ASSIGNMENT-1 MINI PROJECT

Title of The Project: Job Scheduling System using Heaps for Efficient Task Prioritization

Group No: 13

Name: Pranav (4MW24CS105)
Rijesh S (4MW24CS116)
Rohan Mendon (4MW24CS117)
Sagar H Poojary(4MW24CS119)

Abstract:

The Job Scheduling System using Priority Queues is designed to simulate a real-world task scheduling mechanism where jobs are executed based on their priority. In modern computing environments such as operating systems, servers, and task managers, efficient scheduling is crucial to ensure optimal resource utilization.

Application Description:

The Job Scheduling System provides the following functionalities through a menu-driven interface:

1. **Add Job**
   Allows the user to insert a new job with a Job ID, Job Name, and Priority into the priority queue.
2. **Execute Highest Priority Job**
   Removes and executes the job with the highest priority from the queue.
3. **Display Scheduled Jobs**
   Displays all jobs currently present in the priority queue along with their priorities.
4. **Exit**
   Terminates the program safely.

Algorithm:

1. Menu Display and Input

- Display a menu with options: Add Job, Execute Job, Display Jobs, and Exit.
- Prompt the user to enter their choice.

2. Action Based on User Choice

- Use a switch statement to execute the corresponding operation:
  - **Add Job**: Insert a new job into the priority queue and maintain heap property**.**
  - **Execute Job:** Remove the job with the highest priority and re-heapify.
  - **Display Jobs:** Traverse and display all jobs in the queue.
  - **Exit:** Terminate the program.

3. Heap Operations

- **Heapify Up**: Maintains heap property after insertion.
- **Heapify Down**: Maintains heap property after deletion.

4. Error Handling

- Display overflow message when queue is full.
- Display underflow message when queue is empty.
- Handle invalid menu inputs gracefully.

Methodology:

- **Data Structure Used**: Priority Queue implemented using Binary Max Heap.
- **Programming Language**: C
- **Menu-Driven Design**: Enables continuous interaction until the user exits.
- **Memory Management**: Array-based heap ensures efficient memory usage.
- **Error Handling**: Prevents invalid operations such as deletion from an empty queue.

Additional Notes(Optional):

- Priority Queue is implemented using a binary heap.
- Higher priority value indicates higher execution preference.
- The program uses an array-based heap with a fixed size.
- The interface is simple and command-line based.

Source Code:

