# Lab 2

CIS-657 PRINCIPLES OF OPERATING SYSTEMS

PROF. MOHAMMED ABDALLAH

2/2/2021

Team 4

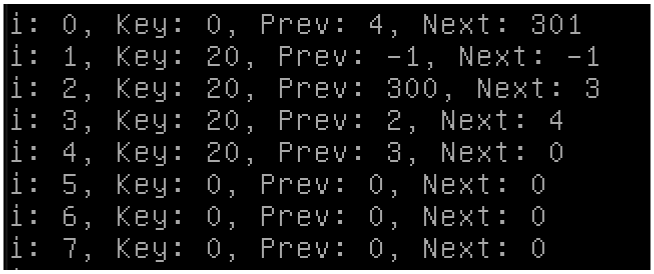
Anthony Redamonti
Adrian Bernat
Dana Dippery
Michael Rice
Ousmane Barry
SYRACUSE UNIVERSITY

## Part 1: The Queuetab

The following main.c program creates three processes, places them on the ready queue, and prints the contents of the queuetab.

```
/* main.c - main*/
#include <xinu.h>

void procA(void), procB(void), procC(void);

/* main - Example of creating processes in Xinu */
void main(void)
{
resume(create(procA, 1024, 20, "p 1", 0));
resume(create(procB, 1024, 20, "p 2", 0));
resume(create(procC, 1024, 20, "p 3", 0));
int i = 0;
while(i < NQENT){
    kprintf("i: %d, Key: %d, Prev: %d, Next: %d\n", i, queuetab[i].qkey, queuetab
[i].qprev, queuetab[i].qnext);
    i++;
    }
}

void procA(void)
{
    while(1);
}

void procB(void)
{
    while(1);
}

void procC(void)
{
    while(1);
}
```

The contents of each qentry are printed to the console shown below. The first column is the index into the queuetab array. The next column is the key followed by the previous and next pointers. The first element is the NULL process and has a key of zero. The key is zero because the priority of the NULL process is zero (lowest possible priority). The NULL process has its prev and next pointers assigned to 4 and 301 respectively.



```
i: 0, Key: 0, Prev: 4, Next: 301
i: 1, Key: 20, Prev: -1, Next: -1
i: 2, Key: 20, Prev: 300, Next: 3
i: 3, Key: 20, Prev: 2, Next: 4
i: 4, Key: 20, Prev: 3, Next: 0
i: 5, Key: 0, Prev: 0, Next: 0
i: 6, Key: 0, Prev: 0, Next: 0
i: 7, Key: 0, Prev: 0, Next: 0
```

The next entry in the array has a key value of 20 with prev and next pointers assigned to -1 (NULL) because this queue node represents the main process, which is the current process. Because it is the current process, it will no longer exist on the ready queue.

Indexes 2 through 4 refer to the processes created by the main function. The node order (by pointer) is:

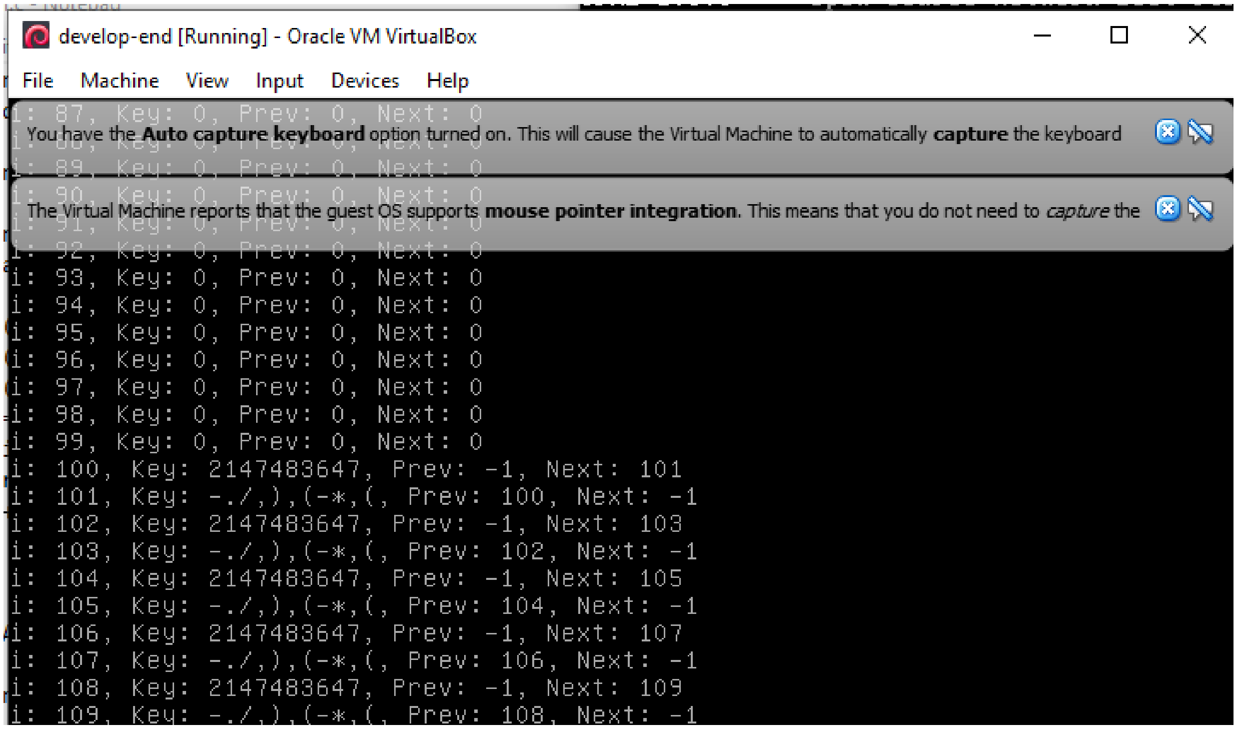
- 300 (head pointer)
- 2 (user process)
- 3 (user process)
- 4 (user process)
- 0 (null process)
- 301 (tail pointer)

The rest of the processes from 0 to NPROC-1 (NPROC equals 100 as per conf.h) have key, prev and next pointers equal to 0 because no further processes have been created.

Then entries from 100 to 299 refer to semaphores. By default (in conf.h), 100 semaphores are allowed in the Xinu system and there are two entries per semaphore. Finally, entries 300 through 303 contain the head and tail pointers for the process tables.

| Index | Key | Prev | Next |
|---|---|---|---|
| 0 | 0 | 4 | 301 |
| 1 | 20 | -1 | -1 |
| 2 | 20 | 300 | 3 |
| 3 | 20 | 2 | 4 |
| 4 | 20 | 3 | 0 |
| 5 | 0 | 0 | 0 |
| … | | | |
| 99 | 0 | 0 | 0 |
| … semaphores | | | |
| 300 | 2147483647 | -1 | 2 |
| 301 | -2147483648 | 0 | -1 |
| 302 | 2147483647 | -1 | 303 |
| 303 | -2147483648 | 302 | -1 |

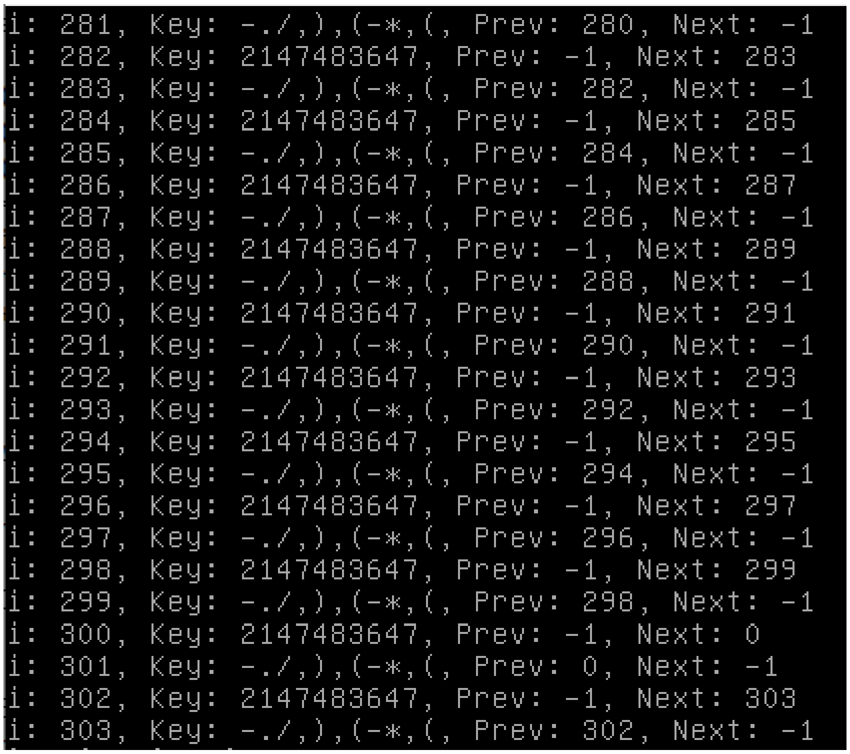
When the elements in the array are outside the range of NPROC-1, the qid portion of the queuetab has been entered. See the transition below.



