Assignment No 3

Spring 2021

CSE-204 Operating System

Submitted by: Ashfaq Ahmad
Registration No: 19PWCSE1795
Class Section: B

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: _____

Submitted to:
Prof: Tariq Kamal
July 14, 2021

Department of Computer Systems Engineering
University of Engineering and Technology, Peshawar

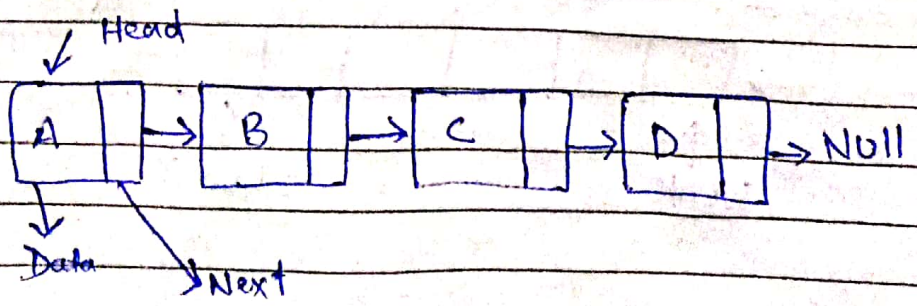
Date: __/__/20

Question No : 1:

→ Linked list:

→ like Array, link list is a linear data Structure.
→ Unlike Array, the linked list elements are not Stored at a Contiguous location;
→ the elements are linked using pointer
→ In linklist elements Store in memory Randomly.
→ it is basically Collection of nodes which together represent a Sequence.
→ each node Contain data and Refrence to the next node in Sequence.
→ this Structure allow for efficient Insertion or removel of elements from any position In the Sequence during Iteration.

Structure of Nodes:



Why linked list??

Arrays can be used to store linear Data of similar types but array have the following limitation.

1) Size of array is fixed, we must know the upper limit of array. Also, generally the allocated memory is equal to the upper limit irrespective of the usage.
2) Inserting the new element in Array is expensive. b/c room has to be created & existing elements have to be shift.

Advantages over Array:

1) Dynamic Size
2) Ease of Insertion/deletion.

Drawback:

→ Random access not allowed
→ Extra memory Required for a pointer.
→ Not cache friendly, Since array elements are Contiguous location there is locality of reference which is not there in case of linked list.

Date: __/__/20

Syntax: C++

```
Class Node {
Public:
    int data;
    Node* Next;
};
```

→ Queues:

→ A queue is a linear Data Structure which follow a particular order in which the operation are performed.

→ It follow FIFO Rule. FIFO stand for first in first out.

→ It is an ordered collection of items where the addition of new items happen at one end called "rear" and Removal of existing items occure at the other end called "front"

→ An Element enter the queue it start at the rear and make its way toward the front, waiting until that the time when

Date: / /20

it is the next element to be removed.

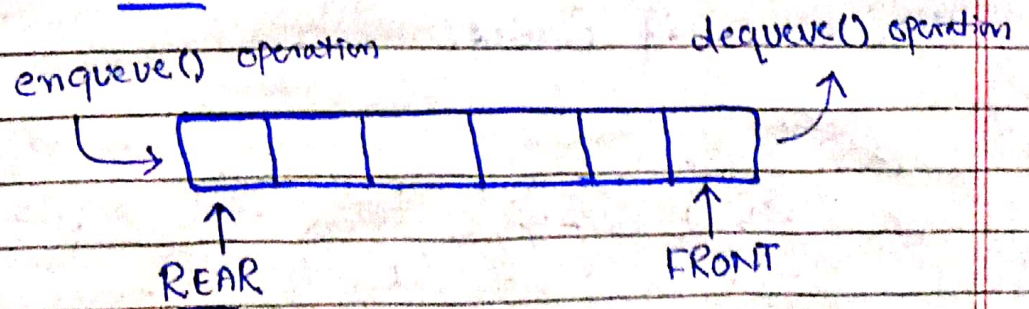
→ the most recently added item in the queue must wait at the End of collection.

→ the process to add an element to queue called **Enqueue** and the process of Removal of an Element from queue is called **Dequeue**.

