Feedback on Quiz # 2 (Summative)

The **Quiz # 2** was composed of **10** questions, which were randomly selected from a Question Bank of 30 questions. Answers and feedback comments for all of the questions are given below:

Q1.1	Which of the following statements about system calls are true?
(a)	A system call is an interface to request services from the Operating System kernel.
(b)	Most system calls are accessed via an Application Programming Interface (API).
(c)	All system calls are written in assembly language, as they need to communicate with the hardware.
(d)	System calls can only be used through a command line interface.
(e)	Most modern smartphone Operating Systems (e.g. iOS, Android) do not provide system calls because they are simplified for handheld devices.
Feedback: Systems calls are programming interfaces to the services provided by the Operating System. File I/O is one of kernel services which can be accessed via System calls. There are other kernel services, such as duplicating a process (fork()), which can also be accessed via System calls. Typically they are written in a high-level language (C or C++). They are mostly accessed by programs via a high-level Application Programming Interface (API), though some of them can be accessed (used) as direct commands in a command line interface. Modern smartphones are compact versions of desktop computers. So, their OS's do provide system calls, despite some features being simplified for handheld devices.	

Q1.2	Which of the following statements about system calls are true?
(a)	System calls are typically written in a high-level programming language (e.g. C, C++)
(b)	System calls can be invoked through a GUI or a command line interface.
(c)	System calls are only used for handling file I/O.
(d)	System calls can only be used in compiled programs.
(e)	Modern smartphone Operating Systems (e.g. iOS, Android) do not provide system calls because they are simplified for handheld devices.
Feedback:	

Systems calls are programming interfaces to the services provided by the Operating System. File I/O is one of kernel services which can be accessed via System calls. There are other kernel services, such as duplicating a process (fork()), which can also be accessed via System calls. Typically they are written in a high-level language (C or C++). They are mostly accessed by programs via a high-level Application Programming Interface (API), though some of them can be accessed (used) as direct commands in a command line interface. Modern smartphones operating systems are compact versions of those of a desktop computer. So, their OS's do provide system calls, despite some features being simplified for handheld devices.

Q1.3	Which of the following statements about system calls are true?
(a)	Modern smartphone Operating Systems (e.g. iOS, Android) provide system calls to mobile applications.
(b)	System calls provide an interface to request services from the Operating System kernel.
(c)	System calls are all written in assembly language, as they need to communicate with the hardware.
(d)	System calls can only be used through a command line interface.
(e)	System calls are only used for handling file I/O.

Feedback:
Systems calls are programming interfaces to the services provided by the Operating System. File I/O is one kernel service which can be accessed via System calls. There are other kernel services, such as duplicating a process (fork()), which can also be accessed via System calls. Typically they are written in a high-level language (C or C++). They are mostly accessed by programs via a high-level Application Programming Interface (API), though some of them can be accessed (used) as direct commands in a command line interface. Modern smartphones are compact versions of desktop computers. So, their OS's do provide system calls, despite some features being simplified for handheld devices.

Q2.1	Which of the following statements about interrupts are true?
(a)	Interrupts can be triggered by both hardware and software
(b)	The hardware triggers interrupts by sending a signal to the CPU
(c)	Hardware can trigger an interrupt at any time
(d)	Interrupts cannot be triggered by software
(e)	Hardware can only trigger interrupts at specific times

Feedback:
Interrupts can be triggered by hardware (for example, a mouse click) or software (for example, when a program tries to access a file, which leads to a system call).

Hardware may trigger an interrupt at any time by sending a signal to the CPU.

Q2.2	Which of the following statements about interrupts are true?
(a)	Interrupts can be triggered by both hardware and software
(b)	Software triggers interrupts by executing a special operation
(c)	System calls are invoked using a software interrupt
(d)	Interrupts cannot be triggered by hardware
(e)	Software interrupts are invoked through the hardware

Feedback:
Interrupts can be triggered by hardware (for example, a mouse click) or software (for example, when a program tries to access a file, which leads to a system call).

Software triggers an interrupt by executing a special operation – to make a system call and switch to kernel mode.

