MIT World Peace University

Fundamental Data Structures
Second Year B. Tech, Semester 1

Job Scheduling using Circular Queue

Practical Report
Assignment 9

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 28, 2022

Contents

1 Aim

Writing a C Program to simualate job scheduling using a linear queue.

2 Objectives

1. To study Queue and its operations
2. To study the importance of queue as a data structure in computer science

3 Problem Statements

Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write a program for simulating job queue. Write functions to add job and delete job from queue.

4 Theory

4.1 Queue

A queue is a linear data structure that follows the FIFO (First In First Out) principle. It is a simple data structure that allows adding and removing elements in a particular order. A real-life example of a queue is a line of people at a ticket counter. The first person in the line is the first one to get the ticket and the last person in the line is the last one to get the ticket. The elements are added at the end of the queue and are removed from the front of the queue. The operations that can be performed on a queue are:

1. Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
2. Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
3. Front: Get the front item from queue.
4. Rear: Get the last item from queue.

4.2 Types of Queues

1. **Linear Queue**: The elements are stored in a linear fashion. The elements are added from one end and removed from the other end. The insertion and deletion operations are performed at the two ends of the queue. The insertion operation is called enqueue and the deletion operation is called dequeue.
2. **Circular Queue**: The elements are stored in a circular fashion. The elements are added from one end and removed from the other end. The insertion and deletion operations are performed at the two ends of the queue. The insertion operation is called enqueue and the deletion operation is called dequeue.

3. **Priority Queue**: This queue is a special type of queue. Its specialty is that it arranges the elements in a queue based on some priority. The priority can be something where the element with the highest value has the priority so it creates a queue with decreasing order of values. The priority can also be such that the element with the lowest value gets the highest priority so in turn it creates a queue with increasing order of values.

4. **Dequeue:** Dequeue is also known as Double Ended Queue. As the name suggests double ended, it means that an element can be inserted or removed from both the ends of the queue unlike the other queues in which it can be done only from one end. Because of this property it may not obey the First In First Out property.

4.3 Applications of Queue

1. **CPU Scheduling**: The operating system uses a queue to keep track of the processes that are in the ready state. The process that is at the front of the queue is the one that is currently being executed by the CPU. The process that is at the rear of the queue is the one that is waiting to be executed.

2. **Disk Scheduling**: The operating system uses a queue to keep track of the disk requests that are waiting to be serviced. The request that is at the front of the queue is the one that is currently being serviced by the disk. The request that is at the rear of the queue is the one that is waiting to be serviced.

3. **Call Center Phone Systems**: The phone system uses a queue to keep track of the incoming phone calls. The call that is at the front of the queue is the one that is currently being serviced by the phone system. The call that is at the rear of the queue is the one that is waiting to be serviced.

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : gcc on linux for C

6 Input

The element or Job number to pass to the queue

7 Output

The Queue

8 Test Conditions

DeleteQ(), AddQ(), AddQ(),delete(),delete(),delete()

9 Code

9.1 Pseudo Code for Add to linear Queue

```
AddQ()
{
  if (rear == MAX-1)
  {
    printf("Queue Overflow");
  }
  else
  {
    if (front == -1)
    {
      front = 0;
    }
    printf("Enter the value to be added in the queue : ");
    scanf("%d", &add_item);
    rear = rear + 1;
    queue_array[rear] = add_item;
  }
}
```

9.2 Pseudo Code for Delete to linear Queue

```

DeleteQ()
{
  if (front == - 1 || front > rear)
  {
    printf("Queue Underflow \n");
    return ;
  }
  else
  {
    printf("Element deleted from queue is : %d\n", queue_array[front]);
    front = front + 1;
  }
}
```

9.3 Pseudo Code to check isEmpty for linear Queue

```

isEmpty()
{
  if (front == -1)
  {
    printf("Queue is empty");
  }
  else
  {
    printf("Queue is not empty");
  }
}
```

9.4 Pseudo Code for check isFull for linear Queue

```

isFull()
{
  if (rear == MAX-1)
  {
    printf("Queue is full");
  }
  else
  {
    printf("Queue is not full");
  }
}
```

