1.Using doubly linked list gives more flexibility in data storage. We can store different type of data with different size because it only changes the objects that the pointers are pointing to. Using an array we have to set the every element with the same data type and size. It's less flexible and efficient to use the memory space. Even we don't use every element of the array it still use some memory space. The linked list also grows fast in queue operation (enque and deque) while the size of array is fixed if we want to increase the size of array we need to recreate it.Thus, a linked list is more memory efficient and flexible than an array. However, the linked list is not good at accessing random elements. To access a particular element by a given value or key we have to iterate through the whole list to find the element, which is not efficient if the size of list is huge. In the array we can directly access one element by the indices.

The output of the demo:

```
[qiuwshou@silo assignment_3]$ ./fifo
here is my queue
process_id:1,key_value:99,priority:10
process_id:2,key_value:98,priority:10
process_id:3,key_value:97,priority:10
deque one process.
process_id:2,key_value:98,priority:10
process_id:3,key_value:97,priority:10
insert with different priority.
process_id:4,key_value:96,priority:20
process_id:2,key_value:98,priority:10
process_id:3,key_value:97,priority:10
process_id:5,key_value:95,priority:5
```

The enque function:

```
//extract after the head - equals to deletion
void deque(){
  struct node *curr = head->next;
  struct node *next = curr->next;
  if(is_empty() == 1 ){ // check if the queue is empty
    return;
  }
  else{
    head->next = next;
    next->prev = head;
  }
}
```

The deque funcion:

```c
//insert before the tail
void enque(int pid, int priority){
  struct node *curr = newNode(pid, priority);
  struct node *temp = tail->prev;
  if(is_empty() == 1 ){ // check if the queue is empty
    head->next = curr;
    tail->prev = curr;
    curr->next = tail;
    curr->prev = head;
  }
  else{
    curr->prev = temp;
    curr->next = tail;
    tail->prev = curr;
    temp->next = curr;
  }
  return;
}
```

The insert function based on the priority:

```c
//insert  by the priority
void insert(int pid,int priority){
  struct node *curr = newNode(pid, priority);
  if(is_empty() == 1) { //insert the process if queue is empty
    head->next = curr;
    tail->prev = curr;
    curr->prev = head;
    curr->next = tail;

  }
  else{ // insert after the process wih higher prioriy
    struct node *temp = head->next;
    while(temp->key_value < NUM_PROCESS){
      if(temp->priority <= priority){
        struct node *temp_prev = temp->prev;
        temp_prev->next = curr;
        temp->prev = curr;
        curr->next = temp;
        curr->prev = temp_prev;
        return;
      }
      else{
        temp = temp->next;
        if(temp->key_value >= NUM_PROCESS){
          struct node * temp_prev = temp->prev;
          temp->prev = curr;
          temp_prev->next = curr;
          curr->next = temp;
          curr->prev = temp_prev;
        }
      }
    }
```

2.

There are a couple of things we need to check for a valid queue ID. By given a queue id first we need to check that the queue is not empty so that we get the information from it. Moreover, the Xinu handles a limited number MAX of queues. We also need to check that the queue id is not larger than MAX-1. The following function is not debugged.

```
/* getitem.c - getfirst, getlast, getitem */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  getfirst  -  Remove a process from the front of a queue
 *------------------------------------------------------------------------
 */
#define MAX 10; /* use 10 for demo*/

int is_valid(qid16 q){
  if(q < MAX - 1 ){
    return 1;
  }
  else{
    return 0;
  }
}


pid32   getfirst(
          qid16         q               /* ID of queue from which to    */
        )                               /* Remove a process (assumed    */
                                        /*   valid with no check)       */
{
        pid32   head;

        if (isempty(q)) {
                return EMPTY;
        }

        if(is_valid(q)){
          head = queuehead(q);
          return getitem(queuetab[head].qnext);
        }
}
```

3.