Q2.3	Which of the following statements about interrupts are true?
(a)	When an interrupt occurs, the CPU immediately stops what it is doing and transfers control to the interrupt service routine
(b)	Hardware devices can trigger an interrupt at any time
(c)	When an interrupt occurs, the CPU waits until the current process is completed before transferring control to the interrupt service routine
(d)	Hardware can only trigger interrupts at specific times
(e)	Software interrupts are invoked through the hardware

Feedback:
The CPU reacts to an interrupt by stopping what it is doing and immediately transferring execution to a fixed location (usually containing the starting address of the interrupt service routine).

Hardware may trigger an interrupt at any time by sending a signal to the CPU.

Software triggers an interrupt by executing a special operation – to make a system call and switch to kernel mode.

Q3.1	What is the time complexity of this algorithm?
	
	`sum=0`
	`product=1`
	
	`for i=1 .. 7*n`
	`  for j=1 .. n`
	`  sum = sum+i*j`
	
	`for j=20 .. 100*n`
	`  product = product*j`

(a)	O(7n)
(b)	O(n)
(c)	O(n^2)
(d)	O(7n^2)
(e)	O(7n+n)

Feedback:
For the first pair of nested loops:
The outer loop is O(n)
Nested within that loop is another loop, which is O(n).
Therefore the combined complexity is O(n^2).
We do not need to consider the constants because they are not relevant to the time complexity.

The second loop is O(n).
Therefore the overall complexity of the algorithm is O(n^2)

Q3.2	What is the time complexity of the following algorithm?
	
	`sum=0`
	`product=1`
	
	`for i=1 .. 10*n`
	`  sum = sum+10*i`
	
	`for i=1 .. 10*n`
	`  for j=1 .. 10*n`
	`  product = product+i*j`

(a)	O(10n)
(b)	O(10n + 10n)
(c)	O(20n)
(d)	O(10n + 100n^2)
(e)	O(n^2)

Feedback:
The complexity of the first loop is O(n)
The complexity of the second pair of nested loops is O(n^2)

We should not consider the constant multipliers because they do not affect the time complexity.
As n becomes large, the complexity of the second loop will dominate.
We therefore consider the overall complexity to be O(n^2)

Q3.3	What is the time complexity of the following algorithm?
	
	`sum=0`
	`product=1`
	
	`for i=20 … 100*n`
	`  for j=20 .. 100*n`
	`  sum=sum+a[i,j]`
	
	`for i=1..100*n`
	`  for j=1 .. 100*n`
	`  product=product*a[i,j]`

(a)	O(n^2)
(b)	O(2n^2)
(c)	O(10000n^2)
(d)	O(6400n^2 + 10000n^2)
(e)	O(16400n^2)

Feedback:
The complexity of the first pair of nested loops is O(n^2)
The complexity of the second pair of nested loops is O(n^2)

We should not consider the constant multipliers because they do not affect the time complexity.
We therefore consider the overall complexity to be O(n^2)

Q4.1	Select all the following statements that are true:
(a)	A Job (long term) scheduler is invoked infrequently compared to a CPU scheduler.
(b)	A CPU (short term) scheduler should execute considerably faster than a Job scheduler.
(c)	A Job (long term) scheduler determines which programs are admitted to the system for processing.
(d)	A Job (long term) scheduler selects a process from the processes that are in the ready queue.
(e)	A CPU (short term) scheduler controls the degree of multiprogramming in a system, particularly when a multi-core CPU is used.

Feedback:

- *Short term Scheduler:*

  It is also known as a **CPU scheduler**. Its main objective is to increase the system's performance in accordance with the chosen set of criteria (like Avg. Waiting Time / Response Time). CPU scheduler selects a process among the processes that are in the ready queue and allocates a CPU to it for execution. CPU (short term) schedulers should execute considerably faster than Job (long term) schedulers as they are invoked frequently as compared to Job (long term) schedulers.

- *Long term Scheduler:*

  It is also known as **Job scheduler**. A Job (long term) scheduler determines which programs are admitted to the system for processing. It selects processes from the job queue and loads them into memory for execution. The Job (long term) scheduler controls the degree of multiprogramming and ensures a balanced mix of I/O vs CPU bound processes in the system. The Job (long term) scheduler is invoked infrequently as compared to the CPU (short term) scheduler; hence it can be relatively slower.

