**Pınar Yücel**

**21802188**

# CS342 Operating Systems - Spring 2021

## Homework #4

---

1) Page size 256 Bytes = $2^8$ Bytes

   16 - 8 = 8

   **0010 101**1 1100 0101

   0010 1011 = 0x2B

   0x2B corresponds to 0xFC = 1111 1100

   Physical address is therefore 1111 1100 1100 0101 = 0xFCC5

   **0001 0011** 0001 1100

   0001 0011 = 0x13

   0x13 corresponds to 0xA3 = 1010 0011

   Physical address is therefore 1010 0011 0001 1100 = 0xA31C

   **1110 0100** 1010 0011

   1110 0100 = 0xE4

   0xE4 corresponds to 0xE5 = 1110 0101

   Physical address is therefore 1110 0101 1010 0011 = 0xE5A3

   **0010 1011** 0001 0111

   0010 1011 = 0x2B

   0x2B corresponds to 0xFC = 1111 1100

   Physical address is therefore 1111 1100 0001 0111 = 0xFC17

2) 9 9 9 9 12

   32 MB = $2^{25}$ Bytes

   Page size is $2^{12}$ Bytes

   $2^{25}/2^{12} = 2^{13}$ fourth level page table entries needed

   A fourth level page table can index at most $2^9$ entries

   $2^{13}/2^9 = 2^4$ fourth level page tables needed

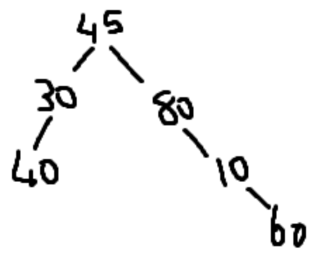
1 first, 1 second, 1 third, 16 fourth level page tables needed

$2^3*2^9 + 2^3*2^9 + 2^3*2^9 + 2^3*2^9*2^4 = 77824$ Bytes of memory is needed

3) ...40...30...

...60...10...80…

45 should be printed last



60 10 80 40 30 45

60 10 40 80 30 45

60 10 40 30 80 45

60 40 30 10 80 45

60 40 10 30 80 45

60 40 10 80 30 45

40 30 60 10 80 45

40 60 30 10 80 45

40 60 10 80 30 45

40 60 10 30 80 45

4) Semaphore empty = 1;

Semaphore smoker1 = 0;

Semaphore smoker2 = 0;

Semaphore smoker3 = 0;

<u>Non-smoking agent</u>

```
do{
        wait(empty);
        items = selectTwoRandomly(tobacco, paper, match);
        if (items are paper and match){
            signal(smoker1);
        }else if (items are tobacco and match){
            signal(smoker2);
        }else{
            signal(smoker3);
        }
    }while(true);
```

<u>Smoker 1</u>

```
do{
    wait(smoker1);
    signal(empty);
    smoke();
}while(true);
```

<u>Smoker 2</u>

```
do{
    wait(smoker2);
    signal(empty);
    smoke();
}while(true);
```

<u>Smoker 3</u>

```
do{
    wait(smoker3);
    signal(empty);
    smoke();
}while(true);
```

5) Optional Question

6) Page size is 64 bytes = $2^6$ bytes
   Segment size is 2 bits

Page size is 8 bits

a) 0x06B2
b) 0x12E4
c) 0x1348
d) Trap because offset is greater than the length.
e) 0x030A
f) 0x0B48

7) **Exists**

| A | B | C |
|---|---|---|
| 9 | 6 | 6 |

**Max Demand**

| | A | B | C |
|---|---|---|---|
| **P1** | 5 | 4 | 2 |
| **P2** | 3 | 1 | 3 |
| **P3** | 6 | 6 | 4 |
| **P4** | 2 | 2 | 1 |
| **P5** | 5 | 2 | 2 |

**Allocation**

| | A | B | C |
|---|---|---|---|
| **P1** | 1 | 2 | 1 |
| **P2** | 1 | 0 | 2 |
| **P3** | 2 | 2 | 2 |
| **P4** | 0 | 0 | 1 |
| **P5** | 3 | 1 | 0 |

**Available**

| A | B | C |
|---|---|---|
| 2 | 1 | 0 |

**Need (Max Demand - Allocation)**

|  | A | B | C |
|---|---|---|---|
| **P1** | 4 | 2 | 1 |
| **P2** | 2 | 1 | 1 |
| **P3** | 4 | 4 | 2 |
| **P4** | 2 | 2 | 0 |
| **P5** | 2 | 1 | 2 |

With the current available resources, none of the possible needs can be satisfied. The system is in an unsafe state.

8) **Exists**

| A | B | C |
|---|---|---|
| 10 | 6 | 4 |

**Available**

| A | B | C |
|---|---|---|
| 3 | 3 | 0 |

P4 can be satisfied. After P4:

**Available**

| A | B | C |
|---|---|---|
| 4 | 3 | 1 |

P1 can be satisfied. After P1:

**Available**

| **A** | **B** | **C** |
|---|---|---|
| 5 | 3 | 1 |

None of the remaining processes can be satisfied with the current available resources. There is a deadlock.

9) monitor Sem{

```
struct process{
        pid_t pid;
        struct process* next;
}

struct Semaphore{
        int value = 0;
        struct process* queue;
}

condition c;
pid_t pid;

void wait(Semaphor& S){
        S->value--;
        if(S->value < 0){
                struct process* newProcess = (struct process*)
malloc(sizeof(struct process));
                newProcess->pid = getpid();
                newProcess->next = NULL;
                struct process* head = S->queue;

                while(head->next != NULL){
```

```
                head = head->next;
            }

            head->next  = newProcess;
            while(getpid() != pid){
                condition.wait();
            }
        }
}

void signal(Semaphor& S){
        S->value++;
        if(S->value <= 0){
            struct process* oldHead = S->queue;
            S->queue = S->queue->next;
            pid = oldHead->pid;
            free(oldHead);
            condition.broadcast();
        }
}

}
```