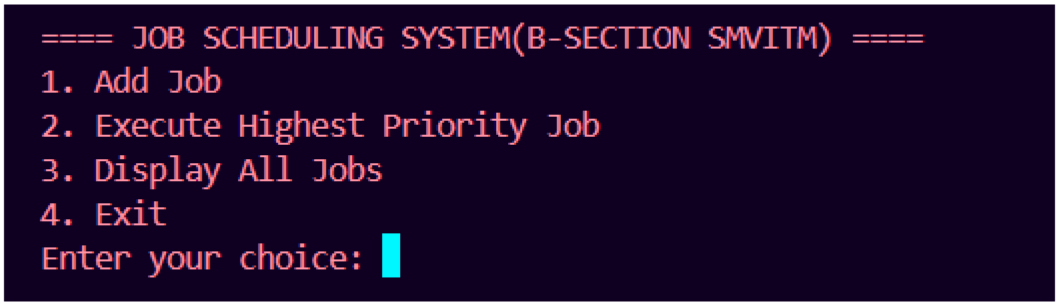
```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define MAX 50


struct Job {

    int jobId;

    char jobName[30];

    int priority;

};


struct Job heap[MAX];

int size = 0;


/* Function to swap two jobs */

void swap(struct Job *a, struct Job *b) {

    struct Job temp = *a;

    *a = *b;

    *b = temp;

}


/* Heapify Up */

void heapifyUp(int index) {
    while (index > 0 && heap[(index - 1) / 2].priority < heap[index].priority) {
        swap(&heap[(index - 1) / 2], &heap[index]);
        index = (index - 1) / 2;
    }
}

/* Heapify Down */
void heapifyDown(int index) {
    int largest = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;

    if (left < size && heap[left].priority > heap[largest].priority)
        largest = left;

    if (right < size && heap[right].priority > heap[largest].priority)
        largest = right;

    if (largest != index) {
        swap(&heap[index], &heap[largest]);
        heapifyDown(largest);
    }
}

/* Insert a Job */
```

```
void insertJob() {
    if (size == MAX) {
        printf("\nQueue Overflow! Cannot insert more jobs.\n");
        return;
    }

    struct Job newJob;
    printf("\nEnter Job ID: ");
    scanf("%d", &newJob.jobId);

    printf("Enter Job Name: ");
    scanf("%s", newJob.jobName);

    printf("Enter Job Priority (Higher value = Higher priority): ");
    scanf("%d", &newJob.priority);

    heap[size] = newJob;
    heapifyUp(size);
    size++;

    printf("\nJob inserted successfully.\n");
}

/* Delete Highest Priority Job */
void deleteJob() {
    if (size == 0) {
        printf("\nQueue Underflow! No jobs to execute.\n");
        return;
    }

    printf("\nExecuting Job:");
    printf("\nJob ID: %d", heap[0].jobId);
    printf("\nJob Name: %s", heap[0].jobName);
    printf("\nPriority: %d\n", heap[0].priority);

    heap[0] = heap[size - 1];
    size--;
    heapifyDown(0);
}

/* Display Jobs */
void displayJobs() {
    if (size == 0) {
        printf("\nNo jobs in the queue.\n");
        return;
    }

    printf("\nScheduled Jobs (Priority Order):\n");
    printf("ID\tName\t\tPriority\n");
    printf("----------------------------------\n");
    for (int i = 0; i < size; i++) {
        printf("%d\t%s\t\t%d\n", heap[i].jobId, heap[i].jobName, heap[i].priority);
    }
}

/* Main Menu */
int main() {
    int choice;

    do {
        printf("\n\n==== JOB SCHEDULING SYSTEM ====");
        printf("\n1. Add Job");
        printf("\n2. Execute Highest Priority Job");
        printf("\n3. Display All Jobs");
        printf("\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            insertJob();
            break;
        case 2:
            deleteJob();
            break;
        case 3:
            displayJobs();
            break;
        case 4:
            printf("\nExiting Program...\n");
            break;
        default:
            printf("\nInvalid choice! Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Results and Discussion:

1. **Main Menu:**

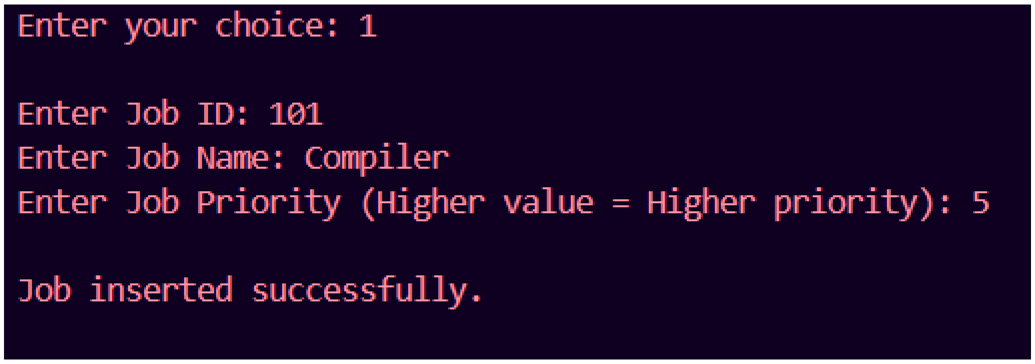
- The program displays a menu with four options.
- User input determines the operation to be performed.



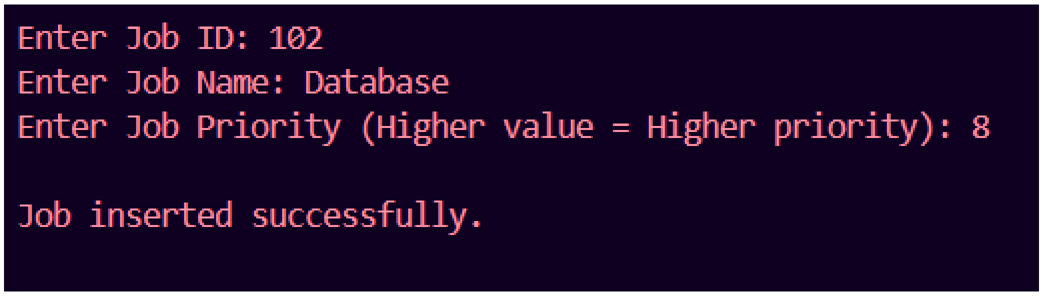
```
==== JOB SCHEDULING SYSTEM(B-SECTION SMVITM) ====
1. Add Job
2. Execute Highest Priority Job
3. Display All Jobs
4. Exit
Enter your choice: 
```

2. Add Job Operation

- Accepts Job ID, Job Name, and Priority.
- Inserts the job into the priority queue.
- Maintains heap property using heapify-up operation.



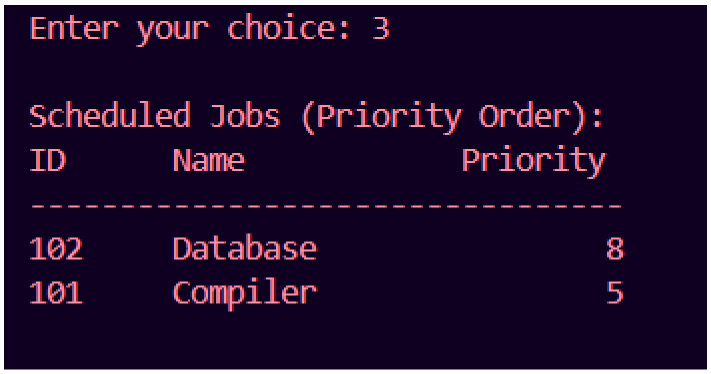
```
Enter your choice: 1

