its write end of the pipe and duplicates the pipe's read end to the `STDIN_FILENO`.
 - Child 2 then uses one of the `exec` variants to execute the `rev` command with no arguments or options. (`rev` command reverses the order of characters in the input. In bash, the command would be: `rev`)
 - The parent closes its both ends of the pipe and waits for both children to exit. It exits after printing some message about successful completion, such as, "Both children exited successfully, parent exiting now!"

Templates of some functions / system calls that you may require are given below:

```
pid_t fork(void);
int pipe(int pipefd[2]);
int dup2(int oldfd, int newfd);
int execvp ( const char * file , char * const argv []);
int execlp ( const char * file , const char * arg, ... , (char *) NULL);
pid_t wait(int * _Nullable wstatus);
```

After providing the code, also explain the behavior of the program, such as, if it takes any input from the user, or if it outputs something, or any other significant aspect of its operation and the final outcome.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int pipefd[2];
    pipe(pipefd);

    pid_t child1 = fork();

    if (child1 == 0) {
        close(pipefd[0]);
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[1]);

        execlp("echo", "echo", "Hello, world!", NULL);
        perror("execlp failed");
        exit(1);
    }

    pid_t child2 = fork();

    if (child2 == 0) {
        close(pipefd[1]);
        dup2(pipefd[0], STDIN_FILENO);
        close(pipefd[0]);

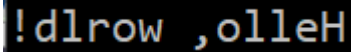
        execlp("rev", "rev", NULL);
        perror("execlp failed");
        exit(1);
    }

    close(pipefd[0]);
    close(pipefd[1]);
```


{Space for Part II – 3}

```
wait(NULL);  
wait(NULL);  
  
return 0;  
}
```

This program does not take any input. It outputs the reverse of “Hello, world!” string on standard output as a result of execution of the `rev` command as shown below:



```
!dlrow ,olleH
```

Rubric for evaluation:

2 marks for pipe creation.

4 marks for children creation using fork and using the “if conditions” correctly for both children and parent code.

4 marks for child 1 code: 2 marks for closing read end and duplicating write end to standard output. 2 marks for calling `exec` correctly for “echo Hello, world!” in Child 1.

4 marks for child 2 code: 2 marks for closing write end and duplicating read end to standard input. 2 marks for calling `exec` correctly for “rev” in Child 2.

4 marks for parent’s code: 1 for closing its read/write ends of the pipe, 2 for waiting for both children, 1 for printing messages, exiting.

2 marks for predicting the output correctly.