Q4.2	Select all the following statements that are true:
(a)	I/O-bound processes usually take longer to execute than CPU-bound processes, as they spend most of the time waiting for data to be read or written.
(b)	I/O-bound processes perform a lot of I/O operations, but they also do computations.
(c)	I/O-bound processes generally have shorter CPU bursts than CPU-bound processes.
(d)	I/O-bound processes perform a lot of I/O operations, that is why they usually occupy the CPU for a long period.
(e)	CPU-bound processes focus on doing computations, therefore, they do not spend any time doing I/O.

Feedback:

- *I/O-bound process:*
    - I/O-bound processes take longer to execute than CPU-bound processes because they spend a lot of time in I/O operations (reading, writing files etc.) which can take a substantial amount of time.
    - I/O-bound processes perform computations as well, but they spend relatively less time doing so.
    - I/O-bound processes perform more I/O operations than computation. Thus, they have shorter CPU bursts than CPU-bound processes.
    - When an I/O operation is performed, the process is not running on the CPU.
- *CPU-bound process:*
    - CPU-bound processes take shorter time to execute than I/O-bound processes because they spend more time doing computations on the CPU than waiting for data to be read or written (I/O operations).
    - CPU-bound processes will sometimes perform I/O operations as well.
    - CPU-bound processes usually have many long CPU bursts.

Q4.3	Select all the following situations that are considered voluntary process terminations:
(a)	A process terminates with an error code.
(b)	The user closes a process by clicking the close button of an application.
(c)	A process completes its execution and exits.
(d)	The operating system terminates an idle process to free up memory space.
(e)	A process is terminated when it tries to access the (n+1)th element of an array which has n elements.

Feedback:

- A process terminates with an error code when an error is caught. The process voluntarily executes the exit system call to indicate to the operating system that it has finished.
- Clicking the close button on an application triggers an event which asks that the process voluntarily executes the exit system call and finish it's execution.
- "A process completes its execution then exits" is a common voluntary exit without any error.
- The operating system can terminate (kill) idle processes to free up memory space when needed, but this is not voluntary process termination.
- "A process tries to access the (n+1)th element of an array which only has n elements" is an out-of-bounds memory access, which is a fatal error (usually reported as a segmentation fault).

Q5.1	Which of the following statements are true about OS memory management and multitasking?
(a)	Data that has been used recently is likely to be stored in a fast memory (cache), which lies between RAM and the CPU.
(b)	Virtual memory abstracts main memory, separating logical and physical memory.
(c)	CPU scheduling is the process of deciding which job in the ready queue is to be executed next
(d)	Virtual memory increases the size of the physical memory in the system
(e)	CPU scheduling is the process of deciding which process in the blocked queue should be executed next.

Feedback:
Cache memory is used to store data that has been used recently – it is located between main memory and the CPU for faster access.
Virtual memory is an abstraction of main memory into a large, uniform array of storage; separating logical memory as viewed by the user processes from physical memory. It does not actually increase the size of physical memory.
CPU scheduling is the process of deciding which job in the ready queue – not the blocked queue – is to be executed next.

Q5.2	Which of the following statements are true about OS memory management and multitasking?
(a)	The CPU performs multitasking by executing multiple jobs and switching frequently between them
(b)	Virtual memory allows the execution of a process that it is not completely in memory
(c)	CPU scheduling is the process of deciding which job in the ready queue is to be executed next
(d)	CPU scheduling is the process of deciding which process in the blocked queue should be executed next.
(e)	Data that has been used recently is stored in cache memory, which is managed by software.

Feedback:
Multi-tasking relies on switching between programs so frequently that the users can interact with each program while it is running.

Virtual memory is an abstraction of main memory into a large, uniform array of storage; separating logical memory as viewed by the user processes from physical memory. It therefore allows users to run programs larger than actual physical memory.
CPU scheduling is the process of deciding which job in the ready queue – not the blocked queue – is to be executed next.
Cache memory is used to store data that has been used recently – it is located between main memory and the CPU for faster access. The cache memory is managed by hardware not by software.

