Assignment 11

21118

Problem statement:
Queues are frequently used in computer programming, & a typical example is the creation of job queue by an operating system. If the operating system doesn't use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job & delete job from queue.

Objectives
1) Understand the queue data structure & implement various operations of queue.
2) Implement job queue using queue data structure.

Outcomes
1) Implement queue data structure using arrays.
2) Menu based program for job simulation in operating system.

Hardware requirement:
Manufacturer & model: Acer Swift-3
Processor: Intel core i5 8th gen (8265U @ 1.6 GHz)
Installed memory: 8GB RAM, 512 GB SSD
Architecture: 64-bit

Software requirement:
Operating system: Ubuntu 20.04 LTS on oracle virtual machine (3 processors & 4096MB base memory is allocated)
C++ version: C++ 14
Compiler for C++: g++ (version: 10.1.0)
Code editor: Sublime Text (Build 2011)

Theory

Queues

A queue is a linear data structure which follows a particular order in which the operations are performed. The order is FIFO

operations on queue:

Enqueue: Add an item to the queue.
Dequeue: Removes item from queue.
front: Get the front item of queue.
Rear: Get the last item of the queue.

Pseudo Code:

Operations of Queue:

```
Algorithm: isEmpty() {  // checks for empty queue
    return (front == rear);
}

Algorithm isfull() {    // check if queue is full
    return ((rear+1) % D == front);
}

Algorithm front() {     // returns front of queue
    return arr[front+1];
}

Algorithm insert(T x) {  // inserts element in queue
    if (isfull()) return.
    else rear = (rear+1) % D
        arr[rear] = x.
}
```

Algorithm remove() { // removes from queue.
    if (isEmpty()) return.
    else front ← (front+1) % D;
}

* ADT of classes

1) class Job { // This class simulate job of operating system
private:
    string job_title;
    int job_number;
    static int job_count;
public:
    Job(string job="") { // constructor.
        job_title = job;
    }
    void setData() { // fn to set values of private variables
        // code
    }
    void getTitle() { // fn to get title of job
        // code
    }
};

2) template <class T>
class Queue { // Array based implementation of queue.
private:
    T* arr[D];
    int front, rear;
public:
    Queue() { // constructor to initialize data members
    // code

```
bool isEmpty(){    //checks is queue is empty/not
    //code
}
bool isfull(){     //checks if queue is full/not
    //code
}
T front(){         //return front element of queue.
    //code
}
void insert(x){    //insert x into the queue.
    //code
}
void remove(){     //remove first element from queue.
    //code
}
}
```

Analysis of algorithms

Operations of queue: (Assuming Circular queues)

1) Inserting the element is queue.
   This algorithm takes O(1) time & also no extra space is needed.

2) Remove the element from queue:
   Algorithm takes O(1) time & no extra space.

3) Get front of queue:
   Algorithm takes O(1) time & no extra space.

4) checking for empty & full queue.
   both operations required const. space & time.

Testcases

S.No	Testcase	Expected output	Actual Output
1)	Adding job to queue. i/p: job title: job1	job should be added successfully.	Job added successfully.
2)	Adding job to full queue.	The job should not be added.	Got the error message failed to add a job.
3)	Remove the current job.	The current job should be removed from queue.	Job removed successfully.
4)	Showing the current job.	The algorithm should show the current front of queue	the algorithm shows the details of job at front of queue.
5)	Printing all jobs in the queue.	The algorithm should print all the jobs present in the queue.	The algorithm prints all the jobs present in the queue.
6)	Removing job from empty queue.	The algorithm should doesn't do anything to queue. Error message expected.	The error message is as follows: Nothing to work on.

Applications

Queues are useful in

1) BFS of graph
2) when resources are shared among multiple consumers.
   eg CPU scheduling, Disk scheduling
3) when data is transfered asynchronously b/w two processes.
   eg. IO Buffers, pipes, files IO etc.

Conclusion.

This assignment gives insights about behind the scenes implementation of queue data structure which is very helpful & extensively used in the task scheduling of modern, advanced CPU's.