

CMPT 300 - Assignment 3

Q1.

Known:

Virtual memory = 2^{12} pages

Physical memory - 16 page frames. 2^{13} bits per page = 2^{10} byte per page.

Lower level page table - 8 bytes per entry

Upper level page table - 8 bytes per entry

a) Assume that each byte of memory in the virtual memory has an address. How many addresses would be needed to address all bytes in one page? Why? How many bits would be needed in an address that provided a unique address for each byte in a page?

Page size = page frame size = 2^{10} byte

Virtual memory = 2^{13} bits per page * 2^{12} = 2^{25} bits = 2^{22} bytes = 2^{22} addresses total in virtual memory.

We need 22 bits to represent each of the addresses in virtual memory in one page.

b) How many page table entries fit in one page of virtual memory? Why? What is the minimum number of bits that would be needed in an address that provided a unique address for every page table entry in a particular page of page table entries?

Number of page table entries per page = page size / size of each page entry
= 2^{10} bytes / 2^3 bytes
= 2^7 entries

Since each lower level table contains 2^7 entries, the minimum # of bits needed in each of these unique addresses would be 7 bits.

c) Each of the lower level page tables contains 2^7 entries. Since there are total 2^{12} entries in our virtual memory, we must have 2^5 of these lower level page tables. Thus, there must be $2^5 = 32$ pages in the upper level page table indexing into each of these lower level page table with 2^7 entries. 5 bits are required to address the pages of lower level page tables.

d) So far, we have $N=22$ bits, K (upper table bits) = 5, and M (lower table bits) = 7.

K 00000 = 5 bits	M = 7 bits	N-K-M =10 bits
------------------	------------	----------------

Virtual Address

(1) K= 10100	(3) M= 1101011	Offset = 101 001 011 1
--------------	----------------	------------------------

Upper level page table

Address	Entry
00000	0 xxxxxxxxxxx
00001	0 xxxxxxxxxxx
00010	0 xxxxxxxxxxx
00011	0 xxxxxxxxxxx
....	
10100	(2) 1 xxxxxxxxxxx
10101	0 xxxxxxxxxxx
10110	0 xxxxxxxxxxx

Physical address

Page frame #	1010 101 001 011 1
--------------	--------------------

Lower level page table

Address	Entry
000 0000	0000 0 xxxxxxxxxxx (StartAddr)
000 0001	1001 0 xxxxxxxxxxx
000 0010	0101 0 xxxxxxxxxxx
000 0011	1000 0 xxxxxxxxxxx
...	...
110 1011	(3) 1010 1 1010010111
110 1100	0001 0 xxxxxxxxxxx
110 1101	1111 0 xxxxxxxxxxx

In the lower level page table, the first 4 bits represent the page frame #, the next single bit represents the flag indicating whether our address is in the physical memory or not, and the last 10 bits represent the offset. The page frame number for our address was arbitrarily chosen as 1010. (5) Prepending the offset to the this page frame number produces a physical address of 1010 1010010111.

e) Steps to translating from the logical to physical address:

1. Use the first K=5 bits to index into the upper level page table. In this case, the K bits were 10100, which is equivalent to page 20 in the upper level page table.
2. Check if the flag = 1 to confirm that the page is in memory.
3. Use the next M=7 bits to index into the lower level page table. In this case, the M bits were 1101011. This was equivalent to entry #107 in the lower level page table on page 20 of the upper level page table.
4. Check if the flag = 1 to confirm that our address is in memory.
5. Prepend the offset to the page frame number of our entry. The resulting product is the physical address of the given vertical address.

Q2.