General Example:

if we go to the ticket Counter to buy movie ticket, and are first in the queue the we will be the first one to get the ticket. Same is the Case with Queue Data structure. Data Inserted first will leave the queue first.

Structure:



Date: __/__/20

Implementation of Queue:

→ Queue can be Implemented using an Array, Stack or linklist

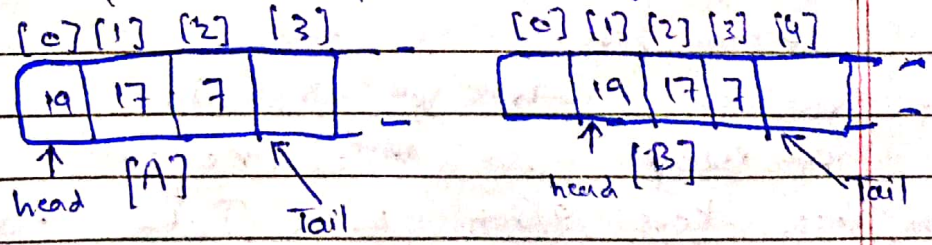
→ the easy way to Implement by using Array.

→ Initially both Rear & front point the first Index of array.

→ As element add to Queue Rear moving ahead, always pointing to the position where the next element will be Inserted while the head will remain at the first Index.

→ When we remove the element from Queue we can follow two possible Approaches given below

e.g.



→ In [A] apporach we remove the element at head position and then one by one Shifting all other elements in forward position.

→ In Approach [B] we remove the element from head position and then move head to the next position.

Date: __/__/20

Syntax: C++:

```
Class queue {
    int a[size];
    int rear;        or tail
    int front;       or head.
Public:
    void enqueue (int x);
    int dequeue ();
    void display ();
};
```

→ MAP Data Structure:

→ A map is a type of fast key lookup data structure that offer a flexible mean of Indexing into its Individual elements.

→ Indices into the elements of a map are called keys.

→ These keys along with Data values associated with them, are stored within the map.

→ Each entry of a map Contain Exactly one unique key and its Corresponding value.

→ Maps are typically Implemented as binary Search tree.

Date: __/__/20__

MAP Data type Characteristics:

MAP Data have a few unique features

→ the keys are unique so no duplicate key are possible.

e.g

if we need to list the name of all the town of ~~washington~~ In United State and population of each town, the map data cannot be used b/c the name of town repeat. the town of Washington repeat in Washington State, New Jersey and Wisconsin.

When data is already saved into a map data data type, there are three ways in which the Date can be viewed.

i) You can view only the list of keys
ii) " " " " " " values
iii) " " " both keys & values as matching pairs.

Note:

Zero sized map are also valid. In that case map.begin() and map.end() point to same location.

Date: __/__/20__

Definition:

Below is the definition of std::map from <map> header file.

```
template <Class key,
          class T,
          class Compare = less<key>,
          class Alloc = allocator<pair
                        <Const key, T>>
          > class map;
```

Parameters:

Key – Type of the key

T – Type of the mapped values.

Compare – A binary predicate that take two element keys as arguments and return a bool).

Alloc – Type of the Allocator object.

Q NO: 2

* Linklist as a Part of Kernel Data Structure:

Example of linklist as a Part of Kernel Data Structure is circuler linklist. when multiple application are running on PC. it is Commen for Kernel to put the running application on a list and then to cycle through them to cycle Through them, giving each of them a Slice of time to execute and then making them wait while the CPU is given to another Application. it is Convenient for operating System to use a Circuler linklist So that when it reaches the end of the list it can cycle around to the front of the list.

Queue as a Part of Kernel Data Structure:

Work queue has been Introduced in linux 2.6 and replace a Similer

Construct called task queue used in linux 2.4. they allow kernel function to be activated and later executed by special kernel thread called worker thread.

Map as a part of Kernel Data Structure:

the kernel provide a map data structure called idr. idr is used for mapping user space UIDs like POSIX timer IDs, to their associated data structure [1, P. 100] we can define an idr by either statically or dynamically allocating an idr struct, then calling idr_init();
the next step is to allocate a new UID. first we do this by telling the idr that we want a new UID [which let it resize the underlying tree if needed.] then we make request for UID [1, P. 101]. the reason for these two steps process is so that the initial call, which can result in additional memory allocation can run without requiring a lock.