# End Semester Examination - Rubrics
# M.Tech (CSE/CSE-R) Refresher 2024
# Systems Programming

**Marks: 40 Marks** **Duration: 120 mins**

***Note:***
***i. Q1 is compulsory to attempt.***
***ii. From Q2 - Q8 attempt any five questions.***

---

## SECTION - A

**Q1. Multiple Choice Questions: (1*10 = 10 marks)**

**1. Shortest job first scheduling algorithm uses ___ data structure and has ____ time complexity.**

A. Array, O(n)
B. Min heap, O(n log n)
C. Queue, O(1)
D. None of the above

Answer - B

**2. Which command prints the entire array in Bash?**

A. echo ${name}
B. echo ${name[@]}
C. echo ${name[*]}
D. echo ${name[]}

Answer - B, C

**3. What range of values can a process's nice value take in Unix systems?**

A. 0 to 100
B. -20 to 19
C. 1 to 50
D. -10 to 10

Answer - B

**4. In which mode does a program not have direct access to memory and hardware?**

A. User mode
B. Kernel mode
C. Supervisor mode
D. Real mode

Answer - A

**5. What will be the output of the following code ?**

```
#include <stdio.h>
int main() {
    int a = 10, b = 20, c = 30;
    int *p1 = &a, *p2 = &b, *p3 = &c;
    int *arr[] = {p1, p2, p3};
    **(arr + 1) = 40;
    printf("%d\n", b);
    return 0;
}
```

A. 20
B. 30
C. 40
D. Compile-time error

Answer - C

**6. Which of the following actions can lead to a stack overflow?**

A. Dereferencing a null pointer.
B. Excessive or uncontrolled recursion.
C. Writing to read-only memory.
D. Accessing an out-of-bound array index.

Answer - B

**7. What are the Advantages of Protected Mode over Real Mode?**

A. User programs need to know about the underlying memory management.
B. Process memory isolation is not essential to multi-user systems.
C. Compiler and Assembler generate physical address, mapped by CPU to main memory.

D. Entire linear address space available to programs.

Answer - D

**8. Which command is used to change file permissions in Linux?**

A. chmod
B. chown
C. ls
D. Rm

Answer - A

**9. What is the effect of the copy-on-write (CoW) technique in process management?**

A. Processes share the same pages of memory until a modification occurs
B. Each process gets its own copy of memory pages at creation
C. Processes do not share any memory pages
D. It increases memory usage significantly

Answer - A

**10. Which of the following is an advantage of MLFQ over MLQ?**

A. More rigid structure.
B. Less starvation and more flexibility.
C. Simplified scheduling algorithms.
D. Permanent process assignment.

Answer - B

## SECTION - B

***Note:*** **From Q2 - Q8 attempt any five questions.**

**Q2. Define the following terms: (1*6 = 6 marks)**

***Assembler, Batch OS, System Programming, Kernel Mode, Multiprocessor Systems, Zombie process.***

Answer -

Assembler:

## Assembler

- **Definition:** An assembler is a program that converts assembly language into machine code. It takes basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.
- **Function:** An assembler enables software and application developers to access, operate, and manage a computer's hardware architecture and components.
- **Additional Role:** An assembler is sometimes referred to as the compiler of assembly language. It also provides the services of an interpreter.
- **Process:** The process involves taking an assembly language program and converting it into a machine language program.

## Batch OS:

- **Definition:** Batch operating systems execute batches of jobs sequentially. If one job is completed, only then is the next job scheduled.
- **Characteristics:** - **No Interaction:** Jobs are processed without user intervention. - **Increased Idleness:** Increased computer idleness between jobs. - **Decreased Throughput:** Lower overall throughput due to waiting times between job executions.
- **Example:** Early computing systems used batch processing for tasks like payroll, where all jobs were collected and processed together, leading to idle times between jobs.

## System Programming:

- **Definition:** System programming involves creating software that interacts directly with hardware and system resources, ensuring efficient operation of the computer system.
- **Importance:** It is essential for the development of system software like operating systems, compilers, and utilities. System programming enables the creation of robust, efficient, and high-performance software systems.

Kernel Mode:

- **Definition of Kernel Mode:** Kernel mode is a privileged mode of operation for the computer's operating system, allowing direct access to hardware and all system resources.

