دانشکده مهندسی کامپیوتر

درس سیستم‌های عامل

پاسخنامه کوییز سوم

مدرس ..... دکتر رضا انتظاری ملکی، دکتر وحید ازهری

طراح ..... علیرضا اسلامی‌خواه

۱) کد زیر یکی از مسائل دنیای سیستم های عامل است که به مسئله Producer Consumer معروف میباشد. حال از شما انتظار میرود اول کد را با دقت خوانده و سعی کنید چگونگی کار آن را متوجه شوید سپس به سوالات زیر پاسخ دهید.

```
1 sem_t empty;
2 sem_t full;
3 sem_t mutex;
4
5 void *producer(void *arg){
6   int i;
7   for (i = 0; i < loops; i++) {
8       sem_wait(&mutex);
9       put(i);
10      sem_post(&full);
11      }
12 }
13
14 void *consumer(void *arg) {
15  int i;
16  for (i = 0; i < loops; i++) {
17      sem_wait(&full);
18      int tmp = get();
19      sem_post(&mutex);
20      printf("%d\n", tmp);
21     }
22 }
23
24 int main(int argc, char *argv[]) {
25 // ...
26 sem_init(&empty, ?, ?);// بخش الف
27 sem_init(&full, ?, ?); // بخش الف
28 sem_init(&mutex, 0, 1);
29 // ...
30 }
```

الف) مقدار value های سمافورهای empty و full باید چه اعدادی باشد؟

ب) آیا این کد خروجی درستی به ما میدهد؟ اگر نیاز به تغییرات دارد از شما میخواهیم بخش هایی که باید به کد اضافه شود را اضافه کنید. (توجه کنید که خط ۱۸ و ۹ صحیح است و نیازی به تغییر ندارد.)

پ) با توجه به آنچه که در جزوه درسی خوانده اید ، فرض کنید در تابع producer یا consumer ما جای خطی که Mutex را اشغال (wait) میکند با خطی که سمافور Empty یا Full را اشغال میکند جا به جا کنیم. آنگاه کدام یک از اصول سه گانه (Mutual Exclusion , Progress , Bounded Waiting) نقض میشود؟ چرا؟

پاسخ:

۱. کد کامل:

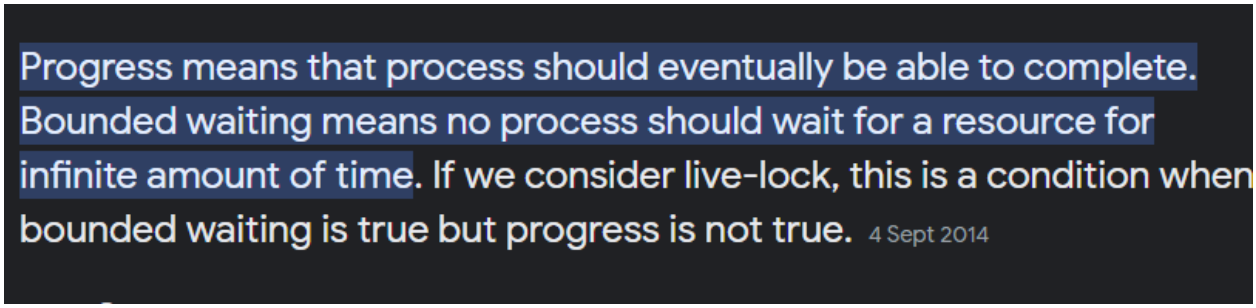
```
1   sem_t empty;
2   sem_t full;
3   sem_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8           sem_wait(&empty);           // line p1
9           sem_wait(&mutex);           // line p1.5 (MOVED MUTEX HERE...)
10          put(i);                     // line p2
11          sem_post(&mutex);           // line p2.5 (... AND HERE)
12          sem_post(&full);            // line p3
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          sem_wait(&full);            // line c1
20          sem_wait(&mutex);           // line c1.5 (MOVED MUTEX HERE...)
21          int tmp = get();            // line c2
22          sem_post(&mutex);           // line c2.5 (... AND HERE)
23          sem_post(&empty);           // line c3
24          printf("%d\n", tmp);
25      }
26  }
27
28  int main(int argc, char *argv[]) {
29      // ...
30      sem_init(&empty, 0, MAX); // MAX buffers are empty to begin with...
31      sem_init(&full, 0, 0);    // ... and 0 are full
32      sem_init(&mutex, 0, 1);   // mutex=1 because it is a lock
33      // ...
34  }
```

الف) مقدار full باید ۱ و مقدار empty باید به تعداد thread های استفاده شده در کد باشد.

ب) مانند کد بالا باید پر شود.

پ) با توجه به این تعریف مشخصا مسئله bounded waiting پیش میاید زیرا اگر mutex را اول بگذاریم باعث میشود که دو پروسس به منابع هم وابسته در نتیجه پیشرفتی حاصل نمیشود. در این سوال جواب progress هم صحیح حساب میشود زیرا در هر صورت کد حرکت رو به جلویی ندارد و در یکجا متوقف میشود.

در اینجا توضیحات بیشتری درباره این دو داده شده است:



شرط mutual exclusion در این بخش هم زمانی نقض میشود که مثلا producer خط full post را اجرا کرده ولی قبل ازینکه تابع مقابل full را wait کند باعث شود که با critical section همدیگر تداخل داشته باشند.

نکته این سوال توضیحات و چرایی مربوط به نقض شونده است. لذا هرکدام از این موارد که ذکر میشود باید توضیحات آن برای گرفتن نمره کامل هم درج شود.

۲) در این سوال از شما خواسته می‌شود کدی بنویسید که در آن ۳ رشته (Thread) مختلف اعدادی تصادفی ایجاد کرده و درون یک بافر با اندازه ۱۰ بریزند. به کامنت های کد و بخش های زده شده توجه کنید زیرا میتوانند شمارا راهنمایی کنند.

```
#define BUFFER_SIZE 10
int buffer[BUFFER_SIZE];
int count = 0;
pthread_mutex_t mutex ;
void *generate_random_numbers(void *thread_id) {
```

```
    // Generate a random number

    // Write the random number to the buffer if there is space
}

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)
    {
        __________(&threads[i], NULL, generate_random_numbers, (void
*)&thread_ids[i]);// جای خالی را پر کنید
    }
    // Join the threads

    return 0;
}
```

پاسخ:

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <pthread.h>
4. #include <unistd.h>
5.
6. #define BUFFER_SIZE 10
7.
8. int buffer[BUFFER_SIZE];
9. int count = 0;
10.
11.pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
12.
13.void *generate_random_numbers(void *thread_id) {
14.     int id = *((int *)thread_id);
15.     while (1) {
16.         // Generate a random number
17.         int random_number = rand() % 100;
18.         pthread_mutex_lock(&mutex);
19.         if (count < BUFFER_SIZE) {
```

```
            buffer[count] = random_number;
            count++;
            printf("Thread %d 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;
}
```