CMPT 300 - Assignment 3

Q1. A particular computer system has a virtual memory with 4096 pages. The physical memory of this computer system has 16 page frames. The size of each page frame is 8Kb (1Kb=2^10 bits). Each lower level page table entry uses 8 bytes. Each upper level entry uses 8 bytes.
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 VM.
We need 22 bits to represent each of the addresses in virtual memory.

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

Minimum number of bits needed in an address that provided a unique address for every page table entry (every address in the lower table) in a particular page of page table 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) What is the minimum number of pages of lower level page table needed to hold all the page table entries? Why? What is the minimum number of bits we could use to address the pages of lower level page tables?

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

Virtual Address

(1) K= 10100	(3) M= 1101011	page offset = 101 001 011 1

Upper level page table

Address	Entry
00000	0(flag) xxxxxxxxxx…. (total 64 bits) Each upper level page table entry uses 8 bytes = 64-bits
00001	0 xxxxxxxxxx…. (total 64 bits)
00010	0 xxxxxxxxxx…. (total 64 bits)
00011	0 xxxxxxxxxx…. (total 64 bits)
....	
10100	(2) 1 xxxxxxxxxx…. (total 64 bits)
10101	0 xxxxxxxxxx…. (total 64 bits)
10110	0 xxxxxxxxxx…. (total 64 bits)

Physical address

Page frame number= 1010	page offset = 101 001 011 1

Lower level page table

Address	Entry
000 0000	0000 0 xxxxxxxxxx…. (total 64 bits)
000 0001	1001 0 xxxxxxxxxx…. (total 64 bits)
000 0010	0101 0 xxxxxxxxxx….(total 64 bits)
000 0011	1000 0 xxxxxxxxxx….(total 64 bits)
...	
110 1011	(3) 1010 1 xxxxxxxx...(total 64 bits)
110 1100	0001 0 xxxxxxxxxx….(total 64 bits)
110 1101	1111 0 xxxxxxxxxx ….(total 64 bits)

In the lower level page table, each entry consists of a 4-bit page frame number, one-bit flag indicating whether our address is in the physical memory or not, and the remaining 49 bits consisting of 0's and 1's.

Offset is not stored in the page table. The page frame number for our address was arbitrarily chosen as 1010. (5) Adding the offset to 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. Adding 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 whoIsLoaded=0;
SEMAPHORE loadingComplete=0;
```

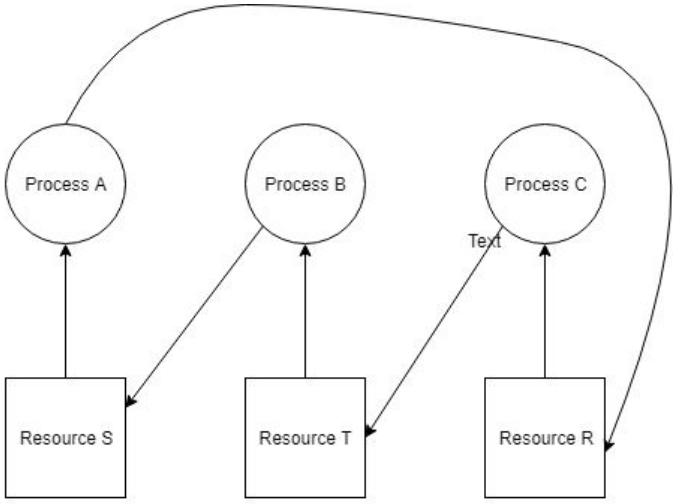
Adult code:

```
while(true) {
        wait(adult_boarding);
        wait(adult_protect);
        adult_onBoard++;
         //load
        load();
        signal(whoIsLoaded);
        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(whoIsLoaded);
         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.