Multiprocessor Systems:

- **Definition:** Multiprocessor systems utilize two or more CPUs within a single computer system to perform tasks simultaneously.

- **Characteristics:** - **Parallel Processing:** Multiple processors work in parallel, increasing system performance. - **Increased Reliability:** If one processor fails, others can continue to operate. - **Scalability:** Additional processors can be added to improve performance.

- **Example:** Modern servers and high-performance computing systems use multiprocessor architectures to handle large volumes of data and complex computations efficiently.

Zombie Process:

A Zombie is created when a parent process does not use the wait system call after a child dies to read its exit status

zombie process is a terminated or completed process that remains in the process table. It will be there until its parent process clears it off.
And it does this by calling *wait()* call on its child and reading the exit value and other accounting information.

**Q3. Describe the following gcc options: (1*6 = 6 marks)**

***I.*** -O
***II.*** -I
***III.*** -g
***IV.*** -break
***V.*** -wall
***VI.*** -print

Answer -

a. **-O option:**

Various levels of optimization of the code

-O1 to -O3 are various degrees of optimization targeted for speed

-O3 is the highest level of optimization

If -O is added, then the code size is considered

-O0 means “no optimization

**b. -I Option:**

Tells gcc where to look for include files (.h).

**c. -g option:**

Includes debugging info in the generated object code. This info can later be used in gdb.

gcc allows to use -g with the optimization turned on (-O) in case there is a need to debug or trace the optimized code.

**d. -break Option:**

*break* sets breakpoints at places where you want the debugger to stop.

*break function-name* will set a breakpoint at the start of the function. You can set multiple breakpoints.

**e. -Wall Option:**

Shows most of the warnings related to possibly incorrect code.

Always a recommended option to save your bacon from some “hidden” bugs.

**f. -print Option:**

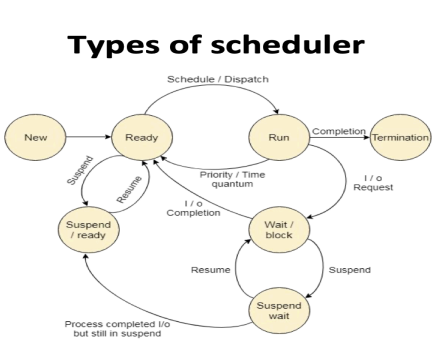
*print* allows you to display variables or any language expression. You may need a cast to display value correctly.

**Q4.** **(3+3 = 6 marks)**

**I. List out the differences between Fork() and Vfork().**

| Fork() | Vfork() |
|---|---|
| In fork() system call, child and parent process have separate memory space. | While in vfork() system call, child and parent process share same address space. |
| The child process and parent process gets executed simultaneously. | Once child process is executed then parent process starts its execution. |
| Child process does not suspend parent process execution in fork() system call. | Child process suspends parent process execution in vfork() system call. |
| Page of one process is not affected by page of other process | Page of one process is affected by page of other process. |
| There is wastage of address space. | There is no wastage of address space. |
| If child process alters page in address space, it is invisible to parent process. | If child process alters page in address space, it is visible to parent process. |

**II. Explain different types of Process Schedulers.**




- **Long-Term Scheduler (Job Scheduler)**
- Controls process admission to ready state.
- Runs infrequently
- Balances CPU-bound and I/O-bound processes for efficiency.
- Maintains optimal multiprogramming

- **Mid-Term Scheduler (Swapper)**
- Manages process swapping
- Runs occasionally
- Enhances performance and responsiveness
- Optimizes resource utilization

- **Short-Term Scheduler (CPU Scheduler)**
- Selects next process to execute
- Runs frequently
- Affects system responsiveness
- Minimizes waiting time and maximizes CPU usage

## Types of Process Schedulers

**Long Term or Job Scheduler**
It brings the new process to the 'Ready State'. It controls the Degree of Multi-programming, i.e., the number of processes present in a ready state at any point in time.

**Short-Term or CPU Scheduler**
It is responsible for selecting one process from the ready state for scheduling it on the running state.

**Medium-Term Scheduler**
It is responsible for suspending and resuming the process. It mainly does swapping(moving processes from main memory to disk and vice versa).

