3.1 Introduction

List processing is fundamental for operating system since one needs

- Lists of processes to be **scheduled (ready for scheduling)**
- Lists of processes **waiting on a semaphore**
- Lists of processes ordered by **priority (time)** in order to implement **time-sharing (sleep list)**.

3.2 Linked Lists Of Processes (Q Structure)

- **FIFO queues** are ordered by **time of insertion**.
- **Priority queues** are ordered by **time to wake up** (in view of time-sharing).

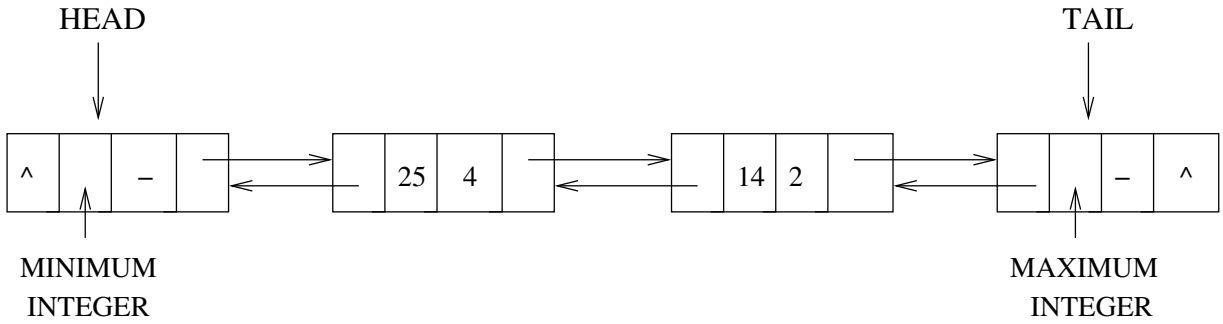


Figure 3.1: Adoubly linked list containing 4 (kek=25) and 2 (keu=14)

	key	next	prev
0			
1			
2	14	33	4
3			
4	25	2	32
5			
NPROC−1			
NPROC			
Head at 32	MINIMUM	4	−1
Tail at 33	MAXIMUM	−1	2

Figure 3.2: The list from Figure 3.1 sorted in the q array

3.3 Implementation Of The Q Structure

```
/* q.h - firstid, firstkey, isempty, lastkey, nonempty */

/*q structure declarations, constants, and inline procedures*/

#ifndef NQENT
#define NQENT    NPROC + NSEM + NSEM + 4/* for ready & sleep*/
#endif
struct  qent   {        /* one for each process plus tow for*/
                        /* each list                        */
        short  qkey;    /* key on whichthe queue is ordered */
        short  qnext;   /* pointer to next process or tail  */
        short  qprev;   /*pointer to prevous process or head*/
        };

extern  struct  qent q[];
extern  int     nextqueue;
/* inline list manipulation procedures */

#define isempty(list)  (q[(list)].qnext >= NPROC)
#define nonempty(list) (q[(list)].qnext < NPROC)
#define firstkey(list) (q[q[(list)].qnext].qkey)
#define lastkey(tail)  (q[q[(tail)].qprev].qkey)
#define firstid(list)  (q[(list)].qnext)
#define EMPTY   -1      /* equivalent of null pointer       */
```

3.3.1 Operations On FIFO And Priority Queue

- **inserting** an item at the tail of a list
- **inserting** an item within an ordered list
- **removing** an item at the head of a list
- **creating** a new list.

programming this operations is simple because only one process executes these operations at a time.

This is due to the fact that they occur only in **critical sections** (programmed by forbidding interrupts in the field of the process status reg.)

- Elements extracted from the **FIFO** queue by removing them from the head, hence they are inserted at its tail as shown in figure 3.3
- If list is a priority queue:
  - *getfirst(head)* removes item with smallest key
  - *getlast(tail)* removes item with biggest key
- Hence priority queue insertion starts at head of the priority queue as shown in figure 3.4

```
#include <conf.h>
#include <kernel.c>
#include <q.h>

/*------------------------------------------------------------
 * enqueue -- insert an item at the tail of a list
 *------------------------------------------------------------
 */
int     enqueue(item, tail);
        int    item; /* item to enqueue on a list        */
        int    tail; /* index in q of list tail          */
{
        struct  qent    *tptr; /* points to tail entry   */
        struct  qent    *mptr; /* points to item entry   */

        tptr = &q[tail];
        mptr = &q[item];
        mptr->qnext = tail;
        mptr->qprev = tptr->qprev;
        q[tptr->qprev].qnext = item;
        tptr->qprev = item;
        return(item);
}
```

