Q)Implement functions to manipulate lists using pointers instead of indices into an array of structures. **What is the difference in memory space and processor time?** To be more specific, implement a doubly linked list structure with the same function as in Xinu. The node in the list should contain information about process id and its key value. Also implement those functions that manipulate lists as in Xinu, which includes the basic functions to manipulate lists, FIFO queue operations(enque,deque) and the priority queue operations. **Compare the difference in memory space between your implementation and the Xinu design.**

Ans)
1)Memory Space- While using indices, it uses 8 bytes of memory to assign qfirst index, qnext index and a qid.
While using pointers, it uses 16 bytes of memory . Implementation of pointers uses extra space because a new variable proc_id of data type pid32 has been introduced as a reference.

2)**Processor Time-** The pointer indexing is faster than the array indexing because the pointers already have the literal address which directly points to the desired location . While, in array indexing, the index value has to iterate through the whole array before reaching the desired position.

Q)Functions getfirst, getlast and getitem do not check whether their argument is a valid ID. Modify the code to insert the appropriate checks. **Please explain what it means to be a valid queue ID in Xinu.**

Ans) For a queue id to be valid ,it should exist between the implicit boundaries. A valid queue id must be greater than ***0*** and less than N**PROC+4+NSEM+NSEM** which is the value that allocates NPROC processes plus head and tail pointers for NSEM semaphore lists , a ready list and a sleep list.

Q)Rewrite resched to have an explicit parameter giving the disposition of the currently executing process, and examine the assembly code generated to determine the number of instructions executed in each case. The disposition of the current process is its next state. The current process will remain eligible to run if the next state is READY or CURR. You could use the "arm-none-eabi-objdump -S resched.o" command to get the generated assembly code for the C code.