Enter Job ID: 101
Enter Job Name: Compiler
Enter Job Priority (Higher value = Higher priority): 5

Job inserted successfully.
```



```
Enter Job ID: 102
Enter Job Name: Database
Enter Job Priority (Higher value = Higher priority): 8

Job inserted successfully.
```

3.Display Jobs:

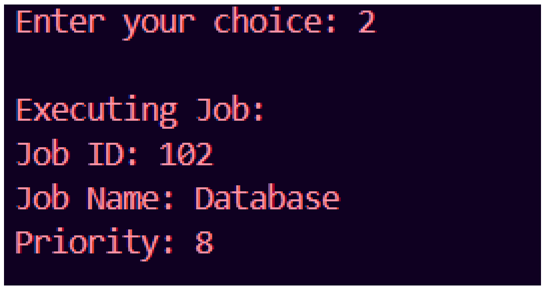
- Displays all jobs currently scheduled.
- Shows job ID, job name, and priority.



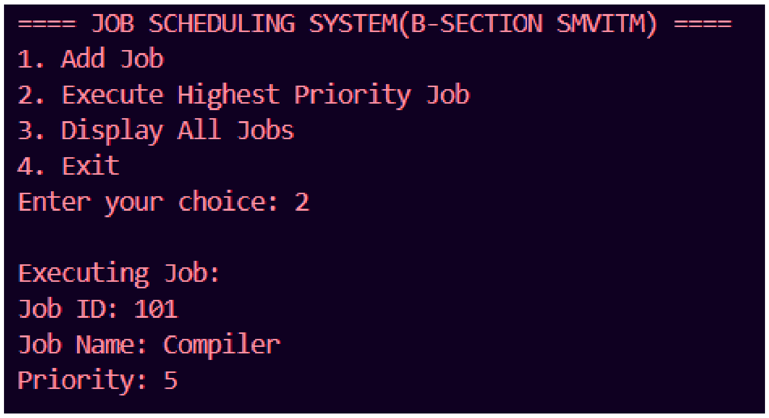
```
Enter your choice: 3

