**Question 1 (30 points)**

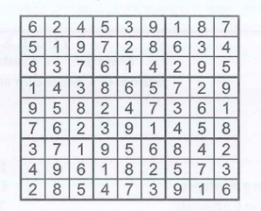
A particular river crossing is shared by both Linux hackers and Microsoft employees. A boat is used to cross the river, but it only seats four people, and must always carry a full load. In order to guarantee the safety of the hackers, you cannot put three employees and one hacker in the same boat; similarly, you cannot put three hackers in the same boat as an employee. Two procedures are needed, hackerArrives and employeeArrives, called by a hacker or employee when he/she arrives at the river bank. The procedures arrange the arriving hackers and employees into safe boatloads. Hackers or employees arrive at the river bank, they must wait until a boat is full.

1. Implement two methods hackerArrives and employeeArrives that arrange the arriving hackers and employees into safe boatloads. (20 points)

2. Write a main method to test these methods. (10 points)

**Question 2 (40 marks)**

A Sudoku puzzle uses a 9 × 9 grid in which each column and row, as well as each of the nine 3 × 3 subgrids, must contain all of the digits 1 through 9. The following figure presents an example of a valid Sudoku puzzle.



Design a multithreaded application that determines whether the solution to a Sudoku puzzle is valid. The suggested strategy is to create threads that check the following criteria:

a. A thread to check that each column contains the digits 1 through 9 (10 marks)

b A thread to check that each row contains the digits 1 through 9 (10 marks)

c. Nine threads to check that each of the 3 × 3 subgrids contains the digits 1 through 9 (10 marks)

d. Write a main method to test your multithread application with the Sudoku puzzle in the figure above (10 marlzs)

**Question 3 (20 marks)**

There is an agent who has infinite supply of all three ingredients A, B and C. There are three consumers C1, C2 and C3. Consumer C1 needs ingredient A, consumer C2 needs ingredient B, and consumer C3 needs ingredient C. The agent and consumers share a table. The agent randomly generates an ingredient and notifies the consumer who needs the ingredient. Once the ingredient is taken from the table, the agent supplies another one. On the other hand, each consumer waits for the agent's notification. Once it is notified, the consumer picks up the ingredient, consume the ingredient for a while, and goes back to the table waiting for his next ingredient.

Write a program that simulates this system, with three consumers and the agent being simulated by threads.

a. Methods called by consumers (5 marks)

b. Methods called by the agent (10 marks)

c. Write a main method to test your program. (5 marks)

**Question 4 (30 points):**

Consider the following set of processes with the length of the CPU-burst time given in milliseconds. Each process is assigned a numerical priority, with a lower number indicating a higher relative priority.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Arrive time** | **Burst time** | **Priority** |
| P1 | 0 | 20 | 4 |
| P2 | 25 | 30 | 3 |
| P3 | 30 | 25 | 2 |
| P4 | 60 | 15 | 2 |
| P5 | 80 | 10 | 3 |

**(1)Draw the Gantt charts** illustrating the execution of these processes using **preemptive SJF** and **preemptive priority**, and **RR (Round-robin)** scheduling (quantum = 10). (15 points)

Gantt Chart of SJF:

0 20 25 55 80 90 105

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 |  | P2 | P3 | P5 | P4 |

Gantt Chart of priority:

0 20 25 30 55 60 75 95 105

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P1 |  | P2 | P3 | P2 | P4 | P2 | P5 |

Gantt Chart of round robin:

0 10 20 25 35 45 55 65 75 85 90 95 105

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P1 |  | P2 | P3 | P2 | P3 | P4 | P2 | P3 | P4 | P5 |

(2) Which of the schedules results in the **minimal average waiting time** (over all processes)?(5 points)

=> the minimal average waiting time is priority

**(3)** What is the **response time** of each process for each of the scheduling algorithms? (5 points)

- priority:

Waiting Time Time:

P1: 20 - 20 = 0

P2: 70 - 30 = 40

P3: 25 - 25 = 0

P4: 15 - 15 = 0

P5: 25 - 10 = 15

Average Waiting Time: 55 / 7 = 7,86

--------------------------------------------------

-SJF:

Waiting Time Time:

P1: 20 - 20 = 0

P2: 30 - 30 = 0

P3: 50 - 25 = 25

P4: 45 - 15 = 30

P5: 10 - 10 = 0

Average Waiting Time: 55 / 5 = 11,00

--------------------------------------------------

-round robin:

Waiting Time Time:

P1: 20 - 20 = 0

P2: 60 - 30 = 30

P3: 60 - 25 = 35

P4: 35 - 15 = 20

P5: 25 - 10 = 15

Average Waiting Time: 100 / 5 = 20,00

**(4)** What is the **turnaround time** of each process for each of the scheduling algorithms?(5 points)

- priority:

Turnaround Time:

P1: 20 - 0 = 20

P2: 95 - 25 = 70

P3: 55 - 30 = 25

P4: 75 - 60 = 15

P5: 105 - 80 = 25

Average Turnaround Time: 155 / 7 = 22,14

--------------------------------------------------

-SJF:

Turnaround Time:

P1: 20 - 0 = 20

P2: 55 - 25 = 30

P3: 80 - 30 = 50

P4: 105 - 60 = 45

P5: 90 - 80 = 10

Average Turnaround Time: 155 / 5 = 31,00

--------------------------------------------------

-round robin:

Turnaround Time:

P1: 20 - 0 = 20

P2: 85 - 25 = 60

P3: 90 - 30 = 60

P4: 95 - 60 = 35

P5: 105 - 80 = 25

Average Turnaround Time: 200 / 5 = 40,00

**Question 5 (30 points):**

(1) Explain the difference between preemptive and nonpreemptive scheduling. (10 points)

- Preemptive Scheduling:

* The CPU can forcibly take control from a running process and allocate it to another process, based on priority or time slice.
* Ensures fair CPU usage and better responsiveness in multitasking environments.
* Example: Round-robin scheduling.

- Nonpreemptive Scheduling:

* Once a process starts running, it runs until it finishes or voluntarily gives up the CPU (e.g., by performing I/O).
* Simpler to implement but may lead to poor CPU utilization or starvation.
* Example: First-Come, First-Served (FCFS) scheduling.

(2) Give one programming example in which multithreading provides better performance than a single-threaded solution. (5 points)

- Example: Web servers

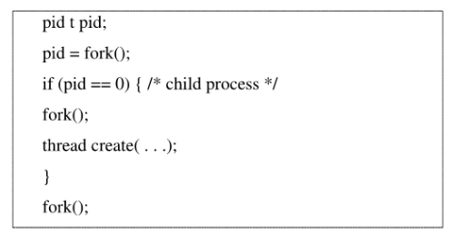
* A multithreaded web server can handle multiple client requests simultaneously.
* Each client request (e.g., serving a webpage) is assigned to a thread, allowing concurrent processing and faster response times.
* Single-threaded servers would handle one request at a time, leading to bottlenecks.

(3) Give one programming example in which multithreading does not provide better performance than a single-threaded solution. (5 points)

- Example: Programs with heavy dependency on sequential operations

* Example: Calculating Fibonacci numbers recursively.
* Recursive calculations depend on previous results, making multithreading inefficient due to overhead and lack of independent tasks.
* A single-threaded solution avoids synchronization and context-switching overhead.

(4) Consider the following code segment:



a. How many unique processes are created? (5 points)

=> 6 processes: Parent + 5 child processes.

Explain:

pid = fork() (1 parent + 1 child)

fork()-if() ( 1 child )

fork()-repeat ( 3 old + 3 new child) = 6

b. How many unique threads are created? (5 points)

=> 1 thread created in Child 1

**Question 6 (30 Points).**

(a) A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the following page reference string 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. **10 Points**

=== Least Recently Used (LRU) Page Replacement ===

4 7 6 1 7 6 1 2 7 2

4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| 7 | 7 | 7 | 7 | 7 | 7 | 2 | 2 | 2 |

| | 6 | 6 | 6 | 6 | 6 | 6 | 7 | 7 |

yes|yes|yes|yes|no |no |no |yes|yes|no |

=> number of page faults: 6

(b) Given 3 memory partitions 50 KB, 300 KB and 600 KB (In Order). Consider the 4 processes of size 300KB, 25KB, 125KB and 50KB (In order). Which algorithm among First Fit, Best Fit and Worst Fit makes the most efficient use of memory? **20 Points**

First Fit:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 50KB |  |  | 50KB | P2:25KB |  | 50KB | P2:25KB |  | 50KB | P4:50KB |
| 300KB | P1:300KB |  | 300KB | P1:300KB |  | 300KB | P1:300KB |  | 300KB | P1:300KB |
| 600KB |  |  | 600KB |  |  | 600KB | P3:125KB |  | 600KB | P3:125KB |

**Remaining Free Space**:

* 50 KB partition: 50−25=25 KB
* 300 KB partition: Fully allocated.
* 600 KB partition: 600−125=475 KB
* Total : 475 + 25 = 500KB

Best Fit:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 50KB |  |  | 50KB | P2:25KB |  | 50KB | P2:25KB |  | 50KB | P4:50KB |
| 300KB | P1:300KB |  | 300KB | P1:300KB |  | 300KB | P1:300KB |  | 300KB | P1:300KB |
| 600KB |  |  | 600KB |  |  | 600KB | P3:125KB |  | 600KB | P3:125KB |

