INDEX

FILE MANAGEMNT

1. Write a C program to create a new text file and write "Hello, World!" to it?
2. Develop a C program to open an existing text file and display its contents?
3. 3Implement a C program to create a new directory named "Test" in the current directory?
4. Write a C program to check if a file named "sample.txt" exists in the current directory?
5. Develop a C program to rename a file from "oldname.txt" to "newname.txt"?
6. Implement a C program to delete a file named "delete_me.txt"?
7. Write a C program to copy the contents of one file to another?
8. Develop a C program to move a file from one directory to another?
9. Implement a C program to list all files in the current directory?
10. Write a C program to get the size of a file named "file.txt"?
11. Develop a C program to check if a directory named "Test" exists in the current directory?
12. Implement a C program to create a new directory named "Backup" in the parent directory?
13. Write a C program to recursively list all files and directories in a given directory?
14. Develop a C program to delete all files in a directory named "Temp"?
15. Implement a C program to count the number of lines in a file named "data.txt"?
16. Write a C program to append "Goodbye!" to the end of an existing file named "message.txt"?
17. Implement a C program to change the permissions of a file named "file.txt" to read-only?
18. Write a C program to change the ownership of a file named "file.txt" to the user "user1"?
19. Develop a C program to get the last modified timestamp of a file named "file.txt"?
20. Implement a C program to create a temporary file and write some data to it?
21. Write a C program to check if a given path refers to a file or a directory?
22. Develop a C program to create a hard link named "hardlink.txt" to a file named "source.txt"?

23. Implement a C program to read and display the contents of a CSV file named "data.csv"?
24. Write a C program to get the absolute path of the current working directory?
25. Develop a C program to get the size of a directory named "Documents"?
26. Implement a C program to recursively copy all files and directories from one directory to another?
27. Write a C program to get the number of files in a directory named "Images"?
28. Develop a C program to create a FIFO (named pipe) named "myfifo" in the current directory?
29. Implement a C program to read data from a FIFO named "myfifo"?
30. Write a C program to truncate a file named "file.txt" to a specified length?
31. Develop a C program to search for a specific string in a file named "data.txt" and display the line number(s) where it occurs?
32. Implement a C program to get the file type (regular file, directory, symbolic link, etc.) of a given path?
33. Write a C program to create a new empty file named "empty.txt"?
34. Develop a C program to get the permissions (mode) of a file named "file.txt"?
35. Implement a C program to recursively delete a directory named "Temp" and all its contents?
36. Write a C program to read and display the first 10 lines of a file named "log.txt"?
37. Develop a C program to read data from a text file named "input.txt" and write it to another file named "output.txt" in reverse order?
38. Implement a C program to create a new directory named with the current date in the format "YYYY-MM-DD"?
39. Write a C program to read and display the contents of a binary file named "binary.bin"?
40. Develop a C program to get the size of the largest file in a directory?
41. Implement a C program to check if a file named "data.txt" is readable?
42. Write a C program to create a new directory named "Logs" and move all files with the ".log" extension into it?
43. Develop a C program to check if a file named "config.ini" is writable?
44. Implement a C program to read the contents of a text file named "instructions.txt" and execute the instructions as shell commands?
45. Write a C program to get the number of hard links to a file named "file.txt"?

46. Develop a C program to copy the contents of all text files in a directory into a single file named "combined.txt"?
47. Implement a C program to recursively calculate the total size of all files in a directory and its subdirectories?
48. Write a C program to get the number of bytes in a file named "data.bin"?
49. Develop a C program to create a new directory named with the current timestamp in the format "YYYY-MM-DD-HH-MM-SS"?
50. Write a C program to create a new directory named "Documents" in the current directory?
51. Develop a C program to check if a file named "file.txt" exists in the current directory?
52. Implement a C program to open a file named "data.txt" in read mode and display its contents?
53. Write a C program to create a new text file named "output.txt" and write "Hello, World!" to it?
54. Develop a C program to delete a file named "delete_me.txt" from the current directory?
55. Implement a C program to rename a file from "old_name.txt" to "new_name.txt"?
56. Write a C program to copy the contents of one file to another file?
57. Develop a C program to move a file named "file.txt" to a directory named "Backup"?
58. Implement a C program to list all files and directories in the current directory?
59. Write a C program to get the size of a file named "data.txt"?
60. Develop a C program to create a new directory named "Pictures" in the parent directory?
61. Develop a C program to count the number of lines in a file named "log.txt"?
62. Implement a C program to append "Goodbye!" to the end of an existing file named "message.txt"?
63. Write a C program to create a symbolic link named "link.txt" to a file named "target.txt"?
64. Develop a C program to change the permissions of a file named "file.txt" to read-only?
65. Implement a C program to change the ownership of a file named "file.txt" to the user "user1?
66. Write a C program to get the last modified timestamp of a file named "file.txt"?
67. Develop a C program to create a temporary file and write some data to it?
68. Implement a C program to get the size of a file named "image.jpg"?

69. Write a C program to create a new text file named "notes.txt" and write multiple lines of text to it?
70. Develop a C program to count the number of words in a file named "essay.txt"?
71. Write a C program to create a symbolic link named "shortcut.txt" to a file named "target.txt"?
72. Develop a C program to change the permissions of a file named "important.doc" to read and write for the owner only?
73. Write a C program to get the last access time of a file named "data.txt"?
74. Develop a C program to read and display the contents of a CSV file named "data.csv"?
75. Implement a C program to truncate a file named "file.txt" to a specified length?
76. Write a C program to search for a specific word in a file named "data.txt" and display the line number(s) where it occurs?
77. Develop a C program to get the file type (regular file, directory, symbolic link, etc.) of a given path?
78. Implement a C program to read and display the contents of a binary file named "binary.bin"?
79. Implement a C program to create a new directory named "Logs" and move all files with the ".log" extension into it?
80. Write a C program to check if a file named "config.ini" is writable?
81. Develop a C program to read the contents of a text file named "instructions.txt" and execute the instructions as shell commands?
82. Develop a C program to read data from a binary file named "data.bin" and display it in hexadecimal format?
83. Develop a C program to recursively copy all files and directories from one directory to another?
84. Develop a C program to get the file type (regular file, directory, symbolic link, etc.) of a given path?
85. What are the types of files present in Linux OS?
86. How do IPC Objects ,named pipes, be accessed?
87. User space application sends request to Hardware by using which I/O calls?
88. Why are basic I/O calls called universal I/O calls?
89. What is the content of the Inode Object?
90. In which Object information of the file get stored?
91. Kernel uses which object to represent a file?
92. Can we access the file information present in inode from the user application if yes, then How?