Scheduled Jobs (Priority Order):
ID      Name                Priority
---------------------------------
102     Database                   8
101     Compiler                   5
```

4.Execute Highest Priority Job

- Removes the job with the highest priority.
- Displays job details.
- Reorganizes the heap using heapify-down.



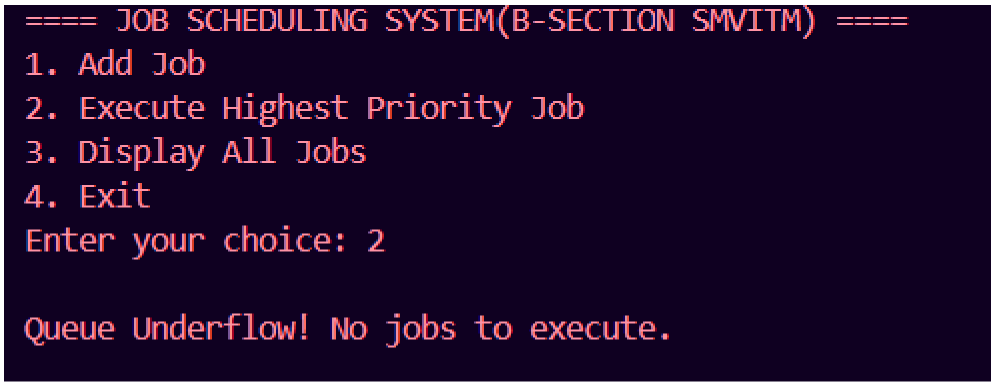
```
Enter your choice: 2

Executing Job:
Job ID: 102
Job Name: Database
Priority: 8
```



```
==== JOB SCHEDULING SYSTEM(B-SECTION SMVITM) ====
1. Add Job
2. Execute Highest Priority Job
3. Display All Jobs
4. Exit
Enter your choice: 2

Executing Job:
Job ID: 101
Job Name: Compiler
Priority: 5
```

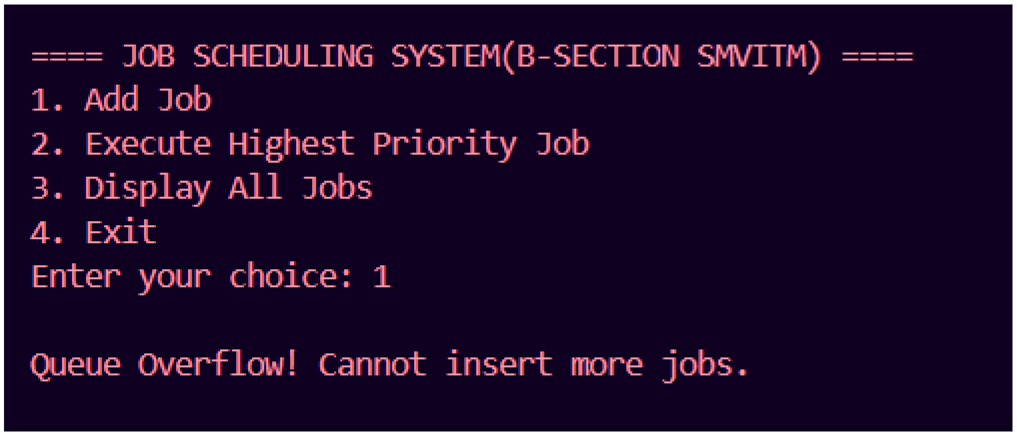
5. Error Handling

- Displays overflow message if queue is full.
- Displays underflow message if no jobs are available for execution.
- Handles invalid menu choices.



```
==== JOB SCHEDULING SYSTEM(B-SECTION SMVITM) ====
1. Add Job
2. Execute Highest Priority Job
3. Display All Jobs
4. Exit
Enter your choice: 2

Queue Underflow! No jobs to execute.
```

*After inserting maximum number of jobs



```
==== JOB SCHEDULING SYSTEM(B-SECTION SMVITM) ====
1. Add Job
2. Execute Highest Priority Job
3. Display All Jobs
4. Exit
Enter your choice: 1

Queue Overflow! Cannot insert more jobs.
```