1. Consider the following different generations/types of operating systems.
   a. Multi-programming time sharing systems.
   b. Multi-programming batch systems.
   c. Pure batch systems.
   d. Personal computers that mainly focus on single users.
   e. An embedded computer that is used on your car to regulate the brake system. (It takes speed and brake pedal pressing level as input and brake pressure as output. The system has a simple CPU and some memory, but no disk.)
   f. A real time system that handles urgent events with different priorities. When a low priority task is in execution and some high priority tasks arrive, the system should switch to execute the one with the highest priority.

Which ones require an interrupt vector? For those that do require an interrupt vector, discuss what types of interrupts are required. For those that do not need an interrupt vector, discuss why not.

2. Consider two functions P and Q, as shown in the following code. Note that each statement is labeled and the label is just for convenience purpose (no impact to execution).

```
int a, b = 0;

function P() begin
  A. a = 8;
  B. a = a + 5;
  C. b = a * 2;
end;

function Q() begin
  D. b = 4;
  E. a = b + 3;
end;

main() …
```

(a) The main program is as follows. List all possible values of the pair (a, b) after the execution of the program.

```
main()
{ int pid;
  a = 1; pid = fork();
  if (pid == 0) { a = 2; P(); } else Q();
}
```

(b) The pseudo code for the main program above is given in the following left. Assume that the program is compiled into executable “a.out”. And, “a.out” is invoked by the program at the following right. List all possible values of the pair (a, b) after the execution of the left program.

```
main()
{ int pid;
  pid = fork();
  if (pid == 0) { a = 2; exec (“a.out”); } else Q();
}
```

main()
{ b = 3;
  create-thread (execute function P);
  create-thread (execute function Q);
}
3. Consider the following pseudo assembly code for computing \( c = a + b \). Assume that \( a, b, \) and \( c \) are assigned to consecutive memory “words” (memory is generally addressed byte by byte and assume that a word is 4 bytes) and address for “\( a \)” is 0x0000e308. Also, we have \( a = 20, b = 136, \) and \( c = 0 \) at the starting time. Assume that the first instruction of the code is stored in 0x00001018. Also, each instruction has the opcode in the first byte (most significant byte) and the remaining 3 bytes specify the corresponding address. The opcode for load is 1, store is 2, and add is 3.

- load a
- add b
- store c

(a) Show the memory addresses and contents for all the instructions and data involved. Use the format as follows to express your answer (but the following is not the answer). For all data, use hexadecimal representation.

<table>
<thead>
<tr>
<th>addresses</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00002104</td>
<td>0x00000001</td>
</tr>
<tr>
<td>0x00002108</td>
<td>0x00000002</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(b) Write the micro instructions for the code segment. Assume that current PC (program counter) is 0x00001018. For each micro-instruction, also indicate the data that is transferred (if it is a memory access). For all data, use the hexdecimal representation. The following are the first two micro-instructions and the data transferred. Complete the rest.

<table>
<thead>
<tr>
<th>Micro-instructions</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC → MAR</td>
<td>0x00001018</td>
</tr>
<tr>
<td>M → MBR</td>
<td>0x0100e308</td>
</tr>
</tbody>
</table>

4. Consider the following two high level programs that are executed concurrently. Some system level events occurred during the execution of each. Follow the example for process state transitions, add the events happened during execution (in addition to the ones that are given). For each event (including the ones you identified and the ones given), elaborate on what is happening in the system.

Program P;
{     read (x);
    compute factorial of x;
    print (x);
    pid = fork ();
    if (pid == 0)
    {     read (y);
        compute factorial of y;
        print (y);
    }
    else   // parent process
    wait (pid);  // while waiting, child process’s “read (y)” is done
    // while waiting, child process’s “print (y)” is done
}

Program Q;
{     initialize matrix A using random number generator;
    -- I/O interrupt: P’s “read (x)” operation is done
    initialize matrix B using random number generator;
    -- I/O interrupt: P’s “print (x)” operation is done
    create a thread to compute A*B;
}