9.5 Pseudo Code for Add to Circular Queue

```

AddQ()
{
  if ((rear + 1) % MAX == front)
  {
    printf("Queue Overflow \n");
  }
  else
  {
    if (front == -1)
    {
      front = 0;
    }
    printf("Inset the element in queue : ");
    scanf("%d", &add_item);
    rear = (rear + 1) % MAX;
    queue_array[rear] = add_item;
  }
}
```

9.6 Pseudo Code for Delete to Circular Queue

```

DeleteQ()
{
  if (front == -1)
  {
    printf("Queue Underflow \n");
  }
  else
  {
    printf("Element deleted from queue is : %d\n", queue_array[front]);
    if (front == rear)
    {
      front = -1;
      rear=-1;
    }
    else
    {
      front = (front + 1) % MAX;
    }
  }
}
```

9.7 Pseudo Code to check isEmpty for Circular Queue

```

isEmpty()
{
  if (front == -1)
  {
    printf("Queue is empty");
  }
  else
  {
    printf("Queue is not empty");
  }
}
```

9.8 Pseudo Code for check isFull for Circular Queue

```

isFull()
{
  if ((rear + 1) % MAX == front)
  {
    printf("Queue is full");
  }
  else
  {
    printf("Queue is not full");
  }
}
```

9.9 C Implementation of Problem Statement

```
// Queue thing
#include <stdio.h>
#define MAX_SIZE 5
int front = -1, rear = -1;

int queue[MAX_SIZE];

int isFull(void)
{
  if (rear == MAX_SIZE - 1)
    return (1);
  else
    return (0);
}

int isEmpty(void)
{
  if (rear == front)
  {
    return (1);
  }
  else
  {
    return (0);
  }
}

int enqueue(int item)
{
  if (!isFull())
  {
    rear++;
    queue[rear] = item;
  }
  else
  {
    printf("\nQUEUE OVERFLOW!\n");
  }
}

int dequeue(void)
{
  if (isEmpty())
  {
    printf("Queue is Empty \n\n QUEUE UNDERFLOW!!\n\n");
    return (-1);
  }
  else
  {
    // printf("Removed this thing %c\n", stack[top]);
    front++;
    return (queue[front]);
  }
}
void display_queue(void)
{
  int i;

  if (isEmpty())
  {
    printf("\n\nQueue is empty\n\n");
  }
  else
  {
    printf("Queue is: \n");
    for (i = front + 1; i <= rear; i++)
    {
      printf("%d \n", queue[i]);
    }
  }
}
int main(void)
{
  int choice;
  int temp;

  choice = 0;
  while (choice != 6)
  {
    printf("Enter what you want to do: \n\
    1. Add Job to the Ready Queue\n\
    2. Kill or Terminate Job from the Queue\n\
    3. See the Queue\n\
    4. Check if Ready Queue is Empty and Processor is free\n\
    5. Check if Ready Queue is full\n\
    6. Exit\n\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the Process Number of the Job you want to add. \n");
      scanf(" %d", &temp);
      enqueue(temp);
      display_queue();
      break;
    case 2:
      printf("Killing or Terminating Job from the Queue\n");
      dequeue();
      display_queue();
      break;
    case 3:
      display_queue();
      break;
    case 4:
      if (isEmpty())
      {
        printf("Yup, Queue is empty\n");
      }
      else
      {
        printf("Nope Queue isnt empty\n");
        display_queue();
      }
      break;
    case 5:
      if (isFull())
      {
        printf("\nYes the Queue is full!\n");
      }
      else
      {
        printf("No Queue isnt full!\n");
      }
      break;
    default:
      printf("\nThank You\n");
      break;
    }
  }
  return 0;
}
```

