Question 1

(a)

1.Memory space, Code and executable files, Open file descriptors, Signal handlers.

2.When working on the thread belongs to different process, we need to switch the page table.

(b)

No, because parent process will wait until child process finished. If there is only one thread in the process, one input will wait for another.

(c)

There will be 4.

First fork() will create a thread 1 from origin thread 0.

Because first fork() return the PID of the child process , which is greater than 0, so next fork() won’t be executed. We then have thread 0 and thread 1.

Then the third fork() will fork thread 0 and 1, leading to 4 threads at last.

(d)

1.No. When there are multiple threads running together, CPU may running other program which has higher priority.

2.It may be the same program before system call, may be operating system itself, may be some concurrent program.

3.There is a trade-off within. Special stack may be more efficient when doing muti-tasking, but would consume more memory and cause additional complexity of synchronization within a process. I think a hybrid approach would be better, that is set special stack space for each process and shared stack space for threads in one process.

Question 2

(a)

Philosopher 1: Acquires the left chopstick (chopsticks[1]).

Philosopher 2: Acquires the left chopstick (chopsticks[2]).

Philosopher 3: Acquires the left chopstick (chopsticks[3]).

Philosopher 4: Acquires the left chopstick (chopsticks[4]).

Philosopher 5: Tries to acquire the left chopstick (chopsticks[5]) but is blocked by Philosopher 0, who has it.

Under this circumstance, philosopher 1-4 has only one chopstick, so they can’t eat, thus can’t release chopsticks, while philosopher 5 has no chopstick to pick, they all stuck.