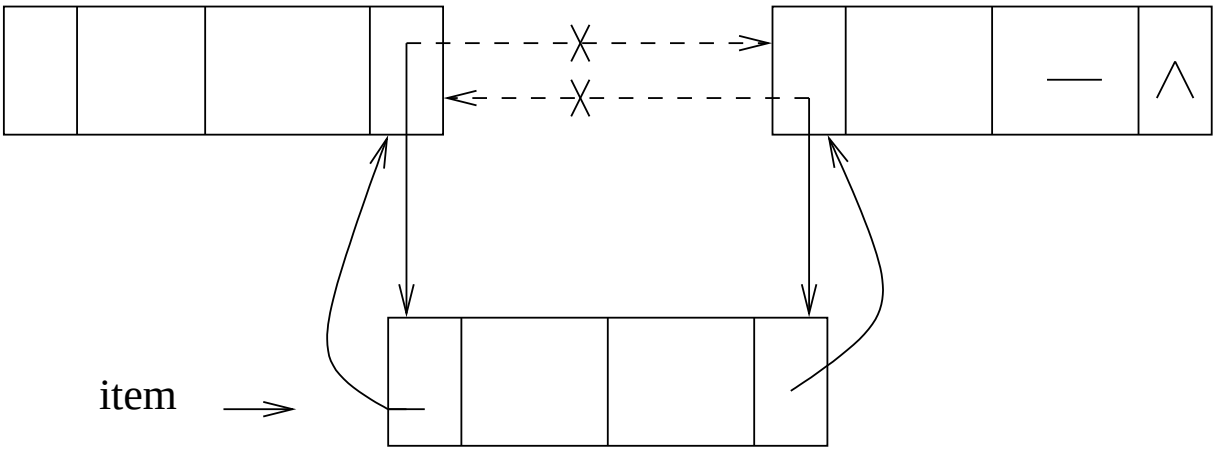
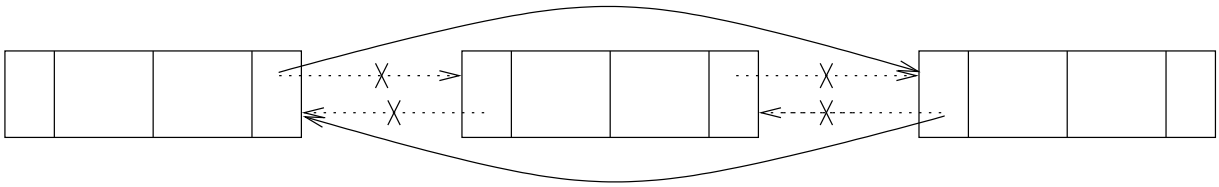


Figure 3.3: Enqueue an item at the tail

```
#include <conf.h>
#include <kernel.c>
#include <q.h>

/*------------------------------------------------------------
 * dequeue -- removes an item from a list and return it
 *------------------------------------------------------------
 */
int     dequeue(item, tail);
        int    item;       /* item to dequeue from a list */

{
        struct  qent    *mptr; /* pointer to q entry item */

        mptr = &q[item];
        q[mptr->qprev].qnext = mptr->qnext;
        q[mptr->qnext].qprev = mptr->qprev;
        return(item);
}
```



dequeue(item)

