**Assignment # 3**



Operating system

Submitted by: **Muhammad Ali**

Registration No: - **19PWCSE1801**

Class Section: **A**

“On my honor, as student of University of Engineering and Technology, I have

neither given nor received unauthorized assistance on this academic work.”

Submitted to:

**Sir Tariq Kamal**

Data:(18,07,2021)

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar.

**Q1. Describe in detail the Linked Lists, Queues, and Map data structures.**

**linked list: -**

 A linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element [points](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) to the next. It is a [data structure](https://en.wikipedia.org/wiki/Data_structure) consisting of a collection of [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) which together represent a [sequence](https://en.wikipedia.org/wiki/Sequence). In its most basic form, each node contains: [data](https://en.wikipedia.org/wiki/Data_(computing)), and a [reference](https://en.wikipedia.org/wiki/Reference_(computer_science)) (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing more efficient insertion or removal of nodes at arbitrary positions. A drawback of linked lists is that access time is linear (and difficult to [pipeline](https://en.wikipedia.org/wiki/Instruction_pipelining)). Faster access, such as random access, is not feasible. [Arrays](https://en.wikipedia.org/wiki/Array_data_structure) have better [cache locality](https://en.wikipedia.org/wiki/Locality_of_reference) compared to linked lists.

**Singly linked list: -**

Singly linked lists contain nodes which have a data field as well as 'next' field, which points to the next node in line of nodes. Operations that can be performed on singly linked lists include insertion, deletion and traversal.

**Doubly linked list: -**

In a 'doubly linked list', each node contains, besides the next-node link, a second link field pointing to the 'previous' node in the sequence. The two links may be called 'forward('s') and 'backwards', or 'next' and 'prev'('previous').

### Multiply linked list: -

In a 'multiply linked list', each node contains two or more link fields, each field being used to connect the same set of data records in a different order of same set (e.g., by name, by department, by date of birth, etc.). While doubly linked lists can be seen as special cases of multiply linked list, the fact that the two and more orders are opposite to each other leads to simpler and more efficient algorithms, so they are usually treated as a separate case.

### Circular linked list: -

In the last node of a list, the link field often contains a [null](https://en.wikipedia.org/wiki/Null_pointer#Null_pointer) reference, a special value is used to indicate the lack of further nodes. A less common convention is to make it point to the first node of the list; in that case, the list is said to be 'circular' or 'circularly linked'; otherwise, it is said to be 'open' or 'linear'. It is a list where the last pointer points to the first node.

**STACKS**: -  
A stack is used to provide temporary storage space for values. It is defined as a data structure which operates on a first in, last out basis. Its uses a single pointer (index) to keep track of the information in the stack.

The basic operations associated with a stack are,

* insert (push) an item onto the stack
* remove (pop) an item from the stack
* Pushing items onto the stack  
  The stack pointer is considered to be pointing to a free (empty) slot in the stack. A push operation thus involves copying data into the empty slot of the stack, then adjusting the stack pointer to point to the next free slot.

Removing items from the stack  
To remove an item, first decrement (subtract 1 from) the stack pointer, then extract the data from that position in the stack.

**Queue: -**

Queue is used to simulate the data structure of "queue" (first in first out FIFO). The head of the queue holds the element with the longest storage time in the queue, and the tail of the queue holds the element with the shortest storage time in the queue. The new element is inserted (offer) at the end of the queue, and the poll operation returns the element at the head of the queue. The queue does not allow random access to the elements in the queue. Queue is a first-in, first-out container. LinkedList implements the queue interface, so we can construct queue through LinkedList through downward transformation.  
Everybody has experience with queues as they are common place. Queues occur in cafeterias, petrol stations, shopping centres, anyplace where many people (customers) line up for few resources (cashier's, salespeople, petrol pumps etc).

