UNIX & C Programming Tasks

# (a) Print Last 10 Lines of a File

## Aim:

To print the last 10 lines of a user-specified file using a UNIX command.

## Algorithm:

1. Accept a filename from the user.  
2. Use the `tail` command to read the last 10 lines.

## Procedure:

1. Open terminal.  
2. Prompt user to enter filename.  
3. Use `tail -n 10 filename` to display last 10 lines.

## Program:

echo "Enter filename:"  
read filename  
tail -n 10 "$filename"

## Output:

Enter filename:  
sample.txt  
(Line 91)  
(Line 92)  
...  
(Line 100)

## Result:

The shell script successfully prints the last 10 lines of the specified file.

# (b) Shell Script to Find Sum and Average

## Aim:

To write a shell script that calculates the sum and average of four integers.

## Algorithm:

1. Read four integers from the user.  
2. Calculate sum: sum = a + b + c + d  
3. Calculate average: average = sum / 4

## Procedure:

1. Start shell.  
2. Prompt user to input four numbers.  
3. Use arithmetic expansion to calculate and display sum and average.

## Program:

echo "Enter four integers:"  
read a  
read b  
read c  
read d  
  
sum=$((a + b + c + d))  
avg=$((sum / 4))  
  
echo "Sum: $sum"  
echo "Average: $avg"

## Output:

Enter four integers:  
10  
20  
30  
40  
Sum: 100  
Average: 25

## Result:

The script correctly computes the sum and average of four given integers.

# (c) Deadlock Detection in C

## Aim:

To implement a simple deadlock detection algorithm using C.

## Algorithm:

1. Define the allocation, request, and available matrices.  
2. Check if each process’s request can be satisfied.  
3. If yes, release its allocated resources.  
4. Repeat until all processes finish or deadlock is found.

## Procedure:

1. Define matrices (allocation, request, available).  
2. Loop to check if requests can be fulfilled.  
3. Mark finished processes and update available resources.  
4. Print result based on whether all processes could finish.

## Program:

#include <stdio.h>  
int main() {  
 int n = 2, m = 2, i;  
 int alloc[2][2] = {{1,0},{0,1}};  
 int req[2][2] = {{0,1},{1,0}};  
 int avail[2] = {0, 0}, finish[2] = {0, 0};  
  
 for (int k = 0; k < 2; k++)  
 for (i = 0; i < 2; i++)  
 if (!finish[i] && req[i][0] <= avail[0] && req[i][1] <= avail[1]) {  
 avail[0] += alloc[i][0];  
 avail[1] += alloc[i][1];  
 finish[i] = 1;  
 }  
  
 if (finish[0] && finish[1]) printf("No deadlock.\n");  
 else printf("Deadlock detected.\n");  
 return 0;  
}

## Output:

Deadlock detected.

## Result:

The C program checks for deadlock using a basic resource-allocation model and successfully detects whether deadlock exists or not.