93. Which system calls are used to access the file information?
94. ls command internally invoked which system call to access the file information?
95. What happens after the kernel finds its inode object?
96. What are the contents of the file object?
97. Kernel uses which object to represent the open files?
98. What is the difference between the primary data and other members of the file object?
99. Where does the file object base address get stored?
100. When the fd table is created and what is the size of the fd table?
101. When is the file object created?
102. What does open() returns?
103. When a file opens for the first time by using open() calls in your program, what is returns?
104. Why do read() system calls need access to file objects?
105. Which system call do we use to change the cursor position without reading and writing on a file?
106. Can we open the same file from multiple processes? Explain the memory segment in kernel space?
107. Kernel uses which object to represent a file?
108. Is there any limit on no. of files that can be opened from the program?
109. What are the standard I/O calls? Give some examples and what is the alternate name of standard I/O call?
110. What are the basic I/O calls? Give some examples and what is the alternate name of basic I/O call?
111. What is the difference between basic I/O calls and standard I/O calls?
112. Other than the basic I/O and standard I/O calls,Is there any method to access the file ?
113. How do user space applications get access to the content of inode objects?
114. How do you get access to kernel objects/kernel data structure/Information present in kernel space?

115. Which system calls are used to access the file object?
116. Which system calls are used to access the Inode object?
117. How to print the Hello world on the screen by using basic I/O calls?
118. Which system call do we use to duplicate the file descriptor?
119. What are the advantages of dup system calls?
120. Which object stores the ownership information of a file?
121. Which command is used to display the username corresponding ID?
122. How can we see the multiple user information in our system?
123. Which command is used to change the UID and GID?

MEMORY MANAGEMENT

1. What is memory management in system programming?
2. Define virtual memory.
3. Differentiate between physical memory and virtual memory.
4. What is the role of an operating system in memory management?
5. Explain the purpose of memory allocation.
6. Describe the significance of memory deallocation.
7. Define fragmentation in memory management.
8. What are the types of fragmentation?
9. Explain internal fragmentation.
10. Explain external fragmentation.
11. How is fragmentation managed in memory allocation?
12. Describe the concept of paging.
13. Explain segmentation.
14. What is the difference between paging and segmentation?
15. Define page table.
16. Define Memory Management Unit (MMU).
17. Explain the role of MMU in memory management.
18. Describe the translation lookaside buffer (TLB).
19. What is TLB miss? How is it handled?
20. Discuss the working principle of MMU.
21. Explain the concept of address translation in MMU.
22. How does MMU support virtual memory?
23. Describe the process of page table traversal in MMU.
24. What is page fault handling in MMU?
25. Explain the page replacement algorithms used in MMU.
26. Define page replacement algorithms.
27. Describe the FIFO page replacement algorithm.
28. Discuss the optimal page replacement algorithm.
29. Explain the LRU (Least Recently Used) page replacement algorithm.

30. What is the clock page replacement algorithm?
31. Discuss the advantages and disadvantages of each page replacement algorithm.
32. Compare and contrast different page replacement algorithms.
33. Explain the working of the NRU (Not Recently Used) page replacement algorithm.
34. Describe the working of the Second Chance page replacement algorithm.
35. Discuss the enhancements to basic page replacement algorithms.
36. Define segmentation in memory management.
37. Explain the benefits of segmentation.
38. What are the disadvantages of segmentation?
39. Describe the implementation of segmentation.
40. Discuss segmentation fault and its causes.
41. Explain the concept of segment registers.
42. What is a segment table?
43. How does segmentation support protection and sharing of memory?
44. Discuss the segmentation with paging approach.
45. Compare and contrast segmentation with paging.
46. Define memory fragmentation.
47. Explain the causes of memory fragmentation.
48. How does memory fragmentation affect system performance?
49. Discuss the techniques to reduce memory fragmentation.
50. Explain compaction as a technique for reducing fragmentation.
51. What is memory compaction?
52. Describe the working of memory compaction algorithms.
53. Discuss the challenges in implementing memory compaction.
54. Explain memory fragmentation in the context of embedded systems.
55. How does memory allocation impact memory fragmentation
56. Define memory mapping.
57. Explain the purpose of memory mapping.
58. Describe the memory mapping techniques.
59. What is memory-mapped I/O?

60. Explain memory-mapped files.
61. Discuss the advantages of memory mapping.
62. What are the drawbacks of memory mapping?
63. How does memory mapping improve performance?
64. Explain memory-mapped graphics.
65. Discuss memory mapping in embedded systems.
66. Define cache memory.
67. Explain the purpose of cache memory.
68. Describe the types of cache memory.
69. Discuss the cache coherence problem.
70. Explain cache replacement policies.
71. What is cache associativity?
72. Describe the working of cache memory.
73. Explain the cache hit and cache miss.
74. Discuss the importance of cache memory in memory management.
75. How does cache memory relate to memory hierarchy?
76. Define memory protection.
77. Explain the need for memory protection.
78. Describe the techniques for implementing memory protection.
79. What is segmentation fault?
80. Explain the role of privilege levels in memory protection.
81. Discuss the mechanism of memory protection in modern operating systems.
82. What are the security implications of memory protection?
83. Explain the concept of memory isolation.
84. Discuss the challenges in implementing memory protection.
85. How does memory protection contribute to system security?
86. Write a C program to demonstrate dynamic memory allocation using malloc().
87. Implement a C program to allocate memory for an array dynamically using calloc().
88. Write a C program to resize dynamically allocated memory using realloc().
89. Develop a program in C to allocate memory for a linked list node dynamically.