The purpose of a queue is to provide some form of buffering. Typical uses of queues in computer systems are,

* process management
* buffer between the fast computer and a slow printer

A queue is defined as a data structure which holds a series of items to be processed on a first in first out basis (though some queues can be sorted in priority). The operations associated with a queue are,

* initialize the queue
* insert an item in the queue
* remove an item from the queue
* count the number of empty slots in the queue
* count the number of items in the queue

**Map data structure: -**

A Map is a type of fast key lookup data structure that offers a flexible means of indexing into its individual elements. Unlike most array data structures in the MATLAB® software that only allow access to the elements by means of integer indices, the indices for a Map can be nearly any scalar numeric value or a character vector.

Indices into the elements of a Map are called keys. These keys, along with the data values associated with them, are stored within the Map. Each entry of a Map contains exactly one unique key and its corresponding value. Indexing into the Map of rainfall statistics shown below with a character vector representing the month of August yields the value internally associated with that month, 37.3.

**Q2. Show example uses of Linked Lists, Queues, and Maps that the kernel uses (as part of kernel data structures) to control and manage processes and other system resources.**

**Note:** In kernel code use list\_for\_each instead of \_\_list\_for\_each

#### The Example

#include <stdio.h>

#include <assert.h>

#include <cstdlib.h>

#include "list.h"

struct foo {

int info;

struct list\_head list\_member;

};

void add\_node(int arg, struct list\_head \*head)

{

struct foo \*fooPtr = (struct foo \*)malloc(sizeof(struct foo));

assert(fooPtr != NULL);

fooPtr->info = arg;

INIT\_LIST\_HEAD(&fooPtr->list\_member);

list\_add(&fooPtr->list\_member, head);

}

void display(struct list\_head \*head)

{

struct list\_head \*iter;

struct foo \*objPtr;

\_\_list\_for\_each(iter, head) {

objPtr = list\_entry(iter, struct foo, list\_member);

printf("%d ", objPtr->info);

}

printf("\n");

}

void delete\_all(struct list\_head \*head)

{

struct list\_head \*iter;

struct foo \*objPtr;

redo:

\_\_list\_for\_each(iter, head) {

objPtr = list\_entry(iter, struct foo, list\_member);

list\_del(&objPtr->list\_member);

free(objPtr);

goto redo;

}

}

int find\_first\_and\_delete(int arg, struct list\_head \*head)

{

struct list\_head \*iter;

struct foo \*objPtr;

\_\_list\_for\_each(iter, head) {

objPtr = list\_entry(iter, struct foo, list\_member);

if(objPtr->info == arg) {

list\_del(&objPtr->list\_member);

free(objPtr);

return 1;

}

}

return 0;

}

main()

{

LIST\_HEAD(fooHead);

add\_node(10, &fooHead);

add\_node(20, &fooHead);

add\_node(25, &fooHead);

add\_node(30, &fooHead);

display(&fooHead);

find\_first\_and\_delete(20, &fooHead);

display(&fooHead);

delete\_all(&fooHead);

display(&fooHead);

}

### Kernel uses of queen: -

### Scheduling Queues: -

* All processes are stored in the **job queue.**
* Processes in the Ready state are placed in the **ready queue.**
* Processes waiting for a device to become available or to deliver data are placed in **device queues**. There is generally a separate device queue for each device.
* Other queues may also be created and used as needed.

## **Structures used for memory mapping: -**

Before discussing the mechanism of memory-mapping a device, we will present some of the basic structures related to the memory management subsystem of the Linux kernel.

Before discussing about the memory mapping mechanism over a device, we will present some of the basic structures used by the Linux memory management subsystem. Some of the basic structures are: struct page, struct vm\_area\_struct, struct mm\_struct.

### struct page

struct page is used to embed information about all physical pages in the system. The kernel has a struct page structure for all pages in the system.

There are many functions that interact with this structure:

* virt\_to\_page() returns the page associated with a virtual address
* pfn\_to\_page() returns the page associated with a page frame number
* page\_to\_pfn() return the page frame number associated with a struct page
* page\_address() returns the virtual address of a struct page; this functions can be called only for pages from lowmem
* kmap() creates a mapping in kernel for an arbitrary physical page (can be from highmem) and returns a virtual address that can be used to directly reference the page