**Q5.** **(3+3 = 6 marks)**

I. **Are null and uninitialized pointers the same? If no, then explain the difference between them by providing an example(write c/c++/java program explaining the same).**

**Answer -**

No, null and uninitialized pointers are not the same. A null pointer is a pointer that explicitly points to the address 0, which means it does not point to any valid memory location. An uninitialized pointer, on the other hand, is a pointer that has not been assigned any value, meaning it points to a random memory location or holds a garbage value.

Null Pointer -> int* nullPtr = nullptr;

Uninitialized Pointer -> int* uninitPtr;

**(1+2 marks)**

II. **Discuss the malloc() function in C, including its syntax, purpose, and how it differs from the calloc() function. (3 marks)**

Malloc - The malloc() function stands for "memory allocation" and is used to dynamically allocate a single block of memory on the heap. The allocated memory is uninitialized, meaning it contains whatever values were already present at that location (garbage values).

malloc syntax -> void* malloc(size_t size);

Calloc - The calloc() function stands for "contiguous allocation" and is used to allocate multiple blocks of memory and initialize all bytes in the allocated memory to zero.

Calloc syntax -> void* calloc(size_t num, size_t size);

- malloc(): Allocates memory but does not initialize it. The memory contains garbage values.
- calloc(): Allocates memory and initializes all bytes to zero.
- malloc(): Takes a single parameter - the total number of bytes to allocate.
- calloc(): Takes two parameters - the number of elements and the size of each element.

**Q6. (3+3 = 6 marks)**

I. **Discuss the Completely Fair Scheduler (CFS) and its key features.**

**Answer -**

**Completely Fair Scheduler (CFS)**

The CFS is designed to ensure that each runnable process gets a fair share of the CPU time. It does this by using a scheduling algorithm that assigns each process a proportion of the CPU based on its priority and the time it has already run.The current default scheduler in Linux is the Completely Fair Scheduler (CFS)

**Key Features:**

- Fairness
- Red-Black Tree Implementation

- configurable scheduling latency
- Reduced Complexity

**II. Explain wild pointers and write syntax in C/C++/java(any lang) for the same.**

**Answer -**

A wild pointer is a pointer that has not been initialized to point to a valid memory location. It may contain a random address, making dereferencing it unpredictable and potentially dangerous. Wild pointers can cause undefined behavior, such as segmentation faults, crashes, or corruption of data, as they might point to any arbitrary memory location.

Example :

```
#include <stdio.h>

int main() {

   int *ptr; // Wild pointer: not initialized

   int x = 10;

   printf("Value at ptr: %d\n", *ptr); // Undefined Behavior

   ptr = &x;

   printf("Value at ptr: %d\n", *ptr);  // Output = 10

 return 0;

}
```

**Q7. (3+3 = 6 marks)**

**I. Explain why we need a copy on Write(CoW) mechanism in fork system call.**

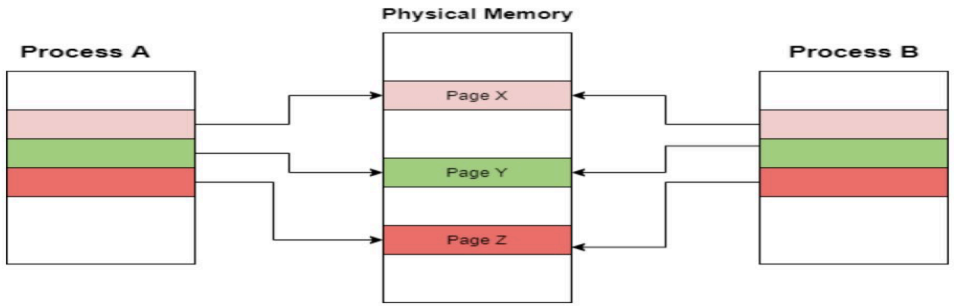
**Answer -**

Copy-on-Write(CoW) is mainly a resource management technique that allows the parent and child process to share the same pages of the memory initially.

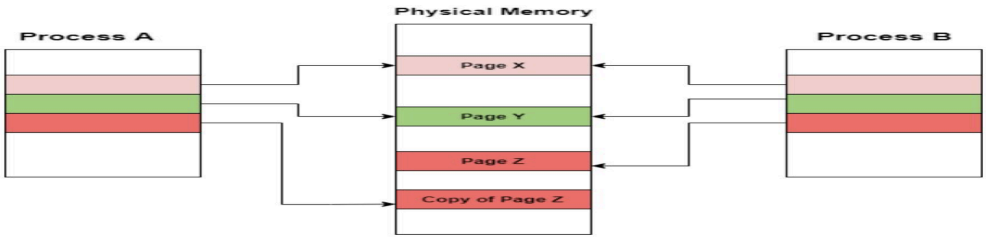
If any process either parent or child modifies the shared page, only then the page is copied.