You have the **Auto capture keyboard** option turned on. This will cause the Virtual Machine to automatically **capture** the keyboard

The Virtual Machine reports that the guest OS supports **mouse pointer integration**. This means that you do not need to *capture* the

```
i: 87, Key: 0, Prev: 0, Next: 0
i: 88, Key: 0, Prev: 0, Next: 0
i: 89, Key: 0, Prev: 0, Next: 0
i: 90, Key: 0, Prev: 0, Next: 0
i: 91, Key: 0, Prev: 0, Next: 0
i: 92, Key: 0, Prev: 0, Next: 0
i: 93, Key: 0, Prev: 0, Next: 0
i: 94, Key: 0, Prev: 0, Next: 0
i: 95, Key: 0, Prev: 0, Next: 0
i: 96, Key: 0, Prev: 0, Next: 0
i: 97, Key: 0, Prev: 0, Next: 0
i: 98, Key: 0, Prev: 0, Next: 0
i: 99, Key: 0, Prev: 0, Next: 0
i: 100, Key: 2147483647, Prev: -1, Next: 101
i: 101, Key: -./,),(-*,(, Prev: 100, Next: -1
i: 102, Key: 2147483647, Prev: -1, Next: 103
i: 103, Key: -./,),(-*,(, Prev: 102, Next: -1
i: 104, Key: 2147483647, Prev: -1, Next: 105
i: 105, Key: -./,),(-*,(, Prev: 104, Next: -1
i: 106, Key: 2147483647, Prev: -1, Next: 107
i: 107, Key: -./,),(-*,(, Prev: 106, Next: -1
i: 108, Key: 2147483647, Prev: -1, Next: 109
i: 109, Key: -./,),(-*,(, Prev: 108, Next: -1
```

