Assignment 3 – Bo Henderson and Burde Prerana Kamath

Question 1

The doubly linked list was implemented with four functions. The FIFO operations which comprises of enqueue and dequeue was implemented followed by a priority enqueue function and a display function.

The structure consists of process id and the key value. The priority enqueue function was implemented using the key value as the priority. It also consisted of next and previous pointers that were local to every node and three global pointers that was current, front and rear.

Xinu uses arrays as the base structure whereas the program uses structures as the base structure. Since XINU uses relative pointers and has an implicit data structure such as the queuetab array, it takes less memory space and processor time. Since our implementation consists of structures using pointers there are some downfalls such as:

1 – Pointers always take up extra memory space.

2 – Since arrays have a better cache locality, they perform in a shorter time than our implementation and hence, their performance and memory requirements is better.

3 – Linked lists cannot be accessed randomly. As seen in the implementation, they need to be accessed using the front pointer always which doesn't hold true for arrays since they have already been assigned a particular amount of memory space since its static memory allocation and can be accessed through their indices.

Pseudocode of the program:

The structure of the program is of the type:-

```
struct node
{
        int pid; //This is the process id
        int key; //This is the key value according to which priority is assigned
        //This is to maintain the connection among nodes. Every node will know the previous node and the node after it.
        struct node *next;
        struct node *prev;
};
struct node *current = NULL; //This is to know where the current position of the queue is present.
struct node *rear = NULL; //This is to know the last node in the queue.
struct node *front = NULL; //This is to know the first node in the queue.
```

The FIFO functions implemented were enqueue and dequeue.

Pseudocode for enqueue:

1. Algorithm – FIFO Enqueue
2. Input – Process ID, Key Value
3. Output – A node has been entered in the queue
4. *** pid – processid, key – keyvalue
5. ***link – new link to enqueue
6. struct node *link ← (struct node*) malloc (sizeof(struct node))
7. if front = NULL *//If the queue is empty*
8. front ← rear ← link
9. endif
10. else *//Pointers of the rear should point to the new link*
11. rear ← next ← link

12. link ← prev ← rear
13. end else
14. rear ← link *//The new addition is the rear node now*

Pseudocode for dequeue:

1. Algorithm – FIFO Dequeue
2. Input – Process ID, Key Value
3. Output – A node has been deleted from the queue
4. *** pid – processid, key – keyvalue
5. ***temp – the node to dequeue
6. ***tempo – temporary variable
7. struct node* temp
8. if front = NULL *//The queue is empty*
9. temp = NULL
10. end if
11. else if front -> next = NULL //The queue has only one node
12. struct node *tempo
13. temp ← tempo
14. front ← NULL
15. rear ← NULL
16. end elseif
17. else *//When there is more than one node in the queue*
18. struct node* tempo ← front
19. temp ← tempo
20. front ← front -> next
21. front -> prev ← NULL
22. end else
23. return temp *//To print the information of the deleted node*

Pseudocode for priority enqueue:

```
1.  Algorithm – Priority Enqueue
2.  Input – Process ID, Key Value
3.  Output – A node has been entered in the queue
4.  *** pid – processid, key – keyvalue
5.  ***link – new link to enqueue
6.  struct node *link ← (struct node*) malloc (sizeof(struct node))
7.  if front = NULL                          //If the queue is empty
8.          front ← rear ← link
9.   endif
10.  else
11.         if link -> key > find -> key      //The node has most priority
12.                 front -> prev ← link
13.                 link -> next ← front
14.                 front ← link
15.         end if
16.         else
17.                 while                     //Find the position per priority
18.                         if find -> next ← NULL
19.                                 break
20.                         endif
21.                         find ← find -> next
22.                 end while
23.                 if find -> next ← NULL and link -> key < find -> key
24.                         rear -> next ← link
25.                         link -> prev ← rear
26.                         rear ← link
27.                 endif
```

```
28.             else
29.                     find -> next -> prev ← link
30.                     link -> prev ← find -> prev
31.                     link -> next ← find
32.                     find -> prev ← link
33.             end else
34.     end else
35. end else
```

Question 2:

In XINU, a valid queue id is defined in the queue.h file.

Valid queue id's are:

- It must be positive
- It must lie between 0 and NQENT – 2 (both inclusive)

NQENT = NPROC + 4 +NSEM + NSEM

NPROC – Number of processes.

NSEM – Number of semaphores.

These are hardcoded values in XINU.

The code changes were made in the getitem.c file present in the system folder of xinu.

The changes were made using the function isbadqid(q) where isbadqid was a function defined in the queue.h file present in the include folder of the Xinu. The functions getfirst and getlast had the check included.

An appropriate check for getitem was made using the pid where the function isbadpid(pid) was also defined in the include folder queue.h.

These checks were included in the getitem.c

A short function was added in each of these functions.

```
if(isbadqid(q))
{
printf(" The queue id is not valid");
return SYSERR;
}
```

Question 3

The critical piece of the assembly code is shown below. As you can see, the two sets of commands are identical up until command 38 at which point two lines are inserted in the after version:

```
3c:     e3500007        cmp   r0, #7
40:     918c00b3        strhls   r0, [ip, r3]
```

After these two lines, the pre-modification and post-modified resched commands go back to being identical. These two lines make sense given the high-level C code that was added to resched:

```
    if (disposition >= 0 && disposition <= 7) {
      ptold->prstate = disposition;
       }
```

We added two operations: a conditional statement followed by an assignment. It should be noted however, that this modification is not ideal as it needs to be negated in several situations by passing a disposition to resched() that is outside the above conditional.

Before the modification to resched():

...

```
 30:  e1a0c18c   lsl     ip, ip, #3
 34:  e59f3094   ldr     r3, [pc, #148]      ; d0 <resched+0xd0>
 38:  e08c4003   add   r4, ip, r3
 3c:  e19c20b3   ldrh   r2, [ip, r3]
 40:  e3520001   cmp   r2, #1
 44:  1a00000d   bne   80 <resched+0x80>
```

...

vs

After the modification to resched():

...

```
 30:  e1a0c18c   lsl     ip, ip, #3
 34:  e59f30a0   ldr     r3, [pc, #160]      ; dc <resched+0xdc>
 38:  e08c5003   add   r5, ip, r3
```

```
  3c:  e3500007  cmp  r0, #7
  40:  918c00b3  strhlsr0, [ip, r3]
  44:  e19c20b3  ldrh   r2, [ip, r3]
...
```