These shared pages between parent and child process will be marked as copy-on-write which means that if the parent or child process will attempt to modify the shared pages then a copy of these pages will be created and the modifications will be done only on the copy of pages by that process and it will not affect other processes.

Process A creates a new process that is Process B, initially both these processes will share the same pages of the memory.




Now, let us assume that process A wants to modify a page in the memory. When the Copy-on-write(CoW) technique is used, only those pages that are modified by either process are copied; all the unmodified pages can be easily shared by the parent and child process.




**II. Describe the use of the following commands:**

A. df
B. du
C. top
D. ps
E. ifconfig
F. Find

Answer -

a. df -

- df
- • Description: Displays disk space usage.
- • Options:
- o -h: Human-readable sizes.

b. du

du
• Description: Estimates file and directory space usage.
• Options:
o -h: Human-readable sizes.
o -s: Display only a total for each argument.

c. top

top
• Description: Displays running processes.
• Options:
o -u user: Show processes for a specific user.

d. ps

ps
• Description: Reports a snapshot of current processes.
• Options:
o -aux: Shows detailed information about all processes.

e. ipconfig

ifconfig
• Description: Configures network interfaces (use ip on newer systems).
• Options:
o ifconfig -a: Display all interfaces.

f. find

find
• Description: Searches for files in a directory hierarchy.
• Options:
o -name: Search by filename.
o -type: Search by file type (e.g., d for directory, f for file).

**Q8.** **(3+3 = 6 marks)**

I. **Write an x86-64 assembly language program that counts the number of 1 bits in a 64-bit integer and stores the result in a variable. The program should:**
   A. Define a 64-bit integer in the data segment.
   B. Use a loop to examine each bit of the integer.
   C. Use appropriate instructions to count the number of 1 bits.
   D. Store the count in a separate variable (i.e. a memory declared in the data segment).

**If required, You can use the following instructions:-**

- 'test rax, 1' : tests if the least significant bit of rax is 1 by performing a bitwise AND with 1.
- 'shr rax, 1' shifts rax right by one bit.

**Answer:**

```
section .data
data dq 12
sum dq 0

section .text
global main

main:
    ; Register usage
    ; rax : bits being examined
    ; rcx : loop counter, 0-63
    ; rdx : sum of 1 bits
    mov rax, [data]
    xor rcx, rcx
    xor rdx, rdx
```

```
while:
    cmp rcx, 64       ; Compare loop counter with 64
    jge end_while     ; Jump to end_while if rcx is greater
than or equal to 64

    test rax, 1       ; Test if the LSB of rax is 1
    jz skip_add       ; If zero, jump to skip_add
    inc rdx           ; Increment rdx if LSB is 1

skip_add:
    shr rax, 1        ; Shift rax right by one bit
    inc rcx           ; Increment loop counter
    jmp while         ; Jump back to the beginning of the
loop

end_while:
    ; End of loop
    mov [sum], rdx    ; Store the result in sum
    xor eax, eax      ; Exit program
```

**II. Explain how variables are used in shell scripting, including how to declare variables, read user input, and perform basic arithmetic operations. Provide examples.**

Answer -

**Declare Variables:**

#!/bin/bash

# Declaring variables

name="Name"

age=20

**Print On console:**

echo "Name: $name"

echo "Age: $age"

**Read User input:**

```
#!/bin/bash

# Get user input

echo "Enter your name:"

read user_name

# Displaying the input

echo "Hello, $user_name!"
```

**Performing Basic Arithmetic Operations**

```
#!/bin/bash

# Declaring variables

a=10

b=5


# Performing arithmetic operations

sum=$(expr $a + $b)

diff=$(expr $a - $b)

prod=$(expr $a \* $b)

quot=$(expr $a / $b)


# Displaying results

echo "Sum: $sum"

echo "Difference: $diff"

echo "Product: $prod"
```

```
echo "Quotient: $quot"
```