Listing 1: Main.Cpp

9.10 Input and Output

```
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

1
Enter the Process Number of the Job you want to add.
1
Queue is:
1
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

1
Enter the Process Number of the Job you want to add.
2
Queue is:
1
2
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

1
Enter the Process Number of the Job you want to add.
3
Queue is:
1
2
3
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

1
Enter the Process Number of the Job you want to add.
5
Queue is:
1
2
3
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

2
Killing or Terminating Job from the Queue
Queue is:
2
3
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

2
Killing or Terminating Job from the Queue
Queue is:
3
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

2
Killing or Terminating Job from the Queue
Queue is:
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

1
Enter the Process Number of the Job you want to add.
5
Queue is:
5
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit
```

```

1
Enter the Process Number of the Job you want to add.
65

QUEUE OVERFLOW!
Queue is:
5
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

5

Yes the Queue is full!
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

4
Nope Queue isnt empty
Queue is:
5
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

2
Killing or Terminating Job from the Queue
Queue is:
5
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

2
Killing or Terminating Job from the Queue


Queue is empty

```

```
Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

1
Enter the Process Number of the Job you want to add.
1

QUEUE OVERFLOW!


Queue is empty

Enter what you want to do:
    1. Add Job to the Ready Queue
    2. Kill or Terminate Job from the Queue
    3. See the Queue
    4. Check if Ready Queue is Empty and Processor is free
    5. Check if Ready Queue is full
    6. Exit

6

Thank You
```

Listing 2: Output

10 Time Complexity

- AddQ() : O(1)
- DeleteQ() : O(1)
- isEmpty() : O(1)
- isFull() : O(1)

11 Conclusion

Thus, implemented Job Scheduling using Linear Queue in C.

12 FAQs

1. **What are the advantages and disadvantages of a linear queue ?**
   Advantages

   (a) A large amount of data can be managed efficiently with ease.

   (b) Operations such as insertion and deletion can be performed with ease as it follows the first in first out rule.

   (c) Queues are useful when a particular service is used by multiple consumers.

(d) Queues are fast in speed for data inter-process communication.

(e) Queues can be used in the implementation of other data structures.

Disadvantages

(a) As the insertion in the queue is from the rear end and in the case of Linear Queue of fixed size insertion is not possible when rear reaches the end of the queue.

(b) But in the case of Circular Queue, the rear end moves from the last position to the front position circularly.

2. **What are the advantages and disadvantages of a circular queue ?**
Advantages

(a) Easier for insertion-deletion: In the circular queue, elements can be inserted easily if there are vacant locations until it is not fully occupied, whereas in the case of a linear queue insertion is not possible once the rear reaches the last index even if there are empty locations present in the queue.

(b) Efficient utilization of memory: In the circular queue, there is no wastage of memory as it uses the unoccupied space, and memory is used properly in a valuable and effective manner as compared to a linear queue.

(c) Ease of performing operations: In the linear queue, FIFO is followed, so the element inserted first is the element to be deleted first. This is not the scenario in the case of the circular queue as the rear and front are not fixed so the order of insertion-deletion can be changed, which is very useful.

Disadvantages

(a) Circular queue is not suitable for implementing the BFS algorithm as it is not FIFO.

(b) Circular queue is not suitable for implementing the DFS algorithm as it is not LIFO.

(c) Searching an element takes O(N) time.

(d) Maximum size of a queue must be defined prior.

3. **Give various applications to the queue .**

(a) Queue used to model real-world situations such as people waiting in line at a bank, airplanes waiting to take off, or data packets waiting to be transmitted over the Internet.

(b) There are various queues quietly doing their job in your computer's (or the network's) operating system.

(c) There's a printer queue where print jobs wait for the printer to be available.

(d) Stores, reservation centers, and other similar services typically process customer requests according to the FIFO principle. A queue would therefore be a logical choice for a data structure to handle transaction processing for such applications. For example, it would be a natural choice for handling calls to the reservation center of an airline.

(e) CPU Scheduling

(f) Disk Scheduling

(g) Handling of interrupts in real-time systems

(h) ATM Booth Line

(i) icket Counter Line

(j) Key press sequence on the keyboard

(k) CPU task scheduling

(l) Waiting time of each customer at call centers.