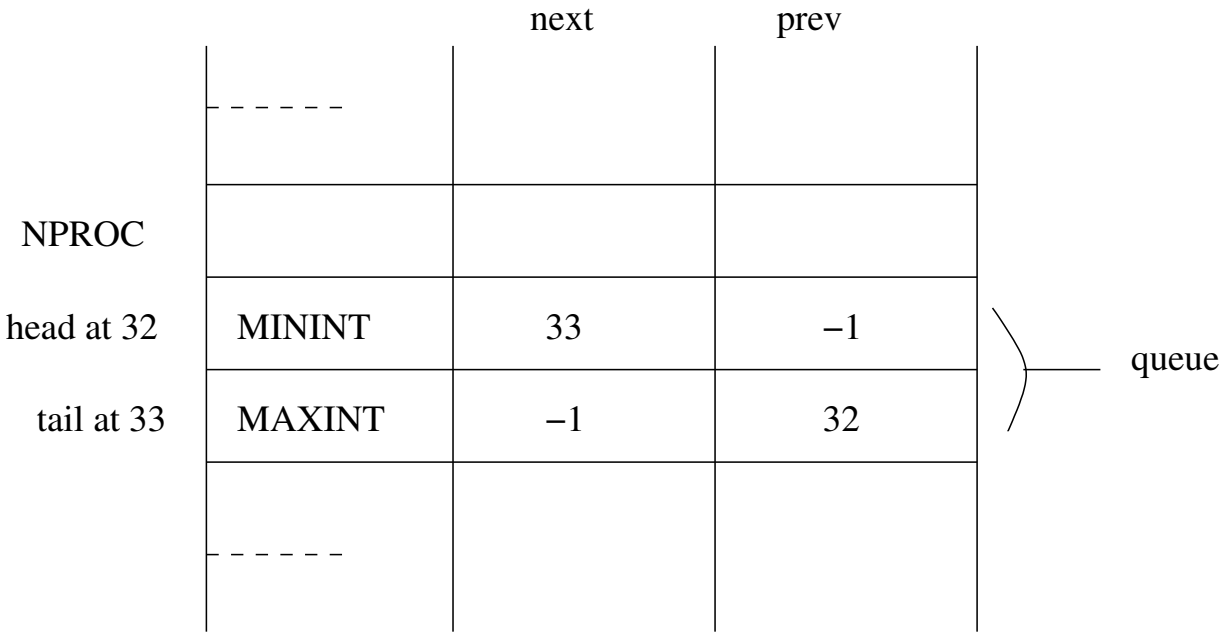
Figure 3.4 : Dequeue an item at the head of the queue

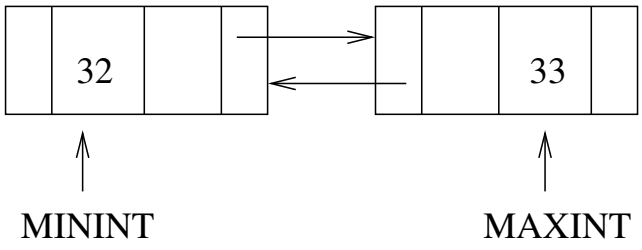
```
/* insert.c - insert */
#include <conf.h>
#include <kernel.h>
#include <q.h>
/*------------------------------------------------------------
 * insert.c -- insert an process into a q list in key order
 *------------------------------------------------------------
 */
int     insert(proc, head, key);
        int    proc;        /* process to insert          */
        int    head;        /* q index of head of list    */
        int    key;         /* key to use for this process*/
{
        int    next;        /* runs through list          */
        int    prev;
        next = q[head].qnext;
        while (q[next.qkey < key)
                next = q[next].qnext;
        q[proc].qnext = next;
        q[proc].qprev = prev = q[next].qprev;
        q[proc].qkey  = key;
        q[proc].qnext = proc;
        q[next].qprev = proc;
        return(OK);
}
```

```
#include <conf.h>
#include <kernel.h>
#include <q.h>

/*------------------------------------------------------------
 * getfirst -- removes and return the first process on a list
 *------------------------------------------------------------
 */
int     getfirst(head)
        int     head;   /* q index of head of list         */
{
        int     proc;   /* first process on the list       */

        if ((proc=q[head].qnext) < NPROC)
                return( dequeue(proc) );
        else
                return(EMPTY);
}
```

3.4 List Initialization

```
#include <conf.h>
#include <kernel.h>
#include <q.h>
/*------------------------------------------------------------
 * newqueue -- initialize a new list in the q structure
 *------------------------------------------------------------
 */
int     newqueue()
{
        struct  qent    *hptr; /* address of new list head */
        struct  qent    *tptr; /* address of new list tail */
        int     hindex, tindex;/* head and tail indexes    */
                            /* nextqueue is global variable */
        hptr = &q[hindex=nextqueue++];
                            /* gives next used q pos.       */
        tptr = &q[tindex=nextqueue++];
        hptr->qnext = tindex;
        hptr->qprev = EMPTY;
        hptr->qkey  = MININT;
        tptr->qnext = EMPTY;
        tptr->qprev = hindex;
        tptr->qkey  = MAXINT;
        return(hindex);
}
```





An eaxample of a new, empty Q−structure, created by calling newqueue()