**Remaining Free Space**:

* 50 KB partition: 50−25=25 KB
* 300 KB partition: Fully allocated.
* 600 KB partition: 600−125=475 KB
* Total : 475 + 25 = 500KB

Worst Fit:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 50KB |  |  | 50KB |  |  | 50KB |  |  | 50KB | P4:50KB |
| 300KB |  |  | 300KB | P2:25KB |  | 300KB | P3:125KB |  | 300KB | P3:125KB |
| 600KB | P1:300KB |  | 600KB | P1:300KB |  | 600KB | P1:300KB |  | 600KB | P1:300KB |

**Remaining Free Space**:

* 50 KB partition: Fully allocated.
* 300 KB partition: 300-25 + 300 – 175 = 275+125 = 400KB.
* 600 KB partition: 600−300=300 KB
* Total : 400 + 300 = 700KB

the most efficient use of memory: First Fit, Best Fit.

**Question 7 (25 Points).**

Suppose that a disk drive has 50 cylinders, numbered 0 to 49. Disk requests come into the disk driver for cylinders: 10, 22, 25, 2, 40, 6 and 38 in that order. The disk head is currently positioned over the cylinder 20. Suppose, the disk head moves from 0 to 49. A seek takes 6 millisecond per cylinder move. What is the total seek time using the algorithm SCAN?

==== Scan ====

Move Distance

20 -> 22 | 2

22 -> 25 | 3

25 -> 38 | 13

38 -> 40 | 2

40 -> 49 | 9

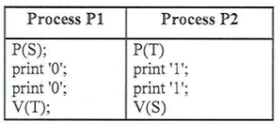
49 -> 10 | 39

10 -> 6 | 4

6 -> 2 | 4

Total seek time = (76) \* 6 = 456 ns

(b) Consider the following two processes P1 and P2. Let us assume S and T are binary semaphores. S is initially 1 and T is initially 0. In the code, P(S) and P(T) means wait operation; and V(S) and V(T) means signal operation. What will be the output string? **10 Points**



=> Output: 0011

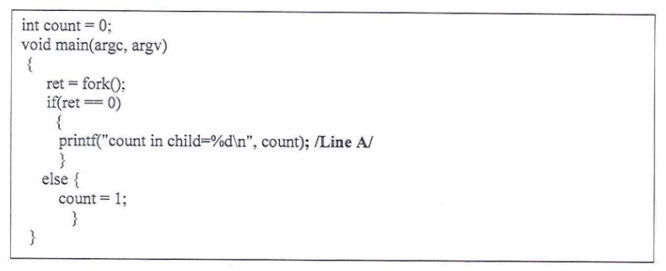
|  |
| --- |
| Explan:  - T=0 -> p(T) not run  - S=1 -> p(S) -> S-- -> S=0  Print(“00”)  - v(T) -> T++ -> T=1  - T=1 -> p(T) -> T-- -> T=0  - v(S) -> S++ -> S=1  Print(“11”) |

(c) What is Indexed File Allocation Method? **5 Points**

- Indexed File Allocation Method is a technique where each file has an **index block** that contains pointers to the disk blocks storing the file's data.

**Question 8 (10 Points):**

**A.**



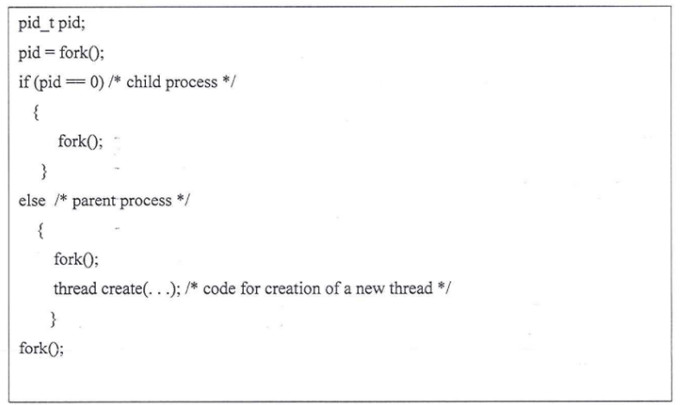
(i) Consider the following C program. Assume there are no syntax errors and the program executes correctly. Assume the fork system calls succeed. What is the output printed to the screen for Line A? **2.5 Points**

=> output: count in child=0

(ii) Why Threads are needed? **2.5 Points**

Because some reasons: Concurrency, Efficient Resource Utilization, Improved Performance, Parallel Computing, Responsiveness, Simplified Communication.

**B. Consider the following code segment:**



(i). How many unique processes are created? **2.5 Points**

Total unique processes = 8

(ii). How many unique threads are created? **2.5 Points**

Total unique threads = 1