Q5.3	Which of the following statements are true about OS memory management and multitasking?
(a)	Data that has been used recently is likely to be stored in a fast memory (cache), which lies between RAM and the CPU.
(b)	The CPU performs multitasking by executing multiple jobs and switching frequently between them.
(c)	A process is allocated a maximum time quantum after which it is interrupted. This has the effect of multiple processes appearing to execute concurrently.
(d)	Virtual memory increases the size of the physical memory in the system.
(e)	Virtual memory maps logical addresses to physical memory addresses, which includes the secondary storage (swap area) as well.

Feedback:
Cache memory is used to store data that has been used recently – it is located between main memory and the CPU for faster access.
Multi-tasking relies on switching between programs so frequently that the users can interact with each program while it is running.
The effect of multiple processes executing concurrently is delivered by allocating each process a maximum time quantum, after which it is interrupted and another process continues.
Virtual memory is an abstraction of main memory into a large, uniform array of storage; separating logical memory as viewed by the user processes from physical memory. It does not actually increase the size of physical memory, and is not concerned with swapping jobs between main memory and the secondary storage (swap area).

Q6.1	Consider the following Java function:
	`public static void compute(long a){`
	`  double b = 3.141;`
	`  float  c = 94.93;`
	`  short  d = 42;`
	`  <some more code here>`
	`}`
	
	Choose option(s) indicating the correct slots allocation for this function in the Java bytecode.

(a)	**a = slots 0,1; b = slots 2,3; c = slot 4; d = slot 5**
(b)	this = slot 0; a = slots 1,2; b = slots 3,4; c = slot 5; d = slot 6
(c)	this = slot 0; a = slot 1; b = slot 2; c = slots 3,4; d = slots 5,6
(d)	a = slot 0; b = slot 1; c = slots 2,3; d = slots 4,5

Feedback:
The slots will be allocated to local variables according to the following scheme:

Variable	Slots	Explanation
a	0,1	a is long, therefore needs two slots
b	2,3	b is double, therefore needs two slots
c	4	c is float, therefore needs one slot
d	5	d is short, therefore needs one slot

Note: The `compute` function is static, therefore **this** reference will not exist.

Q6.2	Consider the following Java function:
	`public static void compute(long a){`
	`  float  b = 94.93;`
	`  short  c = 42;`
	`  double d = 3.141;`
	`  <some more code here>`
	`}`
	
	Choose option(s) indicating the correct slots allocation for this function in the Java bytecode.

(a)	**a = slots 0,1; b = slot 2; c = slot 3; d = slots 4,5**
(b)	this = slot 0; a = slots 1,2; b = slot 3; c = slot 4; d = slots 5,6
(c)	this = slot 0; a = slot 1; b = slots 2,3; c = slots 4,5; d = slot 6
(d)	a = slot 0; b = slots 1,2; c = slots 3,4; d = slot 5

Feedback:
The slots will be allocated to local variables according to the following scheme:

Variable	Slots	Explanation
a	0,1	a is long, therefore needs two slots
b	2	b is float, therefore needs one slot
c	3	c is short, therefore needs one slot
d	4,5	d is double, therefore needs two slots

Note: The `compute` function is static, therefore **this** reference will not exist.

Q6.3	Consider the following Java function:
	`public void compute(long a){`
	`  float  b = 94.93;`
	`  double c = 3.141;`
	`  short  d = 42;`
	`  <some more code here>`
	`}`
	
	Choose option(s) indicating the correct slots allocation for this function in the Java bytecode.

(a)	**this = slot 0; a = slots 1,2; b = slot 3; c = slots 4,5; d = slot 6**
(b)	a = slots 0,1; b = slot 2; c = slots 3,4; d = slot 5
(c)	this = slot 0; a = slot 1; b = slots 2,3; c = slot 4; d = slots 5,6
(d)	a = slot 0; b = slots 1,2; c = slot 3; d = slots 4,5

Feedback:
The slots will be allocated to local variables according to the following scheme:

Variable	Slots	Explanation
this	0	this reference, as the function is non-static
a	1,2	a is long, therefore needs two slots
b	3	b is float, therefore needs one slot
c	4,5	c is double, therefore needs two slots
d	6	d is short, therefore needs one slot

Note: The `compute` function is non-static, therefore **this** reference will exist.