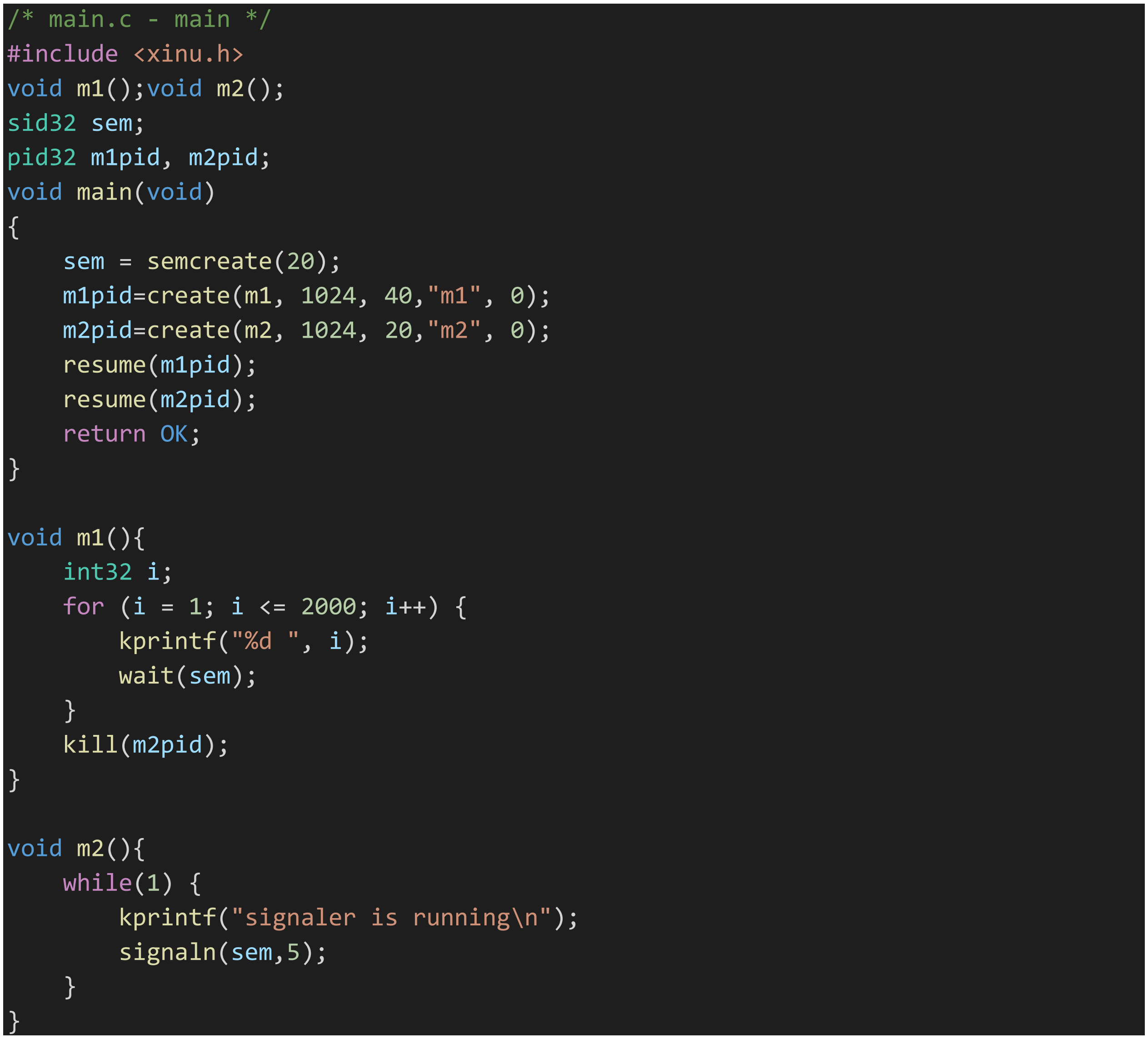


```
i: 281, Key: -./,),(-*,(, Prev: 280, Next: -1
i: 282, Key: 2147483647, Prev: -1, Next: 283
i: 283, Key: -./,),(-*,(, Prev: 282, Next: -1
i: 284, Key: 2147483647, Prev: -1, Next: 285
i: 285, Key: -./,),(-*,(, Prev: 284, Next: -1
i: 286, Key: 2147483647, Prev: -1, Next: 287
i: 287, Key: -./,),(-*,(, Prev: 286, Next: -1
i: 288, Key: 2147483647, Prev: -1, Next: 289
i: 289, Key: -./,),(-*,(, Prev: 288, Next: -1
i: 290, Key: 2147483647, Prev: -1, Next: 291
i: 291, Key: -./,),(-*,(, Prev: 290, Next: -1
i: 292, Key: 2147483647, Prev: -1, Next: 293
i: 293, Key: -./,),(-*,(, Prev: 292, Next: -1
i: 294, Key: 2147483647, Prev: -1, Next: 295
i: 295, Key: -./,),(-*,(, Prev: 294, Next: -1
i: 296, Key: 2147483647, Prev: -1, Next: 297
i: 297, Key: -./,),(-*,(, Prev: 296, Next: -1
i: 298, Key: 2147483647, Prev: -1, Next: 299
i: 299, Key: -./,),(-*,(, Prev: 298, Next: -1
i: 300, Key: 2147483647, Prev: -1, Next: 0
i: 301, Key: -./,),(-*,(, Prev: 0, Next: -1
i: 302, Key: 2147483647, Prev: -1, Next: 303
i: 303, Key: -./,),(-*,(, Prev: 302, Next: -1
```