90. Implement a C program to simulate memory allocation using the first-fit algorithm.
91. Write a C program to simulate memory allocation using the best-fit algorithm.
92. Develop a C program to simulate memory allocation using the worst-fit algorithm.
93. Implement a C program to simulate memory allocation using the next-fit algorithm.
94. Write a C program to implement a simple memory allocator using the buddy system.
95. Develop a C program to implement a memory allocator using a custom memory management algorithm.
96. Write a C program to demonstrate memory mapping using mmap().
97. Implement a C program to read from and write to a memory-mapped file.
98. Develop a C program to demonstrate shared memory usage using shmget() and shmat().
99. Write a C program to create a shared memory segment and synchronize access using semaphores.
100. Implement a C program to simulate page replacement algorithms like FIFO, LRU, and optimal.

Process Management

1. Explain the concept of process creation in operating systems.
2. Differentiate between the fork() and exec() system calls.
3. Write a C program to demonstrate the use of fork() system call.
4. What is the purpose of the wait() system call in process management?
5. Describe the role of the exec() family of functions in process management.
6. Write a C program to illustrate the use of the execvp() function.
7. How does the vfork() system call differ from fork()?
8. Discuss the significance of the getpid() and getppid() system calls.
9. Explain the concept of process termination in UNIX-like operating systems.
10. Write a program in C to create a child process using fork() and print its PID.
11. Describe the process hierarchy in UNIX-like operating systems.
12. What is the purpose of the exit() function in C programming?
13. Explain how the execve() system call works and provide a code example.
14. Discuss the role of the fork() system call in implementing multitasking.
15. Write a C program to create multiple child processes using fork() and display their PIDs.
16. How does the exec() system call replace the current process image with a new one?
17. Explain the concept of process scheduling in operating systems.
18. Describe the role of the clone() system call in process management.
19. Write a program in C to create a zombie process and explain how to avoid it.
20. Discuss the significance of the setuid() and setgid() system calls in process man
21. Explain the concept of process groups and their significance in UNIX-like operating systems.

22. Write a C program to demonstrate the use of the waitpid() function for process synchronization.
23. Discuss the role of the execle() function in the exec() family of calls.
24. Describe the purpose of the nice() system call in process scheduling.
25. Write a program in C to create a daemon process.
26. Explain the role of the getpid() and getppid() functions in process management.
27. Discuss the difference between the fork() and clone() system calls.
28. Write a C program to demonstrate the use of the system() function for executing shell commands.
29. Explain the concept of process states in UNIX-like operating systems.
30. Describe the purpose of the chroot() system call and provide an example.agement.
31. Discuss the role of the execv() function in the exec() family of calls.
32. Write a C program to create a process using fork() and pass arguments to the child process.
33. Explain the significance of process identifiers (PIDs) in process management.
34. Discuss the concept of orphan processes and how they are handled in UNIX-like operating systems.
35. Write a program in C to demonstrate process synchronization using semaphores.
36. Describe the concept of process priority and how it is managed in operating systems.
37. Explain the purpose of the fork() system call in creating copy-on-write (COW) processes.
38. Discuss the role of the execvp() function in searching for executable files.
39. Write a C program to demonstrate the use of the execvpe() function.
40. Explain the concept of process context switching and its impact on system performance.
41. Discuss the role of the clone() system call in creating threads in Linux.

42. Write a C program to create a process group and change its process group ID (PGID).
43. Explain the difference between process creation using fork() and pthread_create().
44. Discuss the significance of the execvp() function in searching for executables in the PATH environment variable.
45. Write a program in C to demonstrate inter-process communication (IPC) using shared memory.
46. Describe the role of the fork() system call in implementing the shell's job control.
47. Explain the purpose of the execlp() function and provide an example.
48. Discuss the significance of the setpgid() system call in managing process groups.
49. Write a C program to create a child process using vfork() and demonstrate its usage.
50. Explain the concept of process priority inheritance and its importance in real-time systems.
51. Describe the role of the fork() system call in copy-on-write (COW) mechanism and its benefits.
52. Write a C program to create a pipeline between two processes using the pipe() system call.
53. Explain the concept of process scheduling policies such as FIFO, Round Robin, and Priority scheduling.
54. Discuss the significance of the execve() function in process creation and execution.
55. Write a program in C to demonstrate the use of the nice() system call for adjusting process priority.
56. Explain the concept of process swapping and its role in memory management.
57. Discuss the difference between the fork() and pthread_create() functions in terms of process/thread creation.
58. Describe the purpose of the prctl() system call in process management.
59. Write a C program to demonstrate the use of the clone() system call to create a thread.
60. Explain the concept of process preemption and its impact on system responsiveness.

61. Discuss the role of the exec functions in handling file descriptors during process execution.
62. Write a C program to create a child process using fork() and communicate between parent and child using pipes.
63. Explain the significance of process priorities and how they affect scheduling decisions.
64. Describe the process of process termination and the various ways it can occur.
65. Discuss the role of the exit status in process termination and how it can be retrieved by the parent process.
66. Write a C program to demonstrate process synchronization using the fork() and wait() system calls.
67. Explain the concept of process suspension and resumption using signals.
68. Discuss the role of process scheduling algorithms in determining the order of execution among processes.
69. Write a C program to create a process using fork() and change its scheduling policy using sched_setscheduler().
70. Explain the concept of process migration and its relevance in distributed systems.
71. Describe the role of process identifiers (PIDs) in process management and their uniqueness within the system.
72. Write a C program to create a child process using fork() and demonstrate inter-process communication (IPC) using shared memory.
73. Explain the concept of process tracing and its importance in debugging and monitoring.
74. Discuss the role of the fork() system call in creating copy-on-write (COW) processes and its impact on memory usage.
75. Describe the concept of process control blocks (PCBs) and their role in process management.