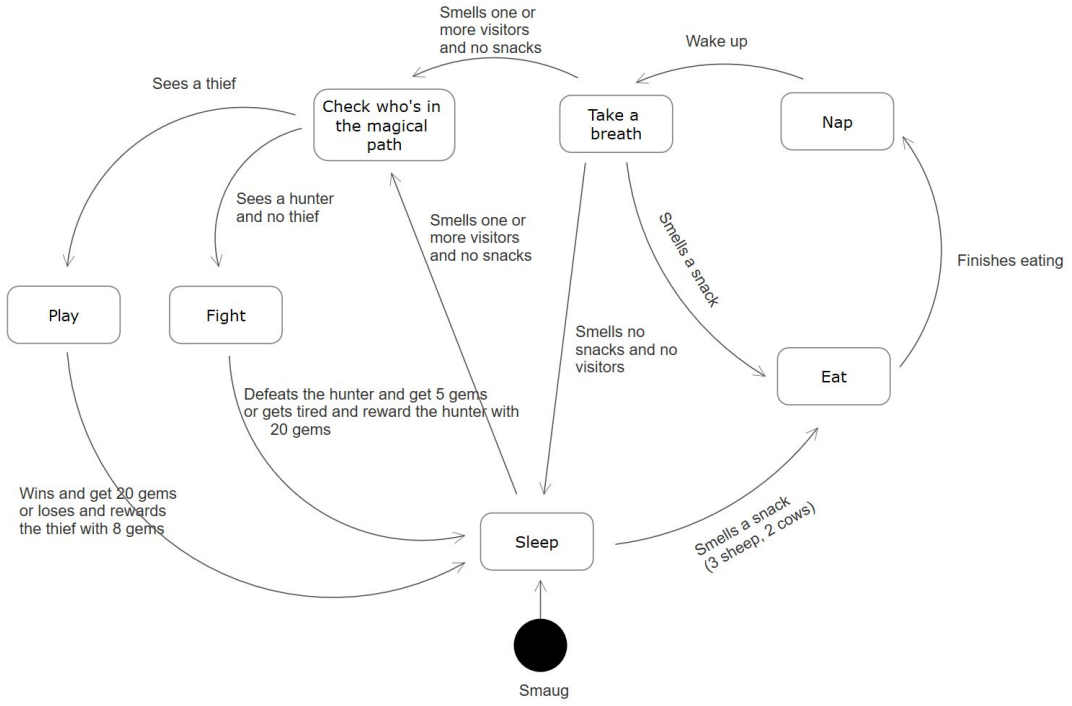
Process A request resource S
Process A holds resource S
Process B request resource T
Process B holds resource T
Process C request resource R
Process C holds resource R
Process A request resource R, but resource R is held by Process C -> blocked
Process B request resource S, but resource S is held by Process A -> blocked
Process C request resource T, but resource T is held by Process B -> blocked
The three processes are now deadlocked.

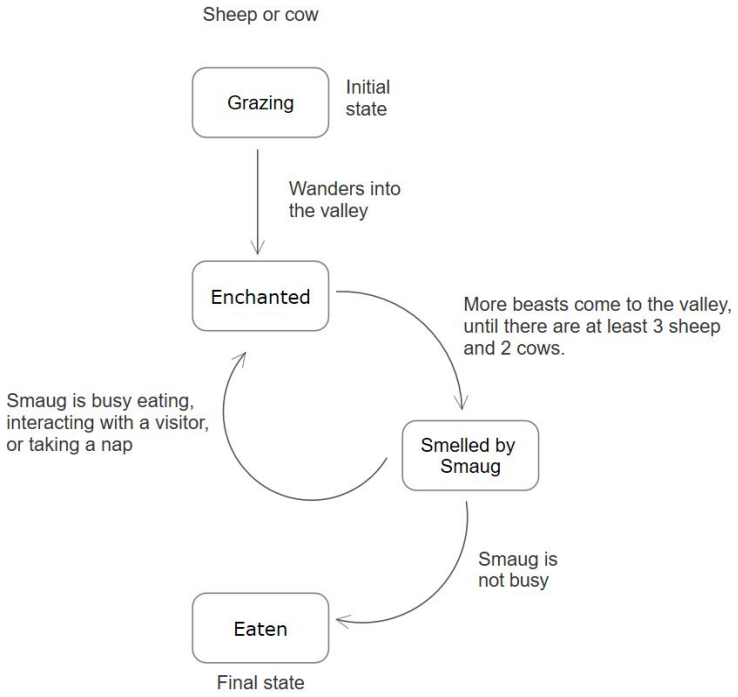
Q3b.
A: available vector: total amount of each type of resource
E: resource vector: amount of each resource currently available
R: claim matrix: resource needed by process
C: allocation matrix: resource possessed by process
| 1 3 0 1 2 |
| 0 0 3 3 3 | = C
| 2 4 0 3 2 |