Ans) Binary file for resched.c

```
resched.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <resched>:
/*------------------------------------------------------------------------
 *  resched  -  Reschedule processor to highest priority eligible process
 *------------------------------------------------------------------------
 */
void   resched(void)                /* Assumes interrupts are disabled      */
{
   0:   e92d4070    push   {r4, r5, r6, lr}
```

```
        struct procent *ptold;      /* Ptr to table entry for old process      */
        struct procent *ptnew;      /* Ptr to table entry for new process      */

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

        if (Defer.ndefers > 0) {
 4:     e59f30c4    ldr     r3, [pc, #196]      ; d0 <resched+0xd0>
 8:     e5933000    ldr     r3, [r3]
 c:     e3530000    cmp     r3, #0
10:     da000003    ble     24 <resched+0x24>
                Defer.attempt = TRUE;
14:     e3a02001    mov     r2, #1
18:     e59f30b0    ldr     r3, [pc, #176]      ; d0 <resched+0xd0>
1c:     e5c32004    strb    r2, [r3, #4]
                return;
20:     e8bd8070    pop     {r4, r5, r6, pc}
        }

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

        ptold = &proctab[currpid];
24:     e59f30a8    ldr     r3, [pc, #168]      ; d4 <resched+0xd4>
28:     e5930000    ldr     r0, [r3]
2c:     e060c180    rsb     ip, r0, r0, lsl #3
30:     e1a0c18c    lsl     ip, ip, #3
34:     e59f309c    ldr     r3, [pc, #156]      ; d8 <resched+0xd8>
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:     1a00000f    bne     88 <resched+0x88>
                if (ptold->prprio > firstkey(readylist)) {
48:     e1d420f2    ldrsh   r2, [r4, #2]
4c:     e59f1088    ldr     r1, [pc, #136]      ; dc <resched+0xdc>
50:     e1d110b0    ldrh    r1, [r1]
54:     e59f5084    ldr     r5, [pc, #132]      ; e0 <resched+0xe0>
58:     e6bf6071    sxth    r6, r1
5c:     e0856206    add     r6, r5, r6, lsl #4
60:     e5966008    ldr     r6, [r6, #8]
64:     e5966000    ldr     r6, [r6]
68:     e0855206    add     r5, r5, r6, lsl #4
6c:     e5955004    ldr     r5, [r5, #4]
70:     e1520005    cmp     r2, r5
74:     c8bd8070    popgt {r4, r5, r6, pc}
                        return;
                }

                /* Old process will no longer remain current */

                ptold->prstate = PR_READY;
78:     e3a0e002    mov     lr, #2
7c:     e18ce0b3    strh    lr, [ip, r3]
```

```
                insert(currpid, readylist, ptold->prprio);
  80:   e6bf1071    sxth    r1, r1
  84:   ebfffffe    bl      0 <insert>
        }

        /* Force context switch to highest priority ready process */

        currpid = dequeue(readylist);
  88:   e59f304c    ldr     r3, [pc, #76]; dc <resched+0xdc>
  8c:   e1d300f0    ldrsh   r0, [r3]
  90:   ebffffe     bl      0 <dequeue>
  94:   e59f3038    ldr     r3, [pc, #56]; d4 <resched+0xd4>
  98:   e5830000    str     r0, [r3]
        ptnew = &proctab[currpid];
  9c:   e0600180    rsb     r0, r0, r0, lsl #3
  a0:   e1a00180    lsl     r0, r0, #3
  a4:   e59f302c    ldr     r3, [pc, #44]; d8 <resched+0xd8>
  a8:   e0801003    add     r1, r0, r3
        ptnew->prstate = PR_CURR;
  ac:   e3a02001    mov     r2, #1
  b0:   e18020b3    strh    r2, [r0, r3]
        preempt = QUANTUM;              /* Reset time slice for process    */
  b4:   e3a02002    mov     r2, #2
  b8:   e59f3024    ldr     r3, [pc, #36]; e4 <resched+0xe4>
  bc:   e5832000    str     r2, [r3]
#ifdef MMU
        FlushTLB();
        setPageTable();
#endif/*MMU*/

        ctxsw(&ptold->prstkptr, &ptnew->prstkptr);
  c0:   e2840004    add     r0, r4, #4
  c4:   e2811004    add     r1, r1, #4
  c8:   ebffffe     bl      0 <ctxsw>
  cc:   e8bd8070    pop     {r4, r5, r6, pc}
        ...

000000e8 <resched_cntl>:
 *------------------------------------------------------------------------
 */
status resched_cntl(                    /* Assumes interrupts are disabled      */
          int32defer            /* Either DEFER_START or DEFER_STOP         */
        )
{
  e8:   e92d4008    push    {r3, lr}
        switch (defer) {
  ec:   e3500001    cmp     r0, #1
  f0:   0a000002    beq     100 <resched_cntl+0x18>
  f4:   e3500002    cmp     r0, #2
  f8:   0a00000b    beq     12c <resched_cntl+0x44>
  fc:   ea000019    b       168 <resched_cntl+0x80>

            case DEFER_START:       /* Handle a deferral request */
```

```
            if (Defer.ndefers++ == 0) {
100:  e59f2088    ldr     r2, [pc, #136]       ; 190 <resched_cntl+0xa8>
104:  e5923000    ldr     r3, [r2]
108:  e2831001    add     r1, r3, #1
10c:  e5821000    str     r1, [r2]
110:  e3530000    cmp     r3, #0
114:  1a000015    bne     170 <resched_cntl+0x88>
                  Defer.attempt = FALSE;
118:  e3a02000    mov     r2, #0
11c:  e59f306c    ldr     r3, [pc, #108]       ; 190 <resched_cntl+0xa8>
120:  e5c32004    strb    r2, [r3, #4]
            }
            return OK;
124:  e3a00001    mov     r0, #1
128:  e8bd8008    pop     {r3, pc}

        case DEFER_STOP:      /* Handle end of deferral */
            if (Defer.ndefers <= 0) {
12c:  e59f305c    ldr     r3, [pc, #92]; 190 <resched_cntl+0xa8>
130:  e5933000    ldr     r3, [r3]
134:  e3530000    cmp     r3, #0
138:  da00000e    ble     178 <resched_cntl+0x90>
                  return SYSERR;
            }
            if ( (--Defer.ndefers == 0) && Defer.attempt ) {
13c:  e2433001    sub     r3, r3, #1
140:  e59f2048    ldr     r2, [pc, #72]; 190 <resched_cntl+0xa8>
144:  e5823000    str     r3, [r2]
148:  e3530000    cmp     r3, #0
14c:  1a00000b    bne     180 <resched_cntl+0x98>
150:  e5d23004    ldrb    r3, [r2, #4]
154:  e3530000    cmp     r3, #0
158:  0a00000a    beq     188 <resched_cntl+0xa0>
                  resched();
15c:  ebfffffe    bl      0 <resched>
            }
            return OK;
160:  e3a00001    mov     r0, #1
164:  e8bd8008    pop     {r3, pc}

        default:
            return SYSERR;
168:  e3e00000    mvn     r0, #0
16c:  e8bd8008    pop     {r3, pc}
        case DEFER_START:     /* Handle a deferral request */

            if (Defer.ndefers++ == 0) {
                  Defer.attempt = FALSE;
            }
            return OK;
170:  e3a00001    mov     r0, #1
174:  e8bd8008    pop     {r3, pc}
```

```
            case DEFER_STOP:      /* Handle end of deferral */
                if (Defer.ndefers <= 0) {
                        return SYSERR;
 178:  e3e00000    mvn   r0, #0
 17c:  e8bd8008    pop   {r3, pc}
                }
                if ( (--Defer.ndefers == 0) && Defer.attempt ) {
                        resched();
                }
                return OK;
 180:  e3a00001    mov   r0, #1
 184:  e8bd8008    pop   {r3, pc}
 188:  e3a00001    mov   r0, #1

            default:
                return SYSERR;
        }
}
 18c:  e8bd8008    pop   {r3, pc}
 190:  00000000    .word 0x00000000
```

