Description of a Problem

**Problem Description: Print Spooler Simulation in C**

In modern computer systems, a **print spooler** is a program that manages print jobs sent from applications to a printer. Since printers can typically only handle one job at a time, the spooler is responsible for queueing print jobs and ensuring that they are printed one after the other. The spooler holds jobs temporarily in a queue and then sends them to the printer in the order they were received.

The objective of this problem is to simulate a **print spooler** system using a **queue** data structure, which follows the **First-In-First-Out (FIFO)** principle. In this simulation, we will implement basic functionality for a print spooler, allowing the user to add new print jobs to the queue, process (print) the jobs in the order they were received, and view the list of jobs currently waiting to be printed.

**Features:**

1. **Adding Jobs to the Spooler:** Users can submit multiple print jobs, each with a unique job ID and job name. These jobs will be added to the queue in the order they are submitted.
2. **Processing Print Jobs:** The spooler will process jobs in the order they were received. Once a job is processed (i.e., printed), it is removed from the queue.
3. **Displaying Pending Jobs:** At any point, the user can view the current list of jobs waiting in the queue to be printed.
4. **Exiting the Spooler:** The user can exit the spooler simulation when desired.

**Key Concepts:**

* **Queue Data Structure:** The print spooler must manage jobs using a queue, ensuring that the first job submitted is also the first one processed (FIFO order).
* **Dynamic Allocation:** Since the number of print jobs is not predetermined, a dynamic data structure like a linked list is used to store the print jobs in the queue.

**Example Scenario:**

* A user submits three print jobs: Job1, Job2, and Job3.
* The print spooler adds these jobs to the queue in the order they were submitted.
* When the user requests the spooler to process a job, Job1 (the first job submitted) is processed and removed from the queue.
* The user can view the remaining jobs (Job2 and Job3) still waiting to be processed.

**Problem Breakdown:**

1. **Input:**
   * The user provides a job name and job ID when submitting a job to the spooler.
   * The user can choose an action: add a job, process a job, display jobs, or exit.
2. **Output:**
   * Confirmation messages when jobs are added.
   * Details of the job being processed when a print job is dequeued.
   * The list of jobs currently waiting in the queue when requested.
3. **Constraints:**
   * Jobs must be processed in the order they are submitted (FIFO).
   * The spooler should handle an arbitrary number of jo

By implementing this system, users can experience how a basic print spooler manages tasks and the significance of the queue data structure in task scheduling systems.

Solution of problem in terms of Flowchart

**Flowchart Explanation:**

1. **Start: Begin the program and initialize the queue.**
2. **Display Menu: Present the user with options to add a job, process a job, display jobs, or exit.**
3. **User Input: Based on the user's choice, proceed to the next steps:**
4. **Add Job: Get job details (Job ID and Job Name) from the user, then enqueue the job in the queue.**
5. **Process Job: If there is a job in the queue, dequeue the front job and process it (print and remove). If no jobs are available, show a message that the queue is empty.**
6. **Display Jobs: Traverse the queue and display all pending jobs. If the queue is empty, show a message.**
7. **Exit: Terminate the program.**
8. **Return to Menu: After performing the selected action, go back to display the menu for further choices until the user exits.**

