Section 1: MCQ (2 marks each)

MCQ 1 and 2 based on the following code fragment:

```
int main( )
{
    int cid[9]= {0}; //init the entire array to zeros

    for (i = 0; i < 9; i++){
        cid[i] = fork();
        <Point Alpha>
        if ( <Condition> ) {
            ...some statements...
            exit(0);
        }
    }

    return 0;
}
```

1. In loop **`i = 3`**, how many zeroes are there in the **`cid[]`** array for the **<u>child process</u>** at **`<point alpha>`**? The **<u>child process</u>** refers to the process just created by the **`fork()`** system call in that loop interation.

   a. 6
   b. 7
   c. 8
   d. 9
   e. None of the above

[redacted]

[redacted]

2. Which of the following condition(s) can be used for <Condition> if we want to create a total of **9 additional processes (i.e. not counting the original process)**?
   i. `cid[i] == 0`
   ii. `cid[i] != 0`
   iii. `cid[0] == 0`

a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i), (ii) and (iii).
e. None of the above.

[redacted]

3. Which of the following statment(s) regarding **zombie** process is **TRUE**?
   i. Zombie process takes up a slot in the OS PCB table.
   ii. Zombie process is created so that **`wait()`** system call can be implemented properly.
   iii. A user command running in the **background** under a shell interpreter can become a zombie process.

a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i), (ii) and (iii).
e. None of the above.

[redacted]

[redacted]

4. Which of the following statment(s) regarding **Unix Shared Memory** IPC is **TRUE**?

   i. Shared memory region created by program **P** can stay around after **P** exited.
   ii. Shared memory region is identified by a pointer (memory address).
   iii. Shared memory region can be accessed by any process (including process from other users).

a. (i) only.
b. (i) and (iii) only.
c. (ii) and (iii) only.
d. (i), (ii) and (iii).
e. None of the above.

[redacted]

[redacted]

[redacted]

[redacted]

5. Given the following pseudo code:

Code A	Code B

`wait( S );`
`<do some work>`
`signal( S );`
	`<heavy computation>`

Which of the following setup can potentially cause **priority inversion**?

a. A high priority task running code B and a lower priority task running code A.
b. A high priority task running code A and a lower priority task running code B.
c. The highest and lowest priority tasks running code A and a middle priority task running task B.
d. The highest and lowest priority tasks running code B and a middle priority task running task A.
e. None of the above

[redacted]

6. Ms. Raycond coded the following function:

```
int globalVar = 0;    //shared among all threads
void* doSum( void* arg)
{
    int i, localVar = 0;

    for (i = 0; i < 50000; i++){
         localVar++;
    }
    globalVar += localVar;
}
```

If we spawn **two threads** to work on the **doSum() function** and **wait for them to finish**, what is the **most accurate** description of the program behavior?

a. The program is now deterministic with the globalVar equal to 100000 for all runs.
b. The program still exhibits race condition. The globalVar value can be 0, 50000 or 100000.
c. The program still exhibits race condition. The globalVar value can be 50000 or 100000.
d. The program still exhibits race condition. The globalVar value can be any positive number.
e. None of the above

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Section 2: Short Questions (28 marks)

Question 7 (7 marks)

Consider an array **A** of **N integer values**, a task can execute two operations: i) **IN**: read and remove one of the N values and ii) **OUT**: write into one of the N values. Below is an attempt to use semaphore to synchronize the tasks in operating on the array values:

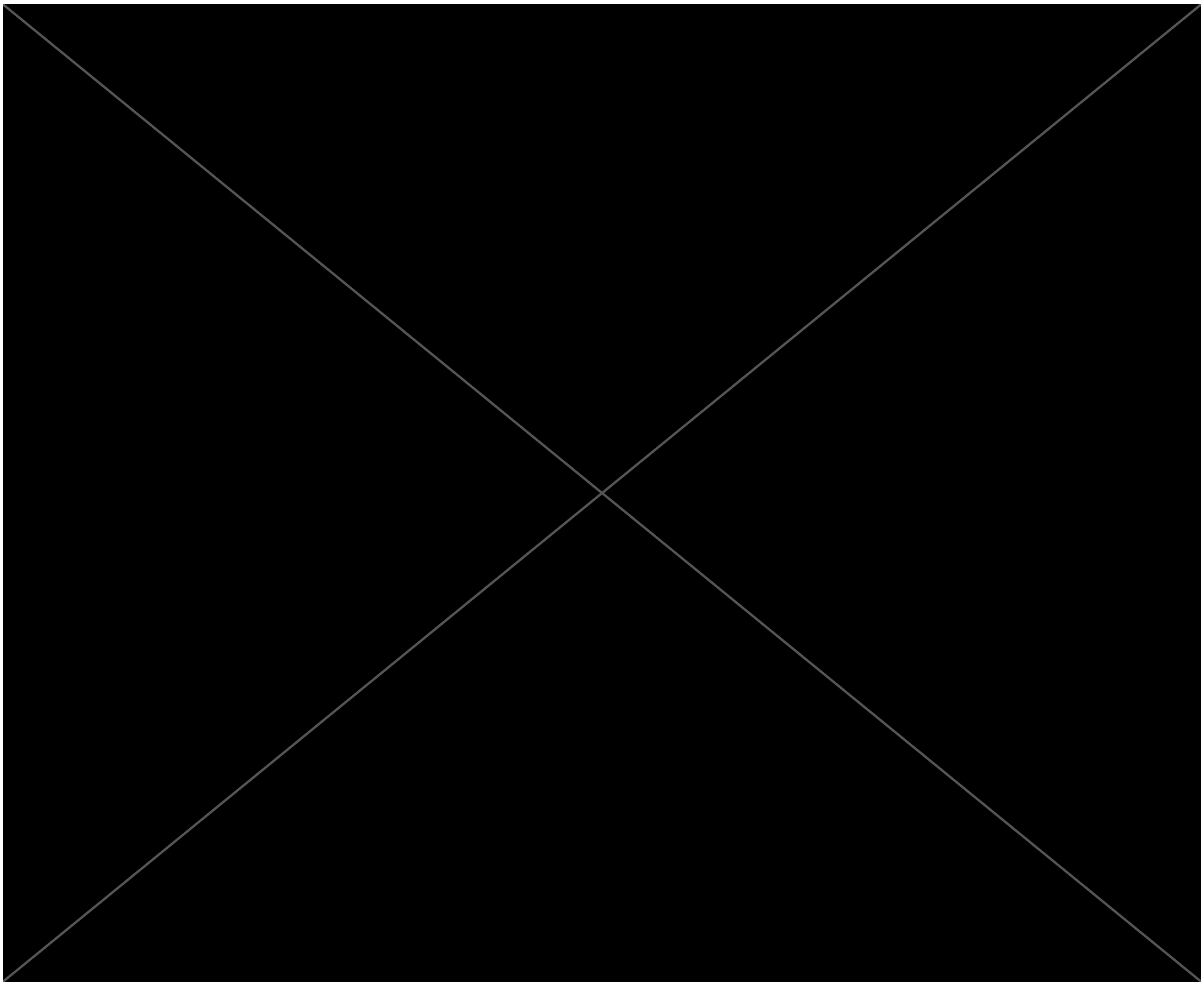
```
Semaphore mutex = 1;    //binary semaphore
int A[N];    //shared array

int IN( int idx ) {
      int result;
      wait( mutex );
      result = A[idx];
      //"remove" value
      A[idx] = -1;
      signal( mutex );
      return result;
}

void OUT( int idx, int newValue ) {
      wait( mutex );
      A[idx] = newValue;
      signal( mutex );
}
```

a. [2 marks] Briefly describe one shortcoming of this implementation.

b. [5 marks] Give an implementation that solve the shortcoming in (a). Note that you can only:

- Introduce / modify the semaphore declaration and initialization.
- Add **only** wait / signal to the IN and OUT operation.

[redacted]



Question 8 (6 marks)

The **responsiveness** of a scheduling algorithm refers to how soon can a newly created task receives its **first share of CPU time**. The following questions focus on a newly created task T_{new} added into an environment where **there are N (N > 0) ready to run tasks**. Restrict your answer to scheduling algorithms discussed in the course so far.

a. [4 marks] Give **two** algorithms that can be responsive. Briefly explain / describe how the algorithms enable responsiveness.

b. [2 marks] Give **one** algorithm that is irresponsive. Similarly explain / describe how the algorithm prohibits responsiveness.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Question 9 (7 marks)

Instead of using the stack memory, Mr. S. Penn suggested the following alternative to support function invocation:

- During compilation, all functions will be allocated a **predetermined** memory location to store their stack frame. (Similar to how global variable has a fixed memory location). There is no change to the stack frame structure.
- The stack pointer (SP) / frame pointer (FP) can simply points to these predetermined locations during the function execution.

For example, suppose there is a program with three functions: **`main()`**, **`f()`** and **`g()`**. The compiler allocated the following locations for their respective stack frame:

	`main()`		**`f()`**		**`g()`**
6600	stack frame items	4400	stack frame items	8800	stack frame items
	……		……		……
	<saved SP>		<saved SP>		<saved SP>