Binary file for resched2.c

resched2.o: file format elf32-littlearm

Disassembly of section .text:

```
00000000 <reschedn>:
/*------------------------------------------------------------------------
 *  resched  -  Reschedule processor to highest priority eligible process
 *------------------------------------------------------------------------
 */
void   reschedn(pid32 pid,uint32 state)                 /* Assumes interrupts are disabled
        */
{
  0:   e92d4070    push  {r4, r5, r6, lr}
        struct procent *ptold;      /* Ptr to table entry for old process      */
        struct procent *ptnew;     /* Ptr to table entry for new process     */
        struct procent *ptnewp;


        ptnewp = &proctab[pid];
  4:   e0600180    rsb     r0, r0, r0, lsl #3
  8:   e1a00180    lsl     r0, r0, #3
        ptnewp->prstate = state;
  c:   e59f30cc     ldr     r3, [pc, #204]        ; e0 <reschedn+0xe0>
 10:   e18310b0    strh    r1, [r3, r0]
```

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

        if (Defer.ndefers > 0) {
14:     e59f30c8    ldr     r3, [pc, #200]      ; e4 <reschedn+0xe4>
18:     e5933000    ldr     r3, [r3]
1c:     e3530000    cmp     r3, #0
20:     da000003    ble     34 <reschedn+0x34>
                Defer.attempt = TRUE;
24:     e3a02001    mov     r2, #1
28:     e59f30b4    ldr     r3, [pc, #180]      ; e4 <reschedn+0xe4>
2c:     e5c32004    strb    r2, [r3, #4]
                return;
30:     e8bd8070    pop     {r4, r5, r6, pc}
        }