+------------------+

| Start |

+------------------+

|

v

+-----------------------------+

| Initialize the print queue |

+-----------------------------+

|

v

+-----------------------------+

| Display Menu |

| 1. Add Print Job |

| 2. Process Next Job |

| 3. Display All Jobs |

| 4. Exit |

+-----------------------------+

|

v

+-----------------------------+

| Get User's Choice |

+-----------------------------+

|

+------------+------------+

|

v v

+------------------+ +------------------+

| Choice: Add Job | | Choice: Process |

+------------------+ | Job |

| Enter Job ID & | +------------------+

| Job Name | | Is Queue Empty? |

+------------------+ +------------------+

| Enqueue Job | | Yes | No |

+------------------+ +---------+--------+

| |

v v

+----------------------------+ +-----------------------+

| Return to Display Menu | | Dequeue & Process Job |

+----------------------------+ +-----------------------+

|

v

+----------------------------+

| Return to Display Menu |

+----------------------------+

|

+-----------+----------+

| |

v v

+-------------------+ +-------------------+

| Choice: Display | | Choice: Exit |

| Jobs | +-------------------+

+----------------------+ |

| Is Queue Empty? | v

+-------------------+ +-------------------+

| Yes | No | | Exit |

+---------+----------+ +-------------------+

| |

v v

+-------------------+ +--------------------------+

| Display: "No jobs | | Traverse and display |

| in queue" | | all jobs in queue |

+-------------------+ +--------------------------+

| |

+-----------------------+

|

v

+----------------------------+

| Return to Display Menu |

+----------------------------+

solution of a problem in terms of coding

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Structure for a print job**

**typedef struct PrintJob {**

**int jobId;**

**char jobName[50];**

**struct PrintJob\* next;**

**} PrintJob;**

**// Queue structure**

**typedef struct {**

**PrintJob\* front;**

**PrintJob\* rear;**

**} Queue;**

**// Function to create a new print job**

**PrintJob\* createPrintJob(int jobId, char\* jobName) {**

**PrintJob\* newJob = (PrintJob\*)malloc(sizeof(PrintJob));**

**newJob->jobId = jobId;**

**strcpy(newJob->jobName, jobName);**

**newJob->next = NULL;**

**return newJob;**

**}**

**// Initialize the queue**

**void initQueue(Queue\* q) {**

**q->front = q->rear = NULL;**

**}**

**// Function to check if the queue is empty**

**int isEmpty(Queue\* q) {**

**return q->front == NULL;**

**}**

**// Function to add a job to the queue (enqueue)**

**void enqueue(Queue\* q, int jobId, char\* jobName) {**

**PrintJob\* newJob = createPrintJob(jobId, jobName);**

**if (q->rear == NULL) {**

**q->front = q->rear = newJob;**

**printf("Added print job: %s (ID: %d)\n", jobName, jobId);**

**return;**

**}**

**q->rear->next = newJob;**

**q->rear = newJob;**

**printf("Added print job: %s (ID: %d)\n", jobName, jobId);**

**}**

**// Function to remove a job from the queue (dequeue)**

**void dequeue(Queue\* q) {**

**if (isEmpty(q)) {**

**printf("No print jobs in the queue.\n");**

**return;**

**}**

**PrintJob\* temp = q->front;**

**printf("Processing print job: %s (ID: %d)\n", temp->jobName, temp->jobId);**

**q->front = q->front->next;**

**if (q->front == NULL) {**

**q->rear = NULL;**

**}**

**free(temp);**

**}**

**// Function to display the current jobs in the queue**

**void displayQueue(Queue\* q) {**

**if (isEmpty(q)) {**

**printf("No print jobs in the queue.\n");**

**return;**

**}**

**PrintJob\* temp = q->front;**

**printf("Current print jobs in the queue:\n");**

**while (temp != NULL) {**

**printf("Job Name: %s, Job ID: %d\n", temp->jobName, temp->jobId);**

**temp = temp->next;**

**} }**

**int main() {**

**Queue q;**

**initQueue(&q);**

**int choice, jobId;**

**char jobName[50];**

**while (1) {**

**printf("\nPrint Spooler Menu:\n");**

**printf("1. Add print job\n");**

**printf("2. Process next job\n");**

**printf("3. Display all jobs\n");**

**printf("4. Exit\n");**

**printf("Enter your choice: ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1:**

**printf("Enter Job ID: ");**

**scanf("%d", &jobId);**

**printf("Enter Job Name: ");**

**scanf("%s", jobName);**

**enqueue(&q, jobId, jobName);**

**break;**

**case 2:**

**dequeue(&q);**

**break;**

**case 3:**

**displayQueue(&q);**

**break;**

**case 4:**

**printf("Exiting program...\n");**

**exit(0);**

**break;**

**default:**

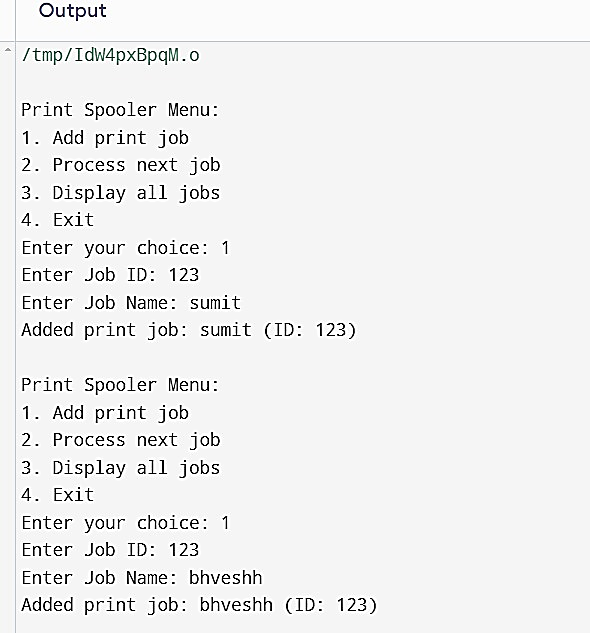
**printf("Invalid choice, please try again.\n");**

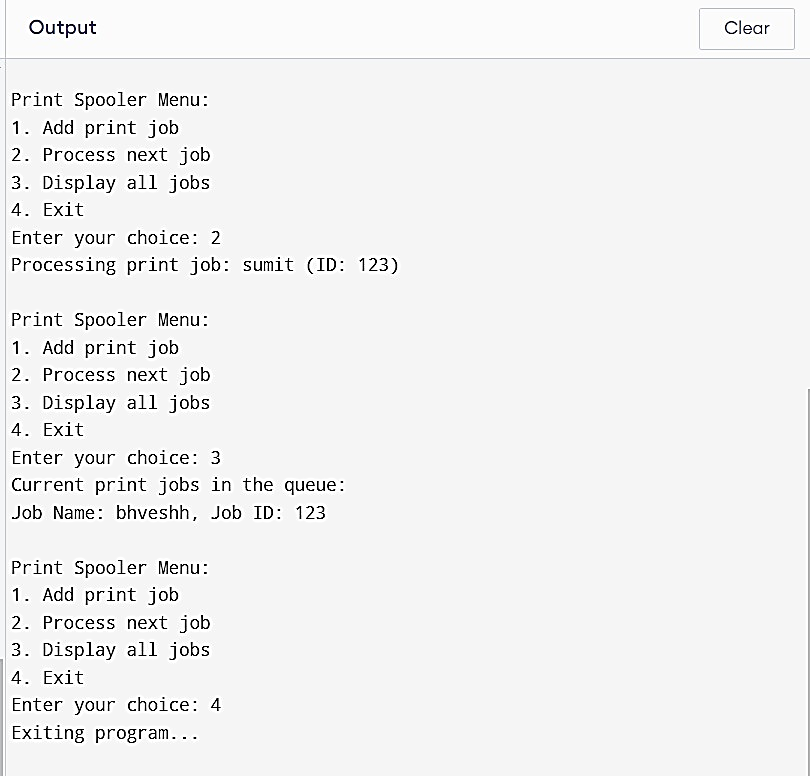
**}**

**}**

**return 0; }**

output screenshots

****

****

REFERENCE:-

Chirag Nareshkumar Soni

g-mail:-chiraganon@gmail.com