Quiz 2 Model answer

1. In the One to One model when a thread makes a blocking system call ----------.
   a. other threads are strictly prohibited from running
   b. other threads are allowed to run
   c. other threads only from other processes are allowed to run
   d. other threads only from other processes are blocked to run
   e. both (a) and (d)

2. Which one of the following is not shared by threads?
   a. program counter
   b. stack
   c. program code area
   d. **both (a) and (b)**
   e. both (a) and (c)

3. The model in which one kernel thread is mapped to many user-level threads is called ----------.
   a. Many to One model
   b. One to Many model
   c. Many to Many model
   d. One to One model
   e. Two level model

Consider the following figure .....

- Including the initial parent process,

-How many processes are created by the program?

- What statement(s) you can add to check if fork() failed and an error occurred?

```
int main()
{
    int i;

    for (i = 0; i < 4; i++)
      fork();

    return 0;
}
```

- 2^4 =16 process include parent.

- The student should draw a binary tree start f parent process each will create a child and keep itself.

- pid_t pid;

  pid = fork();

  if (pid == -1) or (pid<0)

  {

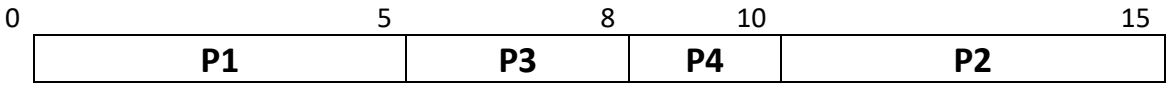
```
    //Error: * When fork() returns -1, an error happened
    fprintf(stderr, "can't fork, error %d\n", errno);
    exit(EXIT_FAILURE);
}
```

Assume you want to implement a web-server for **YouTube** by using multithreading, where each thread serves one incoming request by loading a video file from the disk. Assume the OS only provides the normal blocking read system call for disk reads, do you think user-level threads or kernel-level threads should be used? Why?

Kernel-level threads. Because each thread will make blocking I/O calls. With kernel-level thread, one thread won't block others. But if user-level thread is used, then one thread will block all other threads.

CPU Scheduling. Here is a table of processes and their associated arrival and running times. Show the scheduling order for these processes under Shortest-Job First (SJF) and calculate average waiting time

Process ID	Arrival Time	Expected CPU Running Time
Process 1	0	5
Process 2	1	5
Process 3	5	3
Process 4	6	2



Average Waiting time = (0+(10-1)+(5-5)+(8-6))/4= 11/4=2.75