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

        ptold = &proctab[currpid];
34:     e59f30ac    ldr     r3, [pc, #172]      ; e8 <reschedn+0xe8>
38:     e5930000    ldr     r0, [r3]
3c:     e060c180    rsb     ip, r0, r0, lsl #3
40:     e1a0c18c    lsl     ip, ip, #3
44:     e59f3094    ldr     r3, [pc, #148]      ; e0 <reschedn+0xe0>
48:     e08c4003    add     r4, ip, r3

        if (ptold->prstate == PR_CURR) {  /* Process remains eligible */
4c:     e19c20b3    ldrh    r2, [ip, r3]
50:     e3520001    cmp     r2, #1
54:     1a00000f    bne     98 <reschedn+0x98>
                if (ptold->prprio > firstkey(readylist)) {
58:     e1d420f2    ldrsh   r2, [r4, #2]
5c:     e59f1088    ldr     r1, [pc, #136]      ; ec <reschedn+0xec>
60:     e1d110b0    ldrh    r1, [r1]
64:     e59f5084    ldr     r5, [pc, #132]      ; f0 <reschedn+0xf0>
68:     e6bf6071    sxth    r6, r1
6c:     e0856206    add     r6, r5, r6, lsl #4
70:     e5966008    ldr     r6, [r6, #8]
74:     e5966000    ldr     r6, [r6]
78:     e0855206    add     r5, r5, r6, lsl #4
7c:     e5955004    ldr     r5, [r5, #4]
80:     e1520005    cmp     r2, r5
84:     c8bd8070    popgt {r4, r5, r6, pc}
                        return;
                }

                /* Old process will no longer remain current */

                ptold->prstate = PR_READY;
88:     e3a0e002    mov     lr, #2
8c:     e18ce0b3    strh    lr, [ip, r3]
                insert(currpid, readylist, ptold->prprio);
90:     e6bf1071    sxth    r1, r1
94:     ebfffffe    bl      0 <insert>
```

```
        }

        /* Force context switch to highest priority ready process */

        currpid = dequeue(readylist);
  98:   e59f304c    ldr     r3, [pc, #76]; ec <reschedn+0xec>
  9c:   e1d300f0    ldrsh   r0, [r3]
  a0:   ebfffffe    bl      0 <dequeue>
  a4:   e59f303c    ldr     r3, [pc, #60]; e8 <reschedn+0xe8>
  a8:   e5830000    str     r0, [r3]
        ptnew = &proctab[currpid];
  ac:   e0600180    rsb     r0, r0, r0, lsl #3
  b0:   e1a00180    lsl     r0, r0, #3
  b4:   e59f3024    ldr     r3, [pc, #36]; e0 <reschedn+0xe0>
  b8:   e0801003    add     r1, r0, r3
        ptnew->prstate = PR_CURR;
  bc:   e3a02001    mov     r2, #1
  c0:   e18020b3    strh    r2, [r0, r3]
        preempt = QUANTUM;              /* Reset time slice for process    */
  c4:   e3a02002    mov     r2, #2
  c8:   e59f3024    ldr     r3, [pc, #36]; f4 <reschedn+0xf4>
  cc:   e5832000    str     r2, [r3]
#ifdef MMU
        FlushTLB();
        setPageTable();
#endif/*MMU*/

        ctxsw(&ptold->prstkptr, &ptnew->prstkptr);
  d0:   e2840004    add     r0, r4, #4
  d4:   e2811004    add     r1, r1, #4
  d8:   ebfffffe    bl      0 <ctxsw>
  dc:   e8bd8070    pop     {r4, r5, r6, pc}
        ...