/* shared variables and semaphores */

int adult_onBoard =0;

int child_onBoard = 0;

int adult_CAPACITY =2;

int child_CAPACITY=4;

SEMAPHORE adult_protect=1; //mutex

SEMAPHORE kid_protect=1; //mutex

SEMAPHORE adult_boarding=2;

SEMAPHORE kid_boarding=4;

SEMAPHORE adult_full=0;

SEMAPHORE child_full=0;

SEMAPHORE wholsLoaded=0;

SEMAPHORE loadingComplete=0;

Adult code:

```
while(true) {  
    wait(adult_boarding);  
    wait(adult_protect);  
    adult_onBoard++;  
    //load  
    load();  
}
```

```

    signal(whosLoaded);
    if (adult_onBoard == adult_CAPACITY && child_onBoard == child_CAPACITY)
        signal(loadingComplete);
    else if (adult_onBoard == adult_CAPACITY)
        signal(adult_full);
    else
        signal(adult_boarding);
    signal(adult_protect);
}

```

Child code:

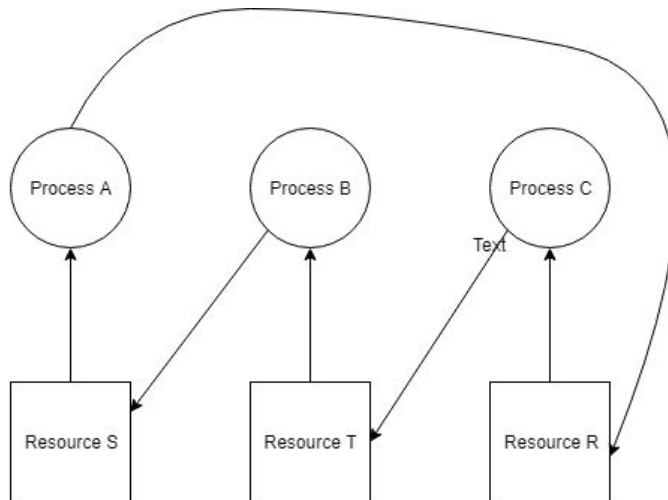
```

while(true) {
    wait(child_boarding);
    wait(child_protect);
    child_onBoard++;
    //load
    load();
    signal(whosLoaded);
    if (adult_onBoard == adult_CAPACITY && child_onBoard == child_CAPACITY)
        signal(loadingComplete);

    else if (child_onBoard == child_CAPACITY)
        signal(child_full);
    else
        signal(child_boarding);
    signal(child_protect);
}

```

Q3a.



Process A request resource S

Process A holds resource S

Process B request resource T

Process B holds resource T

Process C request resource R

Process C holds resource R

Process A request resource R, but resource R is held by Process C -> blocked

Process B request resource S, but resource S is held by Process A -> blocked

Process C request resource T, but resource T is held by Process B -> blocked

The three processes are now deadlocked.

Q3b.