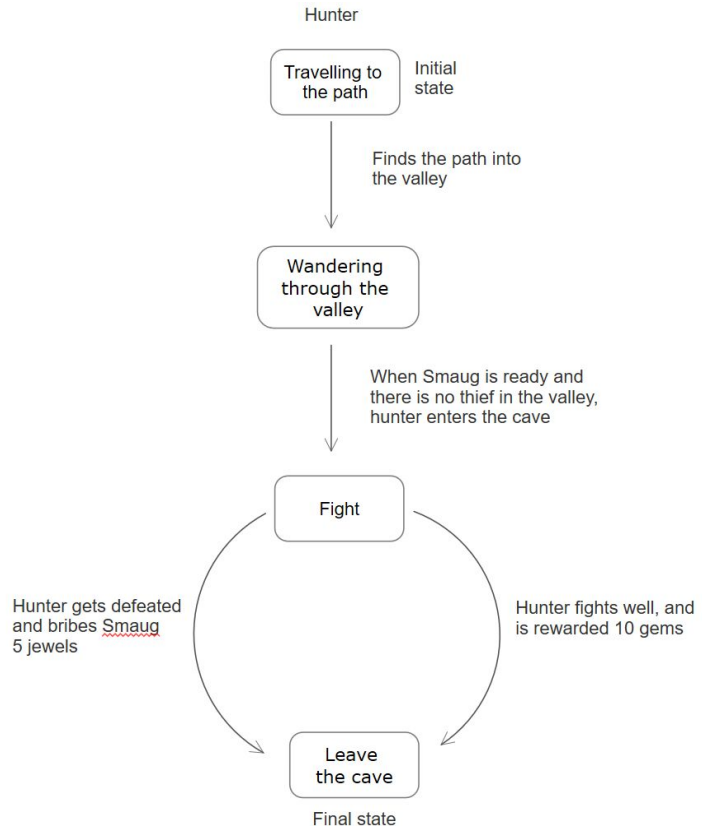
| 2 5 0 0 0 |
| 1 1 2 0 0 | = R - C
| 2 2 1 3 2 |

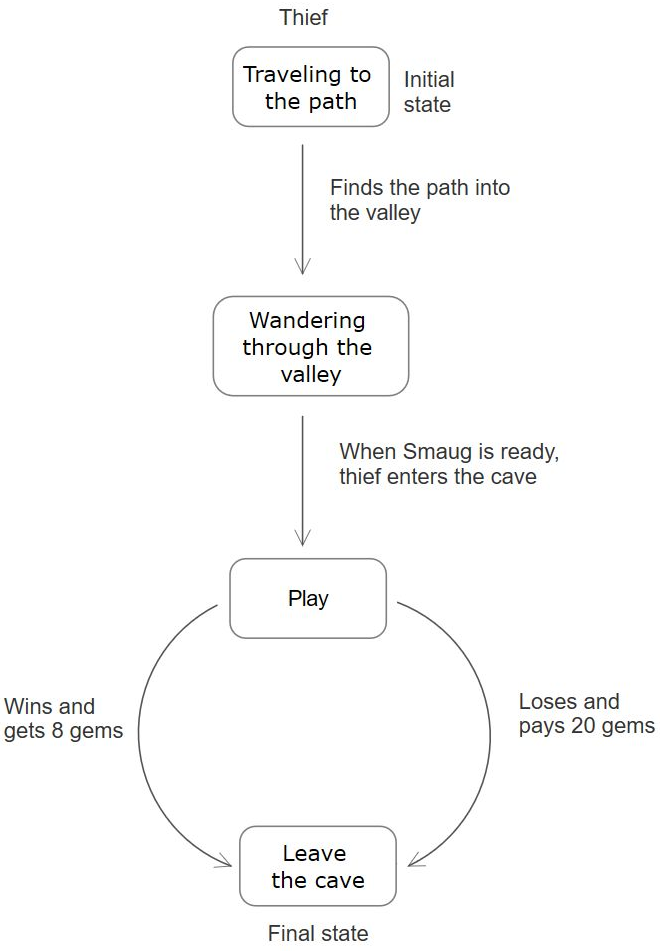
| 2 5 0 0 0 | > | 1 3 2 0 0 | A cannot run
| 1 1 2 0 0 | < | 1 3 2 0 0 | A runs to completion
New A = | 1 3 2 0 0 | + | 0 0 3 3 3 | = | 1 3 5 3 3 |
| 2 5 0 0 0 | > | 1 3 5 3 3 | A cannot run
| 2 2 1 3 2 | > | 1 3 5 3 3 | A cannot run
Processes 1st and 3rd are deadlocked.
The state is unsafe.

Q4.




Sheep or cow
Grazing
Initial state
Wanders into the valley
Enchanted
More beasts come to the valley, until there are at least 3 sheep and 2 cows.
Smaug is busy eating, interacting with a visitor, or taking a nap
Smelled by Smaug
Smaug is not busy
Eaten
Final state


Hunter
Travelling to the path
Initial state
Finds the path into the valley
Wandering through the valley
When Smaug is ready and there is no thief in the valley, hunter enters the cave
Fight
Hunter gets defeated and bribes Smaug 5 jewels
Hunter fights well, and is rewarded 10 gems
Leave the cave
Final state


Thief
Traveling to the path
Initial state
Finds the path into the valley
Wandering through the valley
When Smaug is ready, thief enters the cave
Play
Wins and gets 8 gems
Loses and pays 20 gems
Leave the cave
Final state