We explicitly specifies the disposition of the current process.In the demo we assign it to the PR_READY. In the real case, the disposition can be read from the argument. During the reschedule the current process with disposition of PR_READY or PR_CURR will be add to the ready list for conext switch.

```
*/
void    resched(void)           /* Assumes interrupts are disabled      */
{
        struct procent *ptold;  /* Ptr to table entry for old process   */
        struct procent *ptnew;  /* Ptr to table entry for new process   */

        unit16 disposition = PR_READY;     /*explicit dispostion of the current process, can be given by argument, user ready for demo*/

        /* If rescheduling is deferred, record attempt and return */

        if (Defer.ndefers > 0) {
                Defer.attempt = TRUE;
                return;
        }
        /* change the state of current process by its disposition*/

        ptold = &proctab[currpid];
        if(disposition == PR_CURR || disposition == PR_READY){
          if(ptold->prprio > firstkey(readylist)){
            return;
          }
          ptold->prstate = disposition;
          insert(currpid,readylist,ptold->prprio);
        }
```

The assembly code of old version of resched.c has 188 instruction.

```
        /* Point to process table entry for the current (old) process */

        ptold = &proctab[currpid];
  24:   e59f30a0        ldr     r3, [pc, #160]  ; cc <resched+0xcc>
  28:   e5930000        ldr     r0, [r3]
  2c:   e060c180        rsb     ip, r0, r0, lsl #3
  30:   e1a0c18c        lsl     ip, ip, #3
  34:   e59f3094        ldr     r3, [pc, #148]  ; d0 <resched+0xd0>
  38:   e08c4003        add     r4, ip, r3

        if (ptold->prstate == PR_CURR) {  /* Process remains eligible */
  3c:   e19c20b3        ldrh    r2, [ip, r3]
  40:   e3520001        cmp     r2, #1
  44:   1a00000d        bne     80 <resched+0x80>
                if (ptold->prprio > firstkey(readylist)) {
  48:   e1d420f2        ldrsh   r2, [r4, #2]
  4c:   e59f1080        ldr     r1, [pc, #128]  ; d4 <resched+0xd4>
  50:   e1d110b0        ldrh    r1, [r1]
  54:   e59f507c        ldr     r5, [pc, #124]  ; d8 <resched+0xd8>
  58:   e6bf6071        sxth    r6, r1
  5c:   e0856186        add     r6, r5, r6, lsl #3
  60:   e1d660f4        ldrsh   r6, [r6, #4]
  64:   e7955186        ldr     r5, [r5, r6, lsl #3]
  68:   e1520005        cmp     r2, r5
  6c:   c8bd8070        popgt   {r4, r5, r6, pc}
                        return;
                }

                /* Old process will no longer remain current */

                ptold->prstate = PR_READY;
  70:   e3a0e002        mov     lr, #2
  74:   e18ce0b3        strh    lr, [ip, r3]
                insert(currpid, readylist, ptold->prprio);
  78:   e6bf1071        sxth    r1, r1
  7c:   ebfffffe        bl      0 <insert>
        }
```

The new version has 178 instructions. Only the difference is shown.

```
      }
      /* change the state of current process by its disposition*/
      ptold = &proctab[currpid];
  24:   e59f3090        ldr     r3, [pc, #144]  ; bc <resched+0xbc>
  28:   e5930000        ldr     r0, [r3]
  2c:   e0605180        rsb     r5, r0, r0, lsl #3
  30:   e1a05185        lsl     r5, r5, #3
  34:   e59fc084        ldr     ip, [pc, #132]  ; c0 <resched+0xc0>
  38:   e085400c        add     r4, r5, ip

      if(disposition == PR_CURR || disposition == PR_READY){
        if(ptold->prprio > firstkey(readylist)){
  3c:   e1d420f2        ldrsh   r2, [r4, #2]
  40:   e59f307c        ldr     r3, [pc, #124]  ; c4 <resched+0xc4>
  44:   e1d310b0        ldrh    r1, [r3]
  48:   e59f3078        ldr     r3, [pc, #120]  ; c8 <resched+0xc8>
  4c:   e6bf6071        sxth    r6, r1
  50:   e0836186        add     r6, r3, r6, lsl #3
  54:   e1d660f4        ldrsh   r6, [r6, #4]
  58:   e7933186        ldr     r3, [r3, r6, lsl #3]
  5c:   e1520003        cmp     r2, r3
  60:   c8bd8070        popgt   {r4, r5, r6, pc}
          return;
        }
        ptold->prstate = disposition;
  64:   e3a06002        mov     r6, #2
  68:   e18560bc        strh    r6, [r5, ip]
        insert(currpid,readylist,ptold->prprio);
  6c:   e6bf1071        sxth    r1, r1
  70:   ebfffffe        bl      0 <insert>
      //     insert(currpid, readylist, ptold->prprio);
      //}
```