76. Write a C program to demonstrate the use of the prctl() system call to change process attributes.
77. Explain the concept of process scheduling classes and their significance in real-time operating systems.
78. Discuss the role of the vfork() system call in process creation and its differences from fork().
79. Describe the purpose of the sigaction() system call in handling signals in processes.
80. Write a C program to create a child process using fork() and demonstrate process synchronization using semaphores.
81. Explain the concept of process migration and its implications in distributed computing environments.
82. Write a C program to create a child process using fork() and demonstrate process communication using message queues.
83. Discuss the role of the sigprocmask() system call in managing signal masks for processes.
84. Describe the purpose of the setrlimit() system call in setting resource limits for processes.
85. Discuss the concept of process groups and their importance in job control and signal propagation.
86. Write a C program to create a child process using fork() and demonstrate process synchronization using condition variables.
87. Explain the role of the prlimit() system call in setting resource limits for processes.
88. Discuss the concept of process scheduling policies in multi-core systems and their implications.
89. Describe the role of the setpriority() system call in adjusting process priorities.
90. Write a C program to create a child process using fork() and demonstrate process communication using sockets.

91. Explain the concept of process scheduling latency and its impact on system responsiveness.

92. Write a C program to create a child process using fork() and demonstrate process synchronization using mutexes.

93. Discuss the role of the prlimit64() system call in setting resource limits for processes with 64-bit address space.

94. Describe the purpose of the sched_getaffinity() system call in querying the CPU affinity of a process.

95. Discuss the concept of process checkpointing and its relevance in fault tolerance and process migration.

96. Write a C program to create a child process using fork() and demonstrate process communication using named pipes (FIFOs).

97. Explain the significance of the /proc filesystem in providing information about processes in Linux.

98. Describe the purpose of the sched_setaffinity() system call in setting CPU affinity for processes.

99. Discuss the concept of process re-parenting and its implications in process management.

100. Write a C program to create a child process using fork() and demonstrate process communication using shared memory and semaphores

101. what is process and what is the difference between process and program?
102. what type of infomation pcb contains ?
103. Explain about memory segements ?
104. when fork invoked what are things will happen ?
105. what is the difference between ps and top?
106. what are the types of processes we have explain each process breifly?
107. what is the ppid of orphan process?
108. what is the difference between exit vs return?
109. In parent process can you access to the exit code of return value of child process (or) in parent process how do you block to get the id of child?
110. what is the difference between Zombie and Orphan process?
111. what is the use of fork() ?

112. define system call name some blocking system calls?
113. why do not we run program with in harddisk (or) why do we copy program to RAM
for execution?
114. how do you copy data from RAM to CPU registers ?
115. how do you copy data from CPU registers to RAM ?
116. explain about paging technique ?
117. why the no of pages are not fixed ?
118. explain about shared memory why we require shared memory?
119. How does multiple process can share data ?
120. explain the scenario when pages of process-1 and pages of process-2 are copied into same frames (or) explain the scenario where pages of multiple process are sharing the same physical frames(or) Can the page table of multiple process point to same
physical frames?
121. How child process uses the memory segment of parent process?
122. how the parent process and child process keep track of which instruction they are going to execute ?
123. how parent an child process executes different blocks of code after fork()?
124. explain write operation to pages shared to parent and child process?
explain about adress translation?
125. what adress cpu places an adress bus?
126. when virtual adress are generated ?
127. when physical adresss are generated from virtual adress ?
128. what is exec () family of calls and what is the need of exac() family of calls and what are those?
129. most of the system call why they return error?
130. what is the difference between Execl() and Execv()?
131. What is the difference between Execlp() and Execvp()?

Threads

1.Write a C program to create a thread that prints "Hello, World!"?

2.Modify the above program to create multiple threads, each printing its own message?

3.Develop a C program to create two threads that print numbers from 1 to 10 concurrently?

4.Implement a C program to create a thread that calculates the factorial of a given number?

5.Write a C program to create two threads that print their thread IDs?

6.Develop a C program to create a thread that prints the sum of two numbers?

7.Implement a C program to create a thread that calculates the square of a number?

8.Write a C program to create a thread that prints the current date and time?

9.Develop a C program to create a thread that checks if a number is prime?

10.Implement a C program to create a thread that checks if a given string is a palindrome?

11.Write a C program to create a thread that prints "Hello, World!" with thread synchronization?

12.Develop a C program to create two threads that print their thread IDs and synchronize their output?

13.Implement a C program to create a thread that generates random numbers and synchronizes access to a shared buffer?

14.Write a C program to create a thread that performs addition of two numbers with mutex locks?

15.Implement a C program to create two threads that increment and decrement a shared variable, respectively, using mutex locks?

16.Develop a C program to create a thread that reads input from the user and synchronizes access to shared resources?

17.Implement a C program to create a thread that prints prime numbers up to a given limit with mutex locks?

18.Implement a C program to create a thread that calculates the sum of Fibonacci numbers up to a given limit using dynamic programming with mutex locks?

19.Write a C program to create a thread that checks if a given year is a leap year using dynamic programming with mutex locks?

20.Write a C program to create a thread that checks if a given string is a palindrome using dynamic programming with mutex locks?

21.Implement a C program to create a thread that performs selection sort on an array of integers?

22.Develop a C program to create a thread that calculates the area of a triangle?

23.Write a C program to create a thread that calculates the sum of squares of numbers from 1 to 100?

24.Write a C program to create a thread that generates a random array of integers?

25.Implement a C program to create a thread that performs bubble sort on an array of integers?

26.Develop a C program to create a thread that calculates the greatest common divisor (GCD) of two numbers?

27.Write a C program to create a thread that calculates the sum of Fibonacci numbers up to a given limit?

28.Implement a C program to create a thread that calculates the sum of even numbers from 1 to 100?

29.Develop a C program to create a thread that calculates the factorial of a given number using iteration?

30.Write a C program to create a thread that checks if a given year is a leap year?

31.Implement a C program to create a thread that performs multiplication of two matrices?

32.Develop a C program to create a thread that calculates the average of numbers from 1 to 100?

33.Implement a C program to create a thread that generates a random string?

34.Write a C program to create a thread that checks if a given number is a perfect square?

35.Write a C program to create a thread that calculates the sum of digits of a given number?

36.Implement a C program to create a thread that calculates the factorial of a given number using recursion?

37.Develop a C program to create a thread that finds the maximum element in an array?

38.Write a C program to create a thread that sorts an array of strings?

39.Implement a C program to create a thread that calculates the square root of a number?

40.Develop a C program to create a thread that calculates the average of numbers in an array?

41.Write a C program to create a thread that generates a random password?

