Question 1 – Monitors (33 pts)

The stick figure dormitory has an entryway with a threshold and a door; stick figures walk at constant speed and it always takes 5 seconds after crossing the threshold to reach the door. (a)

- If no one else in the entryway (between the threshold and the door) when a figure gets to the door, it does the logical thing – opens the door, goes through, and closes it. (b)

- If a stick figure has the “door-holding” trait, then if it arrives at a closed door and there are other figures in the entryway, it will open the door and hold it while the figures behind it go through (c), until there are not others in the entryway. It will then go through and close the door. (d)

- If a stick figure of either type (holder or non-holder) gets to the door and it is being held open, it goes through.

Write pseudocode for a simulation of this system. The monitor will have two methods: door-holder() and non-holder(). A thread entering one of these two methods corresponds to a stick figure crossing the threshold, and the method returns after the figure passes through the door and possibly closes it.

Besides normal monitor, integer, and boolean operations you can use the following statements: “delay 5”, “open door”, “go through”, and “close door”.

Space for your answer is provided on the next page.
Question 1 (space for answer)
Question 2 - Context Switching (33 pts)

Here we see the data structures and code for a simplified version of question 4 from homework 1. Code is given in a mix of C and assembly language.

```c
q4() {
    ...
    1   do_switch(&tmp, sp[0])
}
```

```assembly
do_switch(old_ptr, new_val) {
    1   mov *(sp+4), eax // *old_ptr
    2   mov sp, *eax     //        = sp
    3   mov *(sp+8), sp  // sp = new_val
    4   ret
}
```

```c
proc1() {
    1   push &buf_1
    2   call getline  // getline(&buf_1)
    3   ... print line
    goto line 1
}
```

```assembly
proc2() {
    1   push &buf_2
    2   call getline // getline(&buf_2)
    3   ... print line
    goto line 1
}
```

```c
getline(char *buf_arg) {
    1   i = get_term_in(buf_g)
    2   if (i != curr) {
        3       curr = i
        [push arguments]
        4       do_switch(&sp[1-i], sp[i])
        [pop & discard arguments]
    } 
    5   strcpy(buf_arg, buf_g)
    6   return
}
```

Process 1 uses addresses 0x1000-0x1fff, with execution starting from the bottom and stack at the top; process 2 uses 0x2000-2fff and global variables start at 0x3000. Assume that line 1 of proc1() is at address x1000, and line 1 of proc2 is at x2000.

If execution starts at q4, and we receive two lines of input for process 2 (and none for process 1), execution will proceed through a particular series of steps until it finally pauses in get_term_input() waiting for a third line of input. Give that sequence of steps.

More specifically, give the sequence in which lines above are executed, starting with q4.1. If the stack pointer is changed during a step, describe the change and indicate what values are pushed or popped. (numeric values aren't necessary, but be specific) If any of the global variables (sp[0,1], curr, buf_1, buf_2, buf_g) are modified in a step, say so and describe. Note that you can consider get_term_input() and strcpy() to be atomic – i.e. they don't modify the stack pointer, and only affect their inputs and outputs.
Question 2 (space for answer)
Question 3 – Markov Models (33 pts)

Here we analyze a system where a user sits at a terminal, thinks for a period of time, and then makes a request of either type A or type B. Each request takes a certain amount of processing time to bring up a display screen, and then possibly issues a database query. After this the user’s request is complete, and the user is idle for another “think time”.

All times are exponentially distributed, with the following parameters:

- average think time: \( \frac{1}{4} \) minute
- request distribution: \( \frac{1}{4} \) A, \( \frac{3}{4} \) B
- request A processing time: \( \frac{1}{4} \) minute
- request B processing time: \( \frac{1}{8} \) minute
- query processing time: \( \frac{1}{6} \) minute
- fraction of B's resulting in query: \( \frac{3}{8} \)

(a) Label the arcs in the diagram above with the transition rates in transitions/minute

(b) Give the balance equations and solve for the fraction of time the system is in states I, A, B, and Q.
Question 3 (space for work)
**Question 4 – Virtual Memory (33 pts)**

The diagram below shows the page tables in physical memory for two address spaces, one (address space 1) rooted at the page directory in physical page 00000 (to the left below) while the other has a page directory at 00004. The two address spaces share a single physical page (page 00002) mapped read-only at 0x06000000.

The operating system view of address space 2 is shown to the right - note that the shared page is mapped read-only so that it can be copied on write, while 05000000 is demand-paged from disk and BFFF0000 is allocated on demand.

We begin execution in the following state:

- **PC** = 05000000
- **SP** = BFFF0FFC

The code to be paged in contains starts with the following 3 instructions:

```
MOV *(06000100), EAX
PUSH EAX
MOV EAX, *(06000300)
```

Give the sequence of events which occur during the execution of these three instructions. Specifically describe each of the following events when they occur:

- instruction attempts – indicate the instruction
- instruction completions
- page faults – indicate which instruction faulted (1st MOV, PUSH, or 2nd MOV), whether an instruction or data fault, and the fault address if it was a data fault.
- allocation of free physical pages (any order you wish) – indicate page number.
- updates to the page directory or page tables – indicate the index of the entry, which fields are written (page #, read/write, valid) and what values are written.
Question 4 (space for answer)
Please show your work – arithmetic errors are easy to make with hexadecimal, but will receive little or no penalty if I can determine that you were solving the problem correctly.
Helpful definitions

Monitor: A monitor is a particular kind of user-defined class, which has:
- regular instance variables
- methods
- conditions --- this is what normal classes don't have

Threads can:
- enter the monitor by calling any of the methods
- leave the monitor by returning from a method
- leave the monitor by calling wait()
- re-enter the monitor by returning from wait()

If there are any threads waiting on condition C:
- signal(C) will release one of them and
- broadcast(C) will release all of them.
- If there are no threads waiting on C, neither signal(C) nor broadcast(C) will have any effect.

Only one thread can be in the monitor (i.e. entered but not left) at any given time. If thread A is in the monitor and thread B tries to (a) call a method or (b) return from wait(), thread B will block until thread A leaves the monitor. (or longer – see below)

When a thread calls signal() or broadcast() it does not leave the monitor, so that you can be sure that it will continue to run before any thread that it woke up, or before any other new thread which tries to call a monitor method.

However, if you wake thread A via signal() or broadcast(), and thread B tries to enter the monitor at the same time by calling a method, you don't know whether A or B will get to run first.

Hexadecimal numbers: Hex numbers are base 16, with numerals 0-9, A, B, C, D, E and F. The following is a table converting between decimal, hexadecimal, and binary:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
<td>8</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
<td>9</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
<td>10</td>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
<td>11</td>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
<td>12</td>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
<td>13</td>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
<td>14</td>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
<td>15</td>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>