Q7.1	Consider the following program:
	`int main(){`
	`  printf("A");`
	`  fork();`
	`  printf("B");`
	`  fork();`
	`  fork();`
	`  printf("C");`
	`  fork();`
	`  return 0;`
	`}`
	How many times will the letters “A”, “B” and “C” be printed?

(a)	**A: 1 time, B: 2 times, C: 8 times**
(b)	A: 1 time, B: 2 times, C: 4 times
(c)	A: 1 time, B: 4 times, C: 4 times
(d)	A: 1 time, B: 4 times, C: 8 times

Feedback:

When a fork() system call is executed, it creates a copy of that process (the parent process). Execution continues in both processes (parent & child) at the point following the return from fork() call.

The main idea is that the number of times “a letter” is printed is equal to the number of processes that will exist at that point where the letter is printed.

- At the start there is only one process, so ‘A’ will be printed only once because it is executed before making the first fork() system call.
- After the first fork() there are two processes. ‘B’ will be printed 2 times.
- After the second and third calls to fork() there will be 8 processes. ‘C’ will be printed 8 times – once in each process.
- There is a further fork() system call but this won’t lead to any further output but will lead to the creation of further processes.

Therefore, the correct answer is A:1 time, B: 2 times, C: 8 times

Q7.2	Consider the following program:
	`int main(){`
	`  fork();`
	`  printf("A");`
	`  fork();`
	`  printf("B");`
	`  fork();`
	`  printf("C");`
	`  fork();`
	`  return 0;`
	`}`
	How many times will the letters “A”, “B” and “C” be printed?

(a)	**A: 2 time, B: 4 times, C: 8 times**
(b)	A: 2 time, B: 2 times, C: 8 times
(c)	A: 1 time, B: 2 times, C: 8 times
(d)	A: 1 time, B: 2 times, C: 4 times

Feedback:

When a fork() system call is executed, it creates a copy of that process (the parent process). Execution continues in both processes (parent & child) at the point following the return from fork() call.

The main idea is that the number of times “a letter” is printed is equal to the number of processes that will exist at that point where the letter is printed.

- After the first fork(), there are 2 processes. ‘A‘ will be printed twice.
- After the second fork(), there are 4 processes. ‘ B‘ will be printed 4 times.
- After the third fork(), there are 8 processes. ‘ C‘ will be printed 8 times.
- There is a further fork() system call but this won’t lead to any further output but will lead to the creation of further processes.

Therefore, the correct answer is A: 2 time, B: 4 times, C: 8 times

Q7.3	Consider the following program:
	`int main(){`
	`  printf("A");`
	`  fork();`
	`  fork();`
	`  printf("B");`
	`  fork();`
	`  printf("C");`
	`  fork();`
	`  return 0;`
	`}`
	How many times will the letters “A”, “B” and “C” be printed?

(a)	**A: 1 time, B: 4 times, C: 8 times**
(b)	A: 1 time, B: 8 times, C: 16 times
(c)	A: 2 time, B: 4 times, C: 8 times
(d)	A: 2 time, B: 8 times, C: 16 times

Feedback:

When a fork() system call is executed, it creates a copy of that process (the parent process). Execution continues in both processes (parent & child) at the point following the return from fork() call.
The main idea is that the number of times “a letter” is printed is equal to the number of processes that will exist at that point where the letter is printed.

- At the start there is only one process, so ‘A’ will be printed only once because it is executed before making the first fork() system call.
- After the first and second fork(), there are 4 processes. ‘ B‘ will be printed 4 times.
- After the third fork(), there are 8 processes. ‘C’ will be printed 8 times.
- There is a further fork() system call but this won’t lead to any further output but will lead to the creation of further processes.

Therefore, the correct answer is A: 1 time, B: 4 times, C: 8 times

Q8.1	Which of the following statements about the JVM stack are true?
(a)	Slot 0 always contains a reference to ‘this’.
(b)	The slots in the calling method’s stack frame are accessible to the callee method.
(c)	The value of each parameter in a method call is stored in the stack frame.
(d)	Local variables (of the method) are stored in the stack frame.
(e)	A local variable or parameter can use two slots in the stack frame.