42.Implement a C program to create a thread that checks if a number is even or odd?

43.Develop a C program to create a thread that calculates the sum of elements in an array?

44.Write a C program to create a thread that calculates the factorial of numbers from 1 to 10?

45.Implement a C program to create a thread that calculates the area of a rectangle?

46.Develop a C program to create a thread that generates random numbers?

47.Implement a C program to create a thread that sorts an array of integers?

48.Write a C program to create a thread that searches for a given number in an array?

49.Develop a C program to create a thread that reverses a given string?

50.Implement a C program to create a thread that reads input from the user?

51.Write a C program to create a thread that performs addition of two matrices?

52.Develop a C program to create a thread that calculates the length of a given string?

53.Write a C program to create two threads using pthreads library. Each thread should print "Hello, World!" along with its thread ID?

54.Modify the previous program to pass arguments to the threads and print those arguments along with the thread ID?

55.Write a C program to demonstrate thread synchronization using mutex locks. Create two threads that increment a shared variable using mutex locks to ensure proper synchronization?

56.Extend the previous program to use semaphore instead of mutex locks for thread synchronization?

57.Write a C program to implement the producer-consumer problem using pthreads. Create two threads - one for producing items and another for consuming items from a shared buffer?

58.Implement a multithreaded file copy program in C. Create multiple threads to read from one file and write to another file concurrently?

59.Write a C program to demonstrate thread cancellation. Create a thread that runs an infinite loop and cancels it after a certain condition is met from the main thread?

60.Write a C program to implement a reader-writer lock using pthreads. Create multiple reader threads and writer threads and ensure proper synchronization using the reader-writer lock?

61.Write a C program to create a thread that prints the even numbers between 1 and 20?

62.Develop a C program to create two threads that print odd and even numbers alternately?

63.Implement a C program to create a thread that calculates the sum of squares of numbers from 1 to 10?

64.Write a C program to create a thread that calculates the product of numbers from 1 to 5?

65.Develop a C program to create a thread that prints the first 10 terms of the Fibonacci sequence?

66.Write a C program to create a thread that finds the maximum element in a given array?

67.Implement a C program to create a thread that prints the ASCII values of characters in a given string?

68.Develop a C program to create a thread that calculates the sum of all prime numbers up to a given limit?

69.Write a C program to create a thread that calculates the area of a circle using a given radius?

70.Develop a C program to create a thread that calculates the average of a given array of floating-point numbers?

71.Implement a C program to create a thread that prints the factors of a given number?

72.Develop a C program to create a thread that prints the English alphabet in uppercase?

73.Implement a C program to create a thread that checks if a given number is a perfect square?

74.Implement a C program to create a thread that checks if a given number is divisible by another given number?

75.Develop a C program to create a thread that prints the multiplication table of a given number up to a given limit?

76.Develop a C program to create a thread that prints the current system time?

77.Write a C program to create two threads that increment and decrement a shared variable, respectively, using mutex locks?

78.Develop a C program to create a thread that prints numbers from 10 to 1 in descending order using mutex locks?

79.why we are using pthread_create instead of clone() for creating threads?

80.why a stack grows?

81.what segments are shared by multiple threads within a process?

82.can you fetch the thread entry point return value in your main thread?

83.what happens when main function is invoked?

84.what happens when cpu stops executing?

85.During the context switch, which instruction will be usedto copy the contents of cpu register to your context area of PCB?

86.How do you create separate process?

87.How do server create separate thread?

88.Advantage of Thread over process?

89.Advantage of process over Thread?

90.How do you overcome this updation or synchronization issue.when the multiple threads are trying to access the global variables?

91.How much cpu time is given to user space thread and kernel space thread?

92.Explain POSIX and system-v difference?

93.what are the points to remember when mutex locks are used to protect the critical section?

94.By using mutex_lock what you are acheiving?

95.Difference between mutex locks and semaphore?

96.Explain the variants of pthread_mutex_lock?

97.How to create a thread?

98.Explain the compilation of a thread?

99.what are the arguments of pthread_create?

100.Explain the return value of thread?

101.Explain the working of pthread_mutex_trylock()?

102.Explain the application of pthread_mutex_timelock()?

103.what is mutual exclusion?

SIGNALS

THEORITICAL QUESTIONS

1. Can you explain the concept of signals in programming? How are they used in Unix-like operating systems?
2. What are software interrupts and hardware interrupts and mention potential issues when dealing with them?
3. What is synchronous signal and asynchronous signal and how the process can be used for both?
4. Who is responsible for generating signals?
5. What is signal handler?
6. Which system call is used to send a signal to the process?
7. Write a program to send a signal to itself (same process)?
8. Explain the default action associated with the SIGKILL signal?
9. How does a process handle a signal while it is executing in kernel mode?
10. Describe the behaviour of a process when it receives a SIGSEGV signal?
11. What is the role of the sigwait() function in signal handling?
12. Explain the concept of signal correlation in a distributed environment?
13. Explain how a process handles a signal while it is in the ready state?
14. What is the role of the sigqueue() function in signal handling?
15. Describe the interaction between signals and IPC mechanisms in Unix-like systems?
16. Explain how a process can determine the priority of a received signal?
17. What is the role of the sigaltstack() function in signal handling?
18. Explain how a process can determine whether a signal was sent by the kernel or another process?
19. Describe the interaction between signals and system calls in Unix-like systems?
20. How does a process handle a signal while it is waiting for a semaphore?
21. Describe the difference between a signal handler and a signal mask?
22. How does a process handle a signal while it is in the zombie state?
23. What are the advantages and disadvantages of using signals for interprocess communication?
24. Explain how a process handles a signal while it is in a sleep state.
25. How does a process handle a signal while it is in a critical section?
26. What are some of the challenges associated with signal handling in multi-threaded programs?
27. What is a race condition? Explain how it might occur in the context of signals?
28. Discuss how a deadlock situation can be caused or resolved by signal handling?
29. How can you use signals to force a process to dump core?
30. What are the implications of using longjmp() and setjmp() in signal handlers?
31. Difference between termination and suspending of a signal
32. How CPU access the device register?
33. what is IRQ line?
34. How do you find out unique value for each IRQ line?
35. when interrupt occurs?