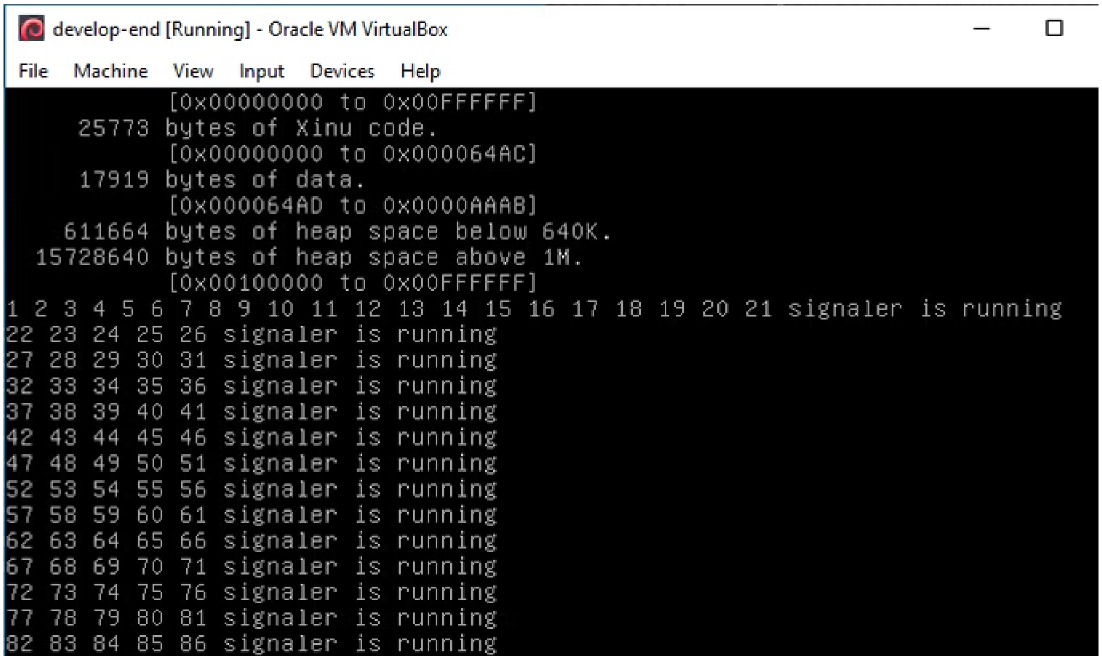
The key in the head of the qid is the MAXKEY. The MAXKEY is defined as the largest value a signed 32-bit integer can product (2,147,483,647). The element following the head is the tail which has a key of MINKEY. The MINKEY should have a value of -2,147,483,648 but instead has a value of “-./,),(-*,(,”. The prev of the head and next of the tail are -1 (NULL) as described in the textbook. The unused queues are empty; i.e. queuehead(q).next is pointing to queuetail(q), and queuetail(q).prev is pointing to queuehead(q). The second to last queue is the only queue being used as it contains one qentry located at index 0.

