得分

第六题．请结合教材第十二章“**并发编程**”的有关知识回答问题(**15 分)**

“生产者-消费者”问题是并发编程中的经典问题。本题中，考虑如下场景：

a. 所有生产者和所有消费者**共享同一个 buffer**

b. 生产者、消费者各有 NUM_WORKERS 个（**大于一个**）

c. buffer 的容量为 BUF_SIZE，**初始情况下 buffer 为空**

d. 每个生产者向 buffer 中添加一个 item；若 buffer 满，则生产者等待 buffer 中有空槽时才能添加元素

e. 每个消费者从 buffer 中取走一个 item；若 buffer 空，则消费者等待 buffer 中有 item 时才能取走元素

1．阅读以下代码并回答问题(**代码阅读提示：主要关注 producer 和 consumer 两个函数**)

```
/* Producer-Consumer Problem (Solution 1) */

#include "csapp.h"

#define BUF_SIZE 3
#define NUM_WORKERS 50
#define MAX_SLEEP_SEC 10

volatile static int items = 0; /* How many items are there in the buffer */

    static sem_t mutex; /* Mutual Exclusion */
    static sem_t empty; /* How many empty slots are there in the buffer */
    static sem_t full;  /* How many items are there in the buffer */

    static void sync_var_init() {
        Sem_init(&mutex, 0, 1);

        /* Initially, there is no item in the buffer */
        Sem_init(&empty, 0, BUF_SIZE);
        Sem_init(&full, 0, 0);
    }

    static void *producer(void *num) {
        ①;
        ②;

        /* Critical section begins */
        Sleep(rand() % MAX_SLEEP_SEC);
        items++;
        /* Critical section ends */

        V(&mutex);
        V(&full);
34.
35.         return NULL;
36.     }
37.
38.     static void *consumer(void *num) {
39.         ③;
40.         ④;
41.
42.         /* Critical section begins */
43.         Sleep(rand() % MAX_SLEEP_SEC);
44.         items--;
45.         /* Critical section ends */
46.
47.         V(&mutex);
48.         V(&empty);
49.
50.         return NULL;
51.     }
52.
53.     int main() {
54.         sync_var_init();
55.
56.         pthread_t pid_producer[NUM_WORKERS];
57.         pthread_t pid_consumer[NUM_WORKERS];
58.
59.         for (int i = 0; i < NUM_WORKERS; i++) {
60.             Pthread_create(&pid_producer[i], NULL, producer, (void *)i);
61.             Pthread_create(&pid_consumer[i], NULL, consumer, (void *)i);
62.         }
63.
64.         for (int i = 0; i < NUM_WORKERS; i++) {
65.             Pthread_join(pid_producer[i], NULL);
66.             Pthread_join(pid_consumer[i], NULL);
67.         }
68.     }
```

a)　补全代码（请从以下选项中选择，可重复选择，每个 1 分，共 4 分）

①______（24 行）
②______（25 行）
③______（39 行）
④______（40 行）

选项：

A.　P(&mutex)
B.　P(&empty)
C.　P(&full)

b)　如果交换 24 行与 25 行（两个 P 操作），__________（单选，2 分）

A.　有可能死锁
B.　有可能饥饿
C.　既不会死锁，也不会饥饿

c)　交换 32、33 行（两个 V 操作）是否可能造成同步错误？_____（2 分）
A.　可能
B.　不可能

d)　rand 函数是不是线程安全的？____（1 分）
A.　是
B.　不是

28 行与 43 行对 rand 函数的使用是否会导致竞争？____（1 分）
A.　会
B.　不会

已知 rand 函数的实现如下
来源：
https://github.com/begriffs/libc/blob/master/stdlib.h
https://github.com/begriffs/libc/blob/master/stdlib.c

```
#define RAND_MAX 32767

unsigned long _Randomseed = 1;

int rand() {
    _Randomseed = _Randomseed * 1103515425 + 12345;
    return (unsigned int)(_Randomseed>>16) & RAND_MAX;
}

void srand(unsigned int seed) {
    _Randomseed = seed;
}
```

2. 考虑“生产者-消费者”问题的另一种解法（**代码阅读提示：12-69 行之外均与上一种解法相同**）

```
1.  /* Producer-Consumer Problem (Solution 2) */
2.
3.  #include "csapp.h"
4.
5.  #define BUF_SIZE 3
6.  #define NUM_WORKERS 50
7.  #define MAX_SLEEP_SEC 10
8.
9.  volatile static int items = 0; /* How many items are there in the buffer */
10.
11. static sem_t mutex;                 /* Mutual Exclusion */
12. static sem_t sem_waiting_producer; /* Wait for empty slots */
13. static sem_t sem_waiting_consumer; /* Wait for available items */
14.
15. volatile static int num_waiting_producer = 0;
16. volatile static int num_waiting_consumer = 0;
```

```

static void sync_var_init() {
    Sem_init(&mutex, 0, 1);

    Sem_init(&sem_waiting_producer, 0, ①);
    Sem_init(&sem_waiting_consumer, 0, ①);
}

static void *producer(void *num) {
    P(&mutex);
    while (items == BUF_SIZE) {
        num_waiting_producer++;
        ②;
        ③;
        P(&mutex);
    }

    /* Critical section begins */
    Sleep(rand() % MAX_SLEEP_SEC);
    items++;
    /* Critical section ends */

    if (num_waiting_consumer > 0) {
        num_waiting_consumer--;
        V(&sem_waiting_consumer);
    }
    V(&mutex);

    return NULL;
}

static void *consumer(void *num) {
    P(&mutex);
    while (items == 0) {
        num_waiting_consumer++;
        ④;
        ⑤;
        P(&mutex);
    }

    /* Critical section begins */
    Sleep(rand() % MAX_SLEEP_SEC);
    items--;
    /* Critical section ends */

    if (num_waiting_producer > 0) {
        num_waiting_producer--;
        V(&sem_waiting_producer);
    }
    V(&mutex);

    return NULL;
}

int main() {
    sync_var_init();

74.     pthread_t pid_producer[NUM_WORKERS];
75.     pthread_t pid_consumer[NUM_WORKERS];
76.
77.     for (int i = 0; i < NUM_WORKERS; i++) {
78.         Pthread_create(&pid_producer[i], NULL, producer, (void *)i);
79.         Pthread_create(&pid_consumer[i], NULL, consumer, (void *)i);
80.     }
81.
82.     for (int i = 0; i < NUM_WORKERS; i++) {
83.         Pthread_join(pid_producer[i], NULL);
84.         Pthread_join(pid_consumer[i], NULL);
85.     }
86. }
```

a) 补全代码（请从以下选项中选择，④⑤无需填写，每个 1 分，共 3 分）

①______（21、22 行）

②______（29 行）

③______（30 行）

选项：

A. 0

B. 1

C. P(&sem_waiting_producer)

D. V(&mutex)

b) 如果 27 行和 50 行的 while 换成 if，是否可能造成同步错误？

_____（2 分）

A. 可能

B. 不可能