36. when are exceptions when they occur?
37. How does kernel informs to the parent that it(child) process is terminated?
38. Which member of PCB contains the information about the signals?
39. How can we replace SIGDEL with SIGING
40. (or)
41. How do you modify signal behaviour table/signal disposition
42. why crash in program occurs?
43. How do you install signal handler in signal disposition table from user space?
44. How do you catch signal?
45. What does header file contains?
46. How do you clear the content of sa_mask?
47. Why signals are needed to be blocked?
48. Signal no 2's corresponding bit is set to 1? what do you meant by it?
49. How can we see the blocking signal information in PC?
50. How do I get access to some inf0ormation present in kernel space?
51. During the execution of signal handler, other than default signal. Can you block additional signal?
52. Explain the scenario where signal is blocked?
53. From user space application can you access (Read/write) signal mask present in your PCB?
54. Where is the process information present?
55. From userspace how do I access PCB information present in kernel space?
56. Can you create a proc virtual file system entry as user?
57. Without sending a signal can you invoke mysighand?

MULTIPLE-CHOICE QUESTIONS (MCQs)

1.Which of the following best describes a signal in system programming?

A) A hardware interrupt generated by the CPU
B) A software interrupt delivered to a process
C) A system call for inter-process communication
D) A method for thread synchronization

2. In a Unix-like operating system, what happens when a process receives a SIGSTOP signal?

A) The process terminates immediately.

B) The process is paused and can be resumed later.

C) The process is terminated, but its resources are not released.

D) The process is suspended until a SIGCONT signal is received.

3. Which of the following statements about signal handling in multithreaded programs is true?

A) Each thread in a process has its own signal handler.

B) Signals can only be handled by the main thread of a process.

C) Signal handling behavior is undefined in multithreaded programs.

D) Signals are always delivered to the thread that created them.

4. In signal handling, what does the SA_NODEFER flag indicate when used with the sigaction() function?

A) It specifies that the signal should not be blocked during its handler's execution.

B) It specifies that the signal should be delivered only once.

C) It specifies that the default action for the signal should be taken.

D) It specifies that the signal should be ignored.

5. The SIGPIPE signal is sent to a process when:

A) It encounters an illegal instruction

B) It writes to a pipe with no one to read from it

C) It receives a termination request

D) It receives a segmentation fault

6. Which signal is typically sent to a parent process when a child process terminates?

A) SIGTERM

B) SIGCHLD

C) SIGINT

D) SIGSEGV

7. Which function is used to establish a signal handler for a specific signal in C programming?

A) signal()

B) sigaction()

C) kill()

D) raise()

PROGRAMMING QUESTIONS

1. Write a C program to catch and handle the SIGINT signal.
2. Implement a C program to send a custom signal to another process.
3. Create a C program to ignore the SIGCHLD signal temporarily.
4. Write a program to block the SIGTERM signal using sigprocmask().
5. Implement a C program to handle the SIGALRM signal using sigaction().
6. Write a C program to install a custom signal handler for SIGTERM?
7. Implement a program to handle the SIGSEGV signal (segmentation fault).
8. Create a program to handle the SIGILL signal (illegal instruction).
9. Write a program to handle the SIGABRT signal (abort).
10. Implement a C program to handle the SIGQUIT signal.
11. Write a program to handle the SIGTERM signal (termination request).
12. Write a program to handle the SIGTSTP signal (terminal stop).
13. Write a program to handle the SIGVTALRM signal (virtual timer expired).
14. Write a program to handle the SIGWINCH signal (window size change).
15. Implement a C program to handle the SIGXFSZ signal (file size limit exceeded).
16. Create a program to handle the SIGPWR signal (power failure restart).
17.
18. Write a program to handle the SIGSYS signal (bad system call).
19. Write a C program to handle the SIGIO signal (I/O is possible on a descriptor).
20. Implement a C program to handle the SIGINFO signal (status request from keyboard).
21. Create a C program to handle the SIGRTMIN signal (minimum real-time signal).
22. Implement a program to handle the SIGRTMAX signal (maximum real-time signal).
23. Write a program to handle the SIGABRT_ABORT signal (abort signal).
24. Create a C program to handle the SIGSEGV_SIGBUS signal (segmentation fault or bus error).
25. Implement a program to handle the SIGUSR1_SIGUSR2 signal (user-defined signal).
26. Create a C program to handle the SIGCONT_SIGSTOP signal (continue or stop executing).
27. Write a C program to catch and handle the SIGSEGV signal.
28. Write a program to demonstrate how to block signals using sigprocmask().
29. Write a program to implement a timer using signals.
30. Write a program to handle a real-time signal using sigqueue().
31. Why we use raise system call explain it programmatically

32. Write a program to handle SIGALRM (alarm clock) signal for implementing a timeout mechanism in system programming.
33. Write a c program on pause system call
34. Create a C program to handle the SIGTRAP signal (trace/breakpoint trap).
35. Write a C program to handle the SIGIO signal (I/O is possible on a descriptor).
36. Create a program to handle the SIGPWR signal (power failure restart).
37. Create a C program to handle the SIGWINCH_SIGINFO signal (window size change or status request from keyboard).
38. Implement a program to handle the SIGSYS_SIGPIPE signal (bad system call or write on a pipe with no one to read it).
39. Write a program to handle the SIGURG_SIGTSTP signal (urgent condition on socket or stop signal).
40. Create a C program to handle the SIGCONT_SIGSTOP signal (continue or stop executing).
41. Implement a program to handle the SIGTTIN_SIGTTOU signal (background read or write attempted from control terminal).
42. Create a C program to handle the SIGXCPU_SIGXFSZ signal (CPU time limit exceeded or file size limit exceeded).
43. Implement a program to handle the SIGVTALRM_SIGWINCH signal (virtual timer expired or window size change).
44. Write a program on watch dog timer and it explain its working
45. Write a program to demonstrate how to handle and recover from a segmentation fault (SIGSEGV) in a system programming scenario.
46. Write a program to demonstrate the use of sigaction() for handling signals.
47. Write a program to demonstrate how to block signals using sigprocmask().
48. Write a program to implement a timer using signals.
49. Write a program to demonstrate signal handling in a multithreaded environment.
50. Write a program to handle a real-time signal using sigqueue().
51. Write a program to handle the SIGTSTP signal and suspend the process.
52. Write a program to demonstrate handling multiple signals using sigaction().
53. Write a program to demonstrate IPC using signals.