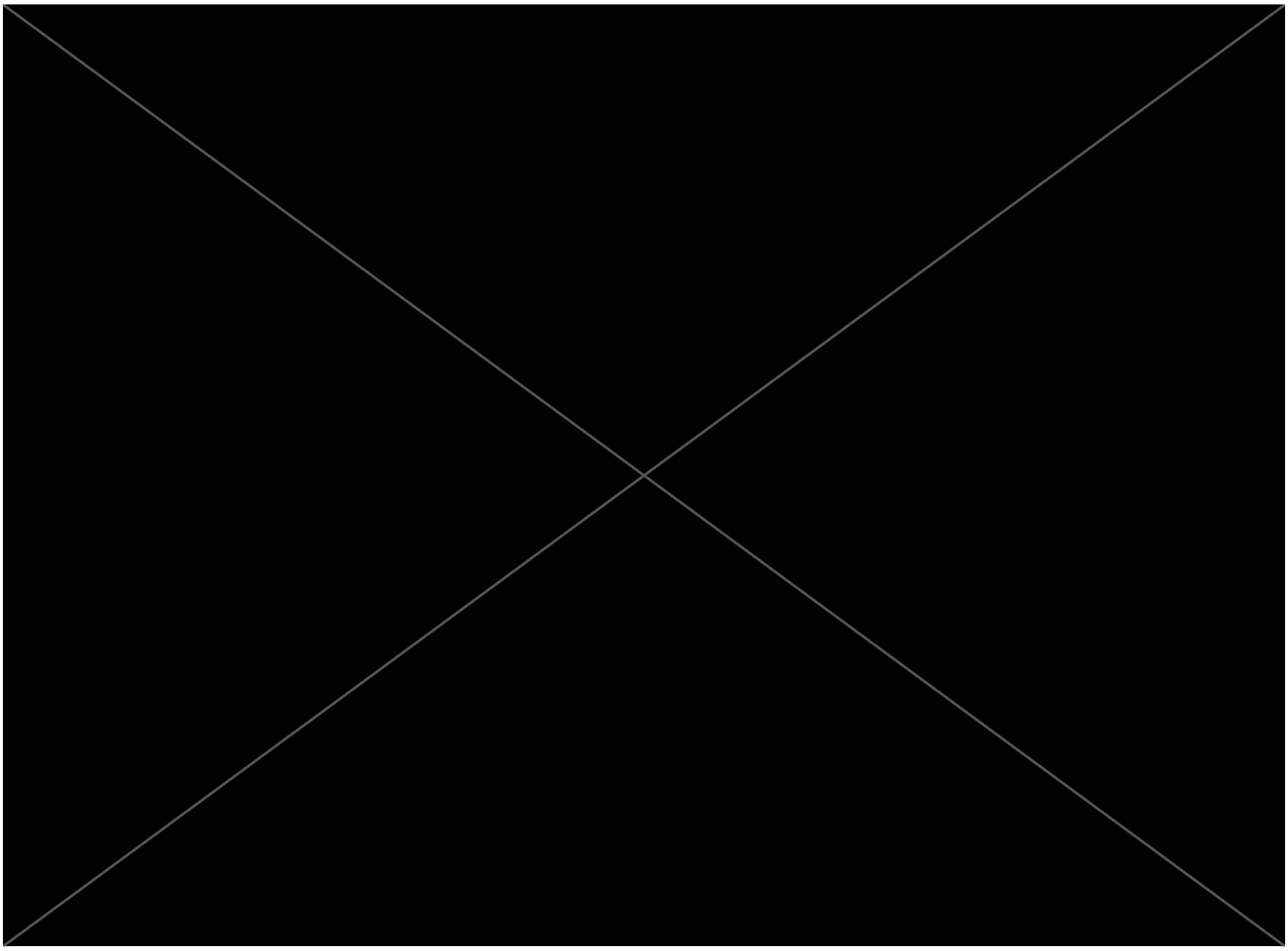
a. [1 mark] Suppose **`main()`** calls **`g()`**, show the value(s) of the **`<saved SP>`** in the relevant stack frame(s) when **`g()`** is still executing. Put a "**`---`**" for irrelevant **`<saved SP>`**. If you think it is not possible, please **cross out the stack frames** as an indication.

b. [2 marks] Suppose **`main()`** calls **`f()`** which calls **`g()`**, show the value(s) of the **`<saved SP>`** in the relevant stack frame(s) when **`g()`** is still executing. Put a "**`---`**" for irrelevant **`<saved SP>`**. If you think it is not possible, please **cross out the stack frames** as an indication.

c. [2 marks] What are the conditions for this implementation scheme to work?

d. [2 marks] Give one example where this implementation scheme fails?

[redacted]



Question 10 (8 marks)

Given the following two multi-threaded processes with their respective execution behavior:

P_1	**T_1**	C1, IO1, C1, IO1, C1, ….. repeats
	T_2	C20
P_2	**T_1**	C1, IO1, C1, IO1, C1, ….. repeats
	T_2	C20

If we use **the standard 3-level MLFQ** scheduling with a time quantum of **2 time units**, answer the following. Note that whenever there is a need to order the processes / threads, you can assume P_1 is ordered before P_2 and the respective T_1 is ordered before T_2.

a. [4 marks] Suppose the threads are implemented as **kernel threads**, give the first 8 time units of the CPU schedule. Remember to indicate both the process number and thread number, e.g. P_2T_1.

b. [4 marks] Suppose the threads are implemented as **user threads**, give the first 8 time units of the CPU schedule. Remember to indicate both the process number and thread number, e.g. P_2T_1. You should try to give fair CPU share to the threads within the same process as much as possible.

For both questions, give **important assumptions you have made (if any)**. You may not receive any mark if key assumption (any assumption not stated in question) is missing.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Section 3: Bonus Question (1 mark)

11. "Know what you don’t know": Predict your score for this assessment (excluding this bonus question). If your prediction is with ±2 marks of your actual score, you will get a bonus "true understanding" 1 mark. ☺

[redacted]