پاسخنامه کوئیز سیستم عامل – دکتر ازهری

مبحث Synchronization

۱.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define BUFFER_SIZE 10
int buffer[BUFFER_SIZE];
int count =0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* generate_random_numbers(void* thread_id) {
    int id = *((int *) thread_id);
    while(1) {
        // Generate a random number
        int random_number = rand() % 100;
        pthread_mutex_lock(&mutex);
        if (count < BUFFER_SIZE) {
            buffer[count] = random_number;
            count++;
            printf("Thread % wrote %d to the buffer\n", id, random_number);
        }
        else {
            printf("Thread %d: Buffer is full. Skipping...\n", id);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    // Seed the random number generator
    srand(time(NULL));
```

```
            printf("Thread % wrote %d to the buffer\n", id, random_number);
        }
        else {
            printf("Thread %d: Buffer is full. Skipping...\n", id);
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    // Seed the random number generator
    srand(time(NULL));

    pthread_t threads[3];
    int thread_ids[3] = {1, 2, 3};

    // Create three threads
    for(int i=0; i<3; i++) {
        if (pthread_create(&threads[i], NULL, generate_random_numbers, (void
                           *)&thread_ids[i]) != 0) {
            fprintf(stderr, "Error creating thread %d/n", i);
            return 1;
        }
    }

    for(int i=0; i<3; ++i) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

۲. اصل mutual exclusion برقرار نیست. زیرا هر دو فرآیند می‌توانند خط قبل از ورود به حلقه را اجرا کنند (true قراردادن متغیرهای varQ و varP) و سپس هردو همزمان می‌توانند وارد حلقه و درنتیجه ناحیه بحرانی شوند.