54. Write a program to demonstrate error handling using signals in system calls.
55. Write a program to demonstrate signal handling in a client-server architecture.
56. Write a program to demonstrate signal handling during fork() and exec().
57. Write a program to demonstrate how to block and unblock signals using sigprocmask() in a system programming context.
58. Write a program to demonstrate how to handle SIGBUS (bus error) signal in a system programming context.
59. Write a program to demonstrate the usage of sigaction() for handling SIGUSR1 and SIGUSR2 signals in a system programming scenario.
60. Create a C program to handle the SIGCHLD_SIGCONT signal (child process terminated or continue executing).
61. Write a program to handle the SIGBUS_SIGCHLD signal (bus error or child process terminated).
62. Implement a program to handle the SIGPWR_SIGSYS signal (power failure restart or bad system call).
63. Write a program to handle the SIGPIPE_SIGQUIT signal (write on a pipe with no one to read it or quit signal).
64. Write a C program that forks a child process. Implement signal handlers in both the parent and child processes to handle the SIGUSR1 signal. When the parent process receives SIGUSR1, it should send the signal to the child process, which then prints a message indicating the signal reception

IPC MECHANISMS

1. What is meant by an IPC Mechanism?
2. Why we use IPC Mechanism?
3. What are the types of IPC Mechanism's?
4. What is meant by “unicast” and “multicast” IPC?
5. What is meant by PIPES?
6. What is meant by Blocking Calls?
7. What are the types of Blocking Calls?
8. What are the different types of I/O Calls?
9. What are the I/O calls we are used in IPC Mechanisms?
10. What are the Blocking Calls used in IPC?
11. What is meant by Named Pipes?
12. Where is the FIFO Object created?
13. What is the call used to create a FIFO Object?
14. What are the Blocking Calls used in Named Pipes?
15. Why read system calls acts as a blocking call?
16. Difference between the Named Pipes and Pipes?
17. What is return value of read system call?
18. What is meant by message queue?
19. Why we use message queues?
20. What is difference between Named Pipe and Message Queue?
21. What is the system call used to create the message queue?
22. Where was the message queue created?
23. What is meant by Shared Memory?
24. Why we use Shared Memory?
25. Difference between Shared Memory and Message Queues?
26. What is use of stat command?
27. What is the use of semctl command?
28. How do we destroy the shared memory object?
29. What is meant by Semaphores?

30. Why we use semaphores?
31. What is meant by Synchronization?
32. What is meant by Asynchronization?
33. Why we use mutex locks?
34. What is the difference between mutex locks and semaphores?
35. What is meant by Race Condition?
36. What is meant by Deadlock?
37. What is meant by Critical Section?
38. What is the difference between system v and POSIX?
39. Describe the steps involved in creating and using a named pipe (FIFO) for IPC
40. Describe the steps involved in creating and using a pipe for IPC
41. Implement a program that uses pipes for communication between a parent and child process. Show how data can be passed between processes using pipes.
42. Create a program where two processes communicate synchronously using pipes. Ensure that one process waits for the other to finish before proceeding.
43. Implement a program that uses Named pipes for communication between two processes.
44. Write a C program to create a message queue using the msgget system call. Ensure that the program checks for errors during the creation process.
45. Develop two separate C programs, one for sending messages and the other for receiving messages through a created message queue.
46. Create a program to remove an existing message queue using the msgctl system call. Ensure that the program prompts the user for confirmation before deleting the message queue.
47. Design a multithreaded program where threads communicate through named pipes.
48. Write a C program where two processes communicate using message queues. Implement sending and receiving messages between the processes using msgget, msgsnd, and msgrcv.
49. Implement a program where two processes communicate synchronously using message queues. Ensure that one process waits for the other to finish before proceeding.
50. Design a program that uses a message queue for synchronization between multiple processes.
51. Write a C program that initializes a shared memory segment using shmget.

52. Develop a program that attaches to a previously created shared memory segment using shmat and detaches using shmdt.
53. Create a program that forks multiple processes, and each process communicates using shared memory.
54. Write a program that dynamically creates shared memory segments based on user input.
55. Create a program that monitors and displays the usage statistics of shared memory segments, such as the amount of memory used and the number of attached processes.
56. Design a program that dynamically resizes a shared memory segment based on changing requirements.
57. Write a C program that uses semget to create a new semaphore set
58. Enhance the program to initialize the values of the semaphores(binary semaphore) in the set using semctl.
59. Design a program where multiple processes are forked and synchronization is achieved using semaphores.
60. Write a program that combines semaphores and shared memory for synchronization between processes.
61. Create a multithreaded program where threads synchronize using semaphore sets.
62. Create a program with two processes that perform semaphore operations (wait and signal) on a semaphore set using semop.
63. Write a program that performs control operations on the semaphore set using semctl.
64. Develop a C program that implements a simple dining philosophers problem using semaphores for synchronization.
65. Write a program where multiple processes compete for access to a critical section using semaphores to ensure mutual exclusion.
66. Write a C program to create a pipe and pass an array of integers from the parent process to the child process through the pipe.
67. Implement a program where multiple child processes are created, and each child process communicates with the parent process using pipes.
68. Develop a program that uses pipes for bidirectional communication between two processes, where each process can send and receive messages.
69. Create a C program where multiple processes write data to a named pipe, and another process reads from the named pipe and displays the received data.