Feedback:
Slot 0 is used to store a reference to ‘this’- but this is not true for ‘static’ methods which do not have an ‘instance’.
Each stack frame is isolated from each other. Therefore a method cannot access the parameters or local variables (or anything else) of the calling method.
Parameters and local variables are allocated slot(s) in the stack frame. Usually this is one slot but ‘long’ and ‘double’ values occupy two slots.

Q8.2	Which of the following statements about the JVM stack are true?
(a)	When a method is called a new stack frame is created.
(b)	The JVM stack can be infinitely large.
(c)	A value of type ‘double’ uses two slots in the stack frame.
(d)	When a method returns, the values in its stack frame are accessible to the calling method.
(e)	Slot 0 always contains a reference to ‘this’.

Feedback:
A new stack frame is created for every method call.
The size of the JVM stack is limited by the memory available and allocated.
Slot 0 is used to store a reference to ‘this’- but this is not true for ‘static’ methods which do not have an ‘instance’.
Each stack frame is isolated from each other. Therefore a method cannot access the parameters or local variables (or anything else) of the calling method.
Parameters and local variables are allocated slot(s) in the stack frame. Usually this is one slot but ‘long’ and ‘double’ values occupy two slots.

Q8.3	Which of the following statements about the JVM stack are true?
(a)	A new stack frame is created for every method call.
(b)	For ‘static’ methods the slot 0 contains a reference to ‘this’.
(c)	Values of type ‘long’ occupy two slots in the stack frame.
(d)	A called method can access the slots in the calling method’s stack frame.
(e)	The values in a method’s stack frame are accessible to the recursive calls of the same method.

Feedback:
Every method call creates a new stack frame.
Each stack frame is isolated from each other. Therefore a method cannot access the parameters or local variables (or anything else) of the calling method. Similarly, a new call leads to a new stack frame being created.
Slot 0 is used to store a reference to ‘this’- but this is not true for ‘static’ methods which do not have an ‘instance’.
Parameters and local variables are allocated slot(s) in the stack frame. Usually this is one slot but ‘long’ and ‘double’ values occupy two slots.

Q9.1

Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	8 ms
P2	3 ms	4 ms
P3	5 ms	6 ms
P4	7 ms	2 ms

What is the **Average Waiting Time** using **SRTF** CPU scheduling policy?

(a) **3.75 ms**

(b) 6.00 ms

(c) 5.75 ms

(d) 6.25 ms

Feedback:

For the given set of processes, here is the SRTF schedule:

P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_4	P_4	P_1	P_1	P_1	P_1	P_1	P_3	P_3	P_3	P_3	P_3	P_3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Waiting Time for P_1 = (0 – 0 + 9 – 3) = 6 ms

Waiting Time for P_2 = (3 – 3) = 0 ms

Waiting Time for P_3 = (14 – 5) = 9 ms

Waiting Time for P_4 = (7 – 7) = 0 ms

Average Waiting Time = (6 + 0 + 9 + 0) / 4 = **3.75 ms**.

Q9.2

Consider the following set of processes:

Processes	Arrival Time	Burst Time	Priority
P1	0 ms	8 ms	3
P2	3 ms	4 ms	2
P3	5 ms	6 ms	1
P4	7 ms	2 ms	2

What is the **Average Waiting Time** using **Preemptive Priority** CPU scheduling policy? A lower priority number indicates higher priority. Ties are broken using FCFS order.

(a) **6.00 ms**

(b) 3.75 ms

(c) 5.75 ms

(d) 6.25 ms

Feedback:

For the given set of processes, here is the Preemptive Priority schedule:

P_1	P_1	P_1	P_2	P_2	P_3	P_3	P_3	P_3	P_3	P_3	P_2	P_2	P_4	P_4	P_1	P_1	P_1	P_1	P_1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Waiting Time for P_1 = (0 – 0 + 15 – 3) = 12 ms
Waiting Time for P_2 = (2 – 2 + 11 – 5) = 6 ms
Waiting Time for P_3 = (5 – 5) = 0 ms
Waiting Time for P_4 = (13 – 7) = 6 ms
Average Waiting Time = (12 + 6 + 0 + 6) / 4 = **6.00 ms**.