000000f8 <resched_cntln>:
 *------------------------------------------------------------------------
 */
status resched_cntln(               /* Assumes interrupts are disabled      */
          int32defer            /* Either DEFER_START or DEFER_STOP         */
        )
{
  f8:   e92d4008    push    {r3, lr}
        switch (defer) {
  fc:   e3500001    cmp     r0, #1
 100:   0a000002    beq     110 <resched_cntln+0x18>
 104:   e3500002    cmp     r0, #2
 108:   0a00000b    beq     13c <resched_cntln+0x44>
 10c:   ea000019    b       178 <resched_cntln+0x80>

            case DEFER_START:       /* Handle a deferral request */

                if (Defer.ndefers++ == 0) {
 110:   e59f2088    ldr     r2, [pc, #136]        ; 1a0 <resched_cntln+0xa8>
```

```
114:  e5923000    ldr     r3, [r2]
118:  e2831001    add     r1, r3, #1
11c:  e5821000    str     r1, [r2]
120:  e3530000    cmp     r3, #0
124:  1a000015    bne     180 <resched_cntln+0x88>
                  Defer.attempt = FALSE;
128:  e3a02000    mov     r2, #0
12c:  e59f306c    ldr     r3, [pc, #108]        ; 1a0 <resched_cntln+0xa8>
130:  e5c32004    strb    r2, [r3, #4]
            }
            return OK;
134:  e3a00001    mov     r0, #1
138:  e8bd8008    pop     {r3, pc}

        case DEFER_STOP:       /* Handle end of deferral */
            if (Defer.ndefers <= 0) {
13c:  e59f305c    ldr     r3, [pc, #92]; 1a0 <resched_cntln+0xa8>
140:  e5933000    ldr     r3, [r3]
144:  e3530000    cmp     r3, #0
148:  da00000e    ble     188 <resched_cntln+0x90>
                  return SYSERR;
            }
            if ( (--Defer.ndefers == 0) && Defer.attempt ) {
14c:  e2433001    sub     r3, r3, #1
150:  e59f2048    ldr     r2, [pc, #72]; 1a0 <resched_cntln+0xa8>
154:  e5823000    str     r3, [r2]
158:  e3530000    cmp     r3, #0
15c:  1a00000b    bne     190 <resched_cntln+0x98>
160:  e5d23004    ldrb    r3, [r2, #4]
164:  e3530000    cmp     r3, #0
168:  0a00000a    beq     198 <resched_cntln+0xa0>
                  resched();
16c:  ebfffffe        bl      0 <resched>
            }
            return OK;
170:  e3a00001    mov     r0, #1
174:  e8bd8008    pop     {r3, pc}

        default:
            return SYSERR;
178:  e3e00000    mvn     r0, #0
17c:  e8bd8008    pop     {r3, pc}
        case DEFER_START:      /* Handle a deferral request */

            if (Defer.ndefers++ == 0) {
                    Defer.attempt = FALSE;
            }
            return OK;
180:  e3a00001    mov     r0, #1
184:  e8bd8008    pop     {r3, pc}

        case DEFER_STOP:       /* Handle end of deferral */
            if (Defer.ndefers <= 0) {
```

```
                return SYSERR;
 188:  e3e00000    mvn   r0, #0
 18c:  e8bd8008    pop   {r3, pc}
            }
            if ( (--Defer.ndefers == 0) && Defer.attempt ) {
                resched();
            }
            return OK;
 190:  e3a00001    mov   r0, #1
 194:  e8bd8008    pop   {r3, pc}
 198:  e3a00001    mov   r0, #1

         default:
            return SYSERR;
      }
}
 19c:  e8bd8008    pop   {r3, pc}
 1a0:  00000000    .word 0x00000000
```

From the above code , the changes made in resched2 resulted in the following changes in the binary file:-

```
e92d4070    push  {r4, r5, r6, lr}
      struct procent *ptold;      /* Ptr to table entry for old process      */
      struct procent *ptnew;      /* Ptr to table entry for new process      */
      struct procent *ptnewp;


      ptnewp = &proctab[pid];
  4:  e0600180    rsb     r0, r0, r0, lsl #3
  8:  e1a00180    lsl     r0, r0, #3
      ptnewp->prstate = state;
  c:  e59f30cc    ldr     r3, [pc, #204]       ; e0 <reschedn+0xe0>
 10:  e18310b0    strh    r1, [r3, r0]
```

There are total 93 number of instructions in resched.c while there are 97 number of instructions in resched2.c