70. Implement a program where two processes exchange messages through a named pipe until a termination signal is received.
71. Develop a C program that acts as a server, continuously reading requests from a named pipe, and a client program that sends requests to the server through the same named pipe
72. Develop a program where multiple processes read from and write to a shared memory segment, synchronizing their access using semaphores.
73. Create a C program where the parent process creates a shared memory segment, writes data into it, and waits for the child process to read the data from the shared memory.
74. Write a program where multiple processes cooperate to calculate the sum of a large array of numbers stored in a shared memory segment.
75. Write a program where two processes communicate through a message queue. One process sends a message containing a string to the other process, which receives and prints the message.
76. Create a program where multiple processes coordinate access to a shared resource using semaphores(system v). Simulate a scenario where processes request and release access to the resource with proper synchronization
77. Develop a program where two processes share an integer value stored in a shared memory segment. One process increments the value and the other process decrements it, with proper synchronization.
78. 1. What is Interprocess communication?

    a) allows processes to communicate and synchronize their actions when using the same address space

    b) allows processes to communicate and synchronize their actions

    c) allows the processes to only synchronize their actions without communication

    d) none of the mentioned
79. Message passing system allows processes to _____

    a) communicate with each other without sharing the same address space

    b) communicate with one another by resorting to shared data

    c) share data

    d) name the recipient or sender of the message
80. Which of the following two operations are provided by the IPC facility?

a) write & delete message

b) delete & receive message

c) send & delete message

d) receive & send message

81. Messages sent by a process _____

a) have to be of a fixed size

b) have to be a variable size

c) can be fixed or variable sized

d) none of the mentioned

82. The link between two processes P and Q to send and receive messages is called _____

a) communication link

b) message-passing link

c) synchronization link

d) all of the mentioned

83. Which of the following are TRUE for direct communication?

a) A communication link can be associated with N number of process(N = max. number of processes supported by system)

b) A communication link is associated with exactly two processes

c) Exactly N/2 links exist between each pair of processes(N = max. number of processes supported by system)

d) Exactly two link exists between each pair of processes

84. In indirect communication between processes P and Q _____

a) there is another process R to handle and pass on the messages between P and Q

b) there is another machine between the two processes to help communication

c) there is a mailbox to help communication between P and Q

d) none of the mentioned

85. In the non blocking send _____

a) the sending process keeps sending until the message is received

b) the sending process sends the message and resumes operation

c) the sending process keeps sending until it receives a message

d) none of the mentioned

86. What is the role of a semaphore in IPC?

a) Coordinating access to shared resources
b) Transferring data between processes
c) Sending signals to other processes
d) Creating communication channels

87. Bounded capacity and Unbounded capacity queues are referred to as _____

a) Programmed buffering
b) Automatic buffering
c) User defined buffering
d) No buffering

88. What is the role of a semaphore in IPC?

a) Coordinating access to shared resources
b) Transferring data between processes
c) Sending signals to other processes
d) Creating communication channels

89. Which IPC mechanism provides a message-oriented communication model?

a) Shared memory
b) Pipes
c) Message queues
d) Sockets

90. In the context of IPC, what are signals used for?

a) Synchronizing processes
b) Coordinating access to shared memory
c) Sending messages between processes
d) Notifying processes about events or conditions

91. Which synchronization primitive is used to ensure mutually exclusive access to critical sections of code?

a) Semaphore
b) Mutex
c) Condition variable
d) Barrier

92. Which IPC mechanism provides a communication endpoint for processes to communicate over a network or between processes on the same system?

a) Named pipes
b) Message queues

c) Shared memory
d) Sockets

93. What is the primary advantage of using message queues for IPC compared to shared memory?
    a) Higher performance
    b) More flexible communication model
    c) Simplicity of implementation
    d) Direct access to shared data
94. Which IPC mechanism involves copying data between processes, potentially incurring additional overhead?
    a) Shared memory
    b) Message queues
    c) Sockets
    d) Semaphores
95. Which aspect is crucial in IPC to ensure that multiple processes or threads coordinate their activities and access shared resources safely?
    a) Data transfer efficiency
    b) Synchronization
    c) Message passing
    d) Resource allocation
96. Which IPC mechanism involves copying data between processes through a temporary storage area in the kernel?
    a. Shared memory
    b. Message queues
    c. Sockets
    d. Named pipes
97. Which IPC mechanism provides a high-performance, shared memory region for data exchange between processes?
    a) Pipes
    b) Semaphores
    c) Message queues
    d) Shared memory

98. Which IPC mechanism is being used in the following pseudocode snip?

```
// Producer process
while (true) {
    produce_item(item);
    send_item_to_queue(item);
}
// Consumer process
while (true) {
    receive_item_from_queue(item);
    consume_item(item);
}
```

a) Shared memory

b) Message queues

c) Sockets

d) Named pipes

99. What synchronization primitive is being used in the following pseudocode snippet?

```
// Mutex initialization
mutex = create_mutex();
// Thread 1
lock_mutex(mutex);
// Critical section
// Access shared resource
unlock_mutex(mutex);
// Thread 2
lock_mutex(mutex);
```

```
// Critical section
// Access shared resource
unlock_mutex(mutex);
```

a) Semaphore

b) Mutex

c) Condition variable

d) Barrier

100. What type of communication is being used in the following pseudocode snippet?

```
// Server process
create_socket();
bind_socket();
listen_for_connections();
while (true)
 {
   accept_connection();
   receive_data_from_client();
   process_data();
   send_response_to_client();
}
// Client process
create_socket();
connect_to_server();
send_data_to_server();
receive_response_from_server();
```

a) Shared memory

b) Message passing

c) Message queues

d) Sockets

DEBUGGING

1. What is meant by debugging?
2. What are the types of debugging tools?
3. Why do we use debugging?
4. What is the difference between hardware debugging and software debugging?
5. What command is used to add debugging information?
6. What are the compilation stages?
7. How do you stop the process after the 1st stage of compilation, and what happens in this stage?
8. How do you stop the process after the 2nd stage of compilation, and what happens in this stage?
9. How do you stop the process after the 3rd stage of compilation, and what happens in this stage?
10. How do you generate assembly code from the executable file?
11. What is the use of Strace?
12. What is the use of Ltrace?
13. How do you see the source code in GDB?
14. How do you see the assembly code in GDB?
15. How do you check if a particular variable is changing or not every time in GDB?
16. What instruction is given to set breakpoints, and why do we need them?
17. How do you check how many breakpoints a program contains?
18. What is meant by a backtrace?
19. What instruction is used to move into a function call in GDB?
20. What is meant by memory leakage?
21. What tools do we have to check for memory leakage?

VIVEN
EMBEDDED ACADEMY