Q9.3

Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	8 ms
P2	3 ms	4 ms
P3	5 ms	6 ms
P4	7 ms	2 ms

What is the **Average Waiting Time** using **Round Robin** CPU scheduling policy?
Time Quantum is given as 4 ms and new processes are added to the tail of the queue.

(a) **5.75 ms**

(b) 3.75 ms

(c) 6.00 ms

(d) 6.25 ms

Feedback:

For the given set of processes, here is the Round Robin schedule:

P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_1	P_1	P_1	P_1	P_3	P_3	P_3	P_3	P_4	P_4	P_3	P_3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Waiting Time for P_1 = (0 – 0 + 8 – 4) = 4 ms
Waiting Time for P_2 = (4 – 3) = 1 ms
Waiting Time for P_3 = (12 – 5 + 18 – 16) = 9 ms
Waiting Time for P_4 = (16 – 7) = 9 ms
Average Waiting Time = (4 + 1 + 7 + 9) / 4 = **5.75 ms**.

Q10.1

Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	8 ms
P2	4 ms	6 ms
P3	5 ms	2 ms
P4	8 ms	4 ms

What is the **Average Turnaround Time** using **SRTF** CPU scheduling policy?

(a) **8.50 ms**

(b) 11.00 ms

(c) 10.75 ms

(d) 11.25 ms

Feedback:

For the given set of processes, here is the SRTF schedule:

P_1	P_1	P_1	P_1	P_1	P_3	P_3	P_1	P_1	P_1	P_4	P_4	P_4	P_4	P_2	P_2	P_2	P_2	P_2	P_2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turnaround Time for P_1 = (10 – 0) = 10 ms

Turnaround Time for P_2 = (20 – 4) = 16 ms

Turnaround Time for P_3 = (7 – 5) = 2 ms

Turnaround Time for P_4 = (14 – 8) = 6 ms

Average Turnaround Time = (10 + 16 + 2 + 6) / 4 = **8.50 ms**.

Q10.2

Consider the following set of processes:

Processes	Arrival Time	Burst Time	Priority
P1	0 ms	8 ms	3
P2	3 ms	4 ms	2
P3	5 ms	6 ms	1
P4	7 ms	2 ms	2

What is the **Average Turnaround Time** using **Preemptive Priority** CPU scheduling policy? A lower priority number indicates higher priority. Ties are broken using FCFS order.

(a) **11.00 ms**

(b) 8.50 ms

(c) 10.75 ms

(d) 11.25 ms

Feedback:

For the given set of processes, here is the Preemptive Priority schedule:

P_1	P_1	P_1	P_2	P_2	P_3	P_3	P_3	P_3	P_3	P_3	P_2	P_2	P_4	P_4	P_1	P_1	P_1	P_1	P_1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turnaround Time for P_1 = (20 – 0) = 20 ms

Turnaround Time for P_2 = (13 – 3) = 10 ms

Turnaround Time for P_3 = (11 – 5) = 6 ms

Turnaround Time for P_4 = (15 – 7) = 8 ms

Average Turnaround Time = (20 + 10 + 6 + 8) / 4 = **11.00 ms**.

Q10.3

Consider the following set of processes:

Processes	Arrival Time	Burst Time
P1	0 ms	8 ms
P2	3 ms	4 ms
P3	5 ms	6 ms
P4	7 ms	2 ms

What is the **Average Turnaround Time** using **Round Robin** CPU scheduling policy? Time Quantum is given as 4 ms and new processes are added to the tail of the queue.

(a) **10.75 ms**

(b) 8.50 ms

(c) 11.00 ms

(d) 11.25 ms

Feedback:

For the given set of processes, here is the Round Robin schedule:

P_1	P_1	P_1	P_1	P_2	P_2	P_2	P_2	P_1	P_1	P_1	P_1	P_3	P_3	P_3	P_3	P_4	P_4	P_3	P_3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Turnaround Time for P_1 = (12 – 0) = 12 ms

Turnaround Time for P_2 = (8 – 3) = 5 ms

Turnaround Time for P_3 = (20 – 5) = 15 ms

Turnaround Time for P_4 = (18 – 7) = 11 ms

Average Turnaround Time = (12 + 5 + 15 + 11) / 4 = **10.75 ms**.