## Part 2: Using a Semaphore

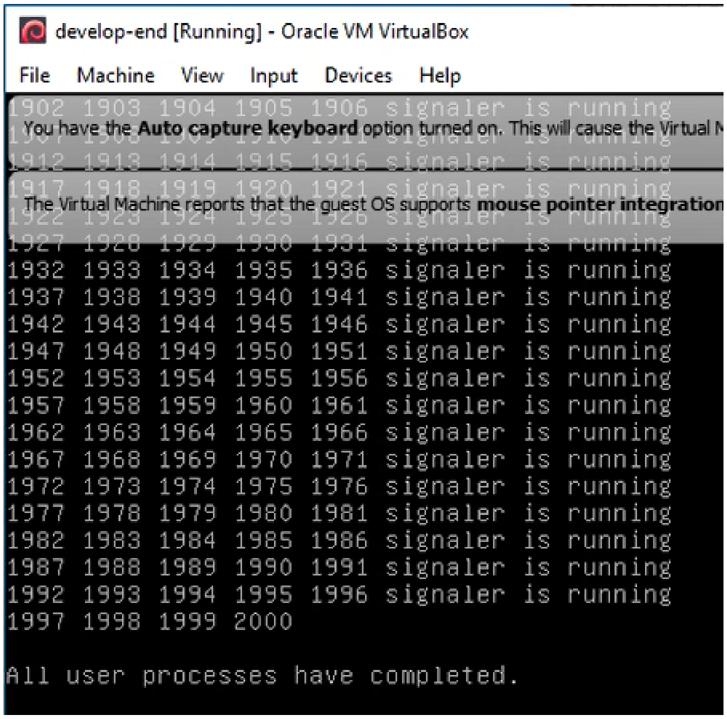
Below is the program written to create two processes “m1” and “m2”. Both processes share a semaphore “sem”. M1 will print out the first 21 numbers of integer ‘i’ then block on the semaphore. M2 will then print “signaler is running” on a new line. Then m2 will increment the semaphore 5 times, allowing m1 to print the next 5 numbers. To ensure that m1 runs first, it is given a priority of 40, and m2 is given a priority of 20. The pattern will continue until the m1 integer has incremented integer ‘i’ 2,000 times. Then m1 kills m2. Output of the code is on the next page.



```c
/* main.c - main */
#include <xinu.h>
void m1();void m2();
sid32 sem;
pid32 m1pid, m2pid;
void main(void)
{
    sem = semcreate(20);
    m1pid=create(m1, 1024, 40,"m1", 0);
    m2pid=create(m2, 1024, 20,"m2", 0);
    resume(m1pid);
    resume(m2pid);
    return OK;
}

void m1(){
    int32 i;
    for (i = 1; i <= 2000; i++) {
        kprintf("%d ", i);
        wait(sem);
    }
    kill(m2pid);
}

void m2(){
    while(1) {
        kprintf("signaler is running\n");
        signaln(sem,5);
    }
}
```



```
               [0x00000000 to 0x00FFFFFF]
       25773 bytes of Xinu code.
               [0x00000000 to 0x000064AC]
       17919 bytes of data.
               [0x000064AD to 0x0000AAAB]
      611664 bytes of heap space below 640K.
    15728640 bytes of heap space above 1M.
               [0x00100000 to 0x00FFFFFF]
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 signaler is running
22 23 24 25 26 signaler is running
27 28 29 30 31 signaler is running
32 33 34 35 36 signaler is running
37 38 39 40 41 signaler is running
42 43 44 45 46 signaler is running
47 48 49 50 51 signaler is running
52 53 54 55 56 signaler is running
57 58 59 60 61 signaler is running
62 63 64 65 66 signaler is running
67 68 69 70 71 signaler is running
72 73 74 75 76 signaler is running
77 78 79 80 81 signaler is running
82 83 84 85 86 signaler is running
```



You have the **Auto capture keyboard** option turned on. This will cause the Virtual N

The Virtual Machine reports that the guest OS supports **mouse pointer integration**

```
1902 1903 1904 1905 1906 signaler is running
1907 1908 1909 1910 1911 signaler is running
1912 1913 1914 1915 1916 signaler is running
1917 1918 1919 1920 1921 signaler is running
1922 1923 1924 1925 1926 signaler is running
1927 1928 1929 1930 1931 signaler is running
1932 1933 1934 1935 1936 signaler is running
1937 1938 1939 1940 1941 signaler is running
1942 1943 1944 1945 1946 signaler is running
1947 1948 1949 1950 1951 signaler is running
1952 1953 1954 1955 1956 signaler is running
1957 1958 1959 1960 1961 signaler is running
1962 1963 1964 1965 1966 signaler is running
1967 1968 1969 1970 1971 signaler is running
1972 1973 1974 1975 1976 signaler is running
1977 1978 1979 1980 1981 signaler is running
1982 1983 1984 1985 1986 signaler is running
1987 1988 1989 1990 1991 signaler is running
1992 1993 1994 1995 1996 signaler is running
1997 1998 1999 2000

All user processes have completed.
```