A: available vector: total amount of each type of resource

E: resource vector: amount of each resource currently available

R: claim matrix: resource needed by process

C: allocation matrix: resource possessed by process

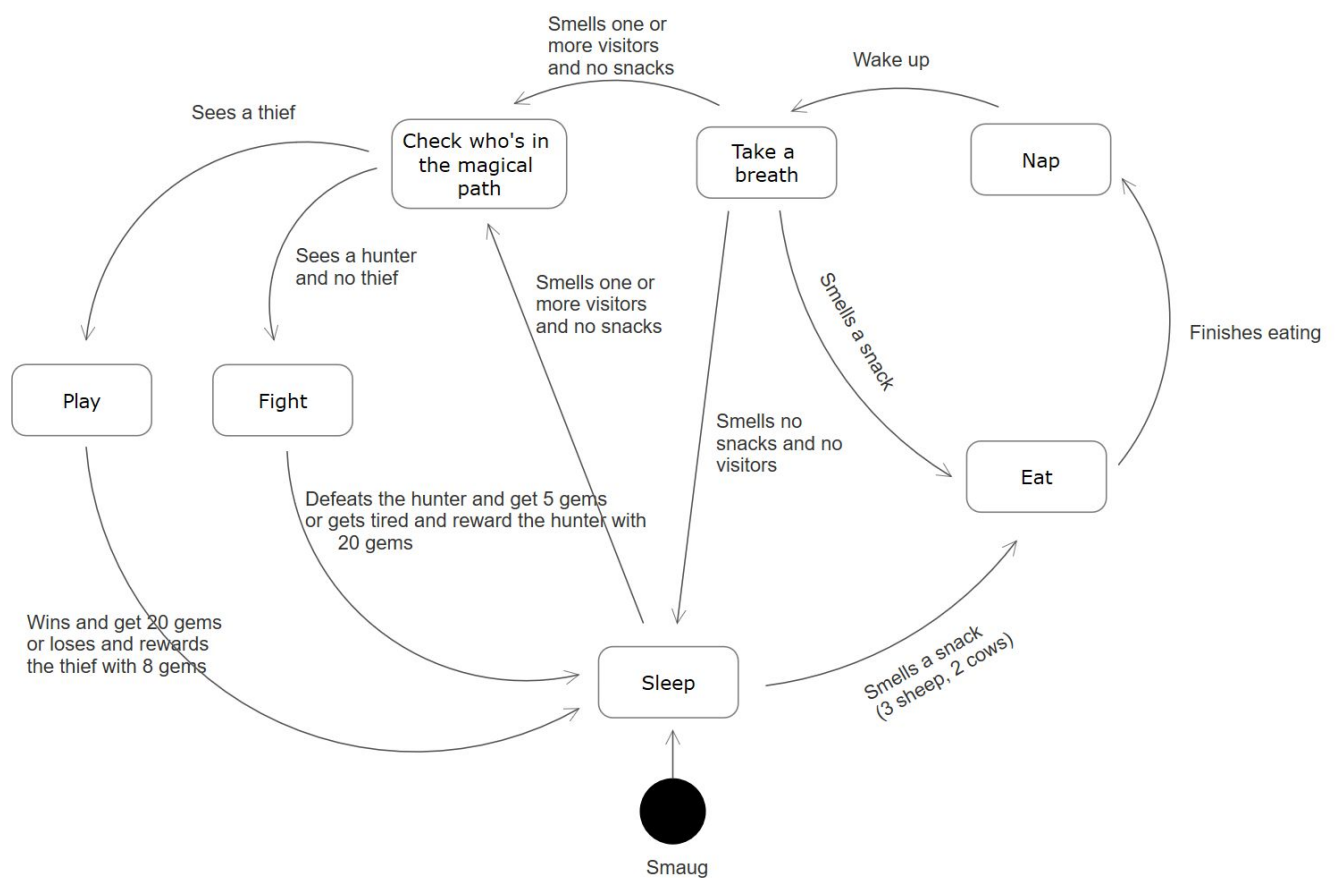
$$\begin{bmatrix} 1 & 3 & 0 & 1 & 2 \\ 0 & 0 & 3 & 3 & 3 \\ 2 & 4 & 0 & 3 & 2 \end{bmatrix} = C$$

$$\begin{bmatrix} 2 & 5 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 \\ 2 & 2 & 1 & 3 & 2 \end{bmatrix} = R - C$$

$|2\ 5\ 0\ 0\ 0| > |1\ 3\ 2\ 0\ 0|$ A cannot run

$|1\ 1\ 2\ 0\ 0| < |1\ 3\ 2\ 0\ 0|$ A runs to completion
 New A = $|1\ 3\ 2\ 0\ 0| + |0\ 0\ 3\ 3\ 3| = |1\ 3\ 5\ 3\ 3|$
 $|2\ 5\ 0\ 0\ 0| > |1\ 3\ 5\ 3\ 3|$ A cannot run
 $|2\ 2\ 1\ 3\ 2| > |1\ 3\ 5\ 3\ 3|$ A cannot run
 Processes 1st and 3rd are deadlocked.
 The state is unsafe.

Q4.



Sheep or cow

