دستور کار آزمایشگاه سیستم های عامل

آزمایشگاه سیستم های عامل

آرین محسنی

کاوه احمدی

آزمایش ۹

بخش دوم و سوم

دی 1403

دستور کار آزمایشگاه سیستم های عامل

در بخش دوم و سوم آزمایش 9 میخواهیم به مسئله خوانندگان نویسندگان و مسئله غذای فیلسوفان بپردازیم.

بخش دوم :

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

#define MAX_COUNT 20  // The maximum value of count

sem_t rw_mutex;         // Semaphore to manage writer access
sem_t mutex;            // Semaphore to manage reader count access
int read_count = 0;     // Number of active readers
int count = 0;          // Shared buffer

void *reader(void *arg) {
    int pid = *((int *)arg);

    while (1) {
        sem_wait(&mutex);       // Lock to modify reader count
        read_count++;
        if (read_count == 1) { // First reader locks the buffer
            sem_wait(&rw_mutex);
        }
        sem_post(&mutex);       // Unlock reader count

        // Reading the buffer
        printf("Reader %d reads count: %d\n", pid, count);

        sem_wait(&mutex);
        read_count--;
        if (read_count == 0) { // Last reader unlocks the buffer
            sem_post(&rw_mutex);
        }
        sem_post(&mutex);

        sleep(1);
    }

    return NULL;
}
```

```c
void *writer(void *arg) {
    int pid = *((int *)arg);

    while (count < MAX_COUNT) {
        sem_wait(&rw_mutex); // Lock the buffer for writing

        count++;
        printf("Writer %d writes count: %d\n", pid, count);

        sem_post(&rw_mutex); // Unlock the buffer
        sleep(1);
    }

    return NULL;
}

int main() {
    pthread_t r1, r2, w1;
    int r1_id = 1, r2_id = 2, w1_id = 1;

    // Initialize semaphores
    sem_init(&rw_mutex, 0, 1);
    sem_init(&mutex, 0, 1);

    // Create threads
    pthread_create(&r1, NULL, reader, &r1_id);
    pthread_create(&r2, NULL, reader, &r2_id);
    pthread_create(&w1, NULL, writer, &w1_id);

    // Wait for threads to complete
    pthread_join(w1, NULL);
    pthread_cancel(r1);  // Stop readers after the writer is done
    pthread_cancel(r2);

    // Destroy semaphores
    sem_destroy(&rw_mutex);
    sem_destroy(&mutex);

    return 0;
}
```

دستور کار آزمایشگاه سیستم های عامل

خروجی :

```
Reader 1 reads count: 0
Reader 2 reads count: 0
Writer 1 writes count: 1
Reader 1 reads count: 1
Writer 1 writes count: 2
Reader 2 reads count: 2
Reader 1 reads count: 2
Reader 2 reads count: 2
Writer 1 writes count: 3
Reader 1 reads count: 3
Writer 1 writes count: 4
Reader 2 reads count: 4
Reader 1 reads count: 4
Writer 1 writes count: 5
Reader 2 reads count: 5
Reader 1 reads count: 5
Writer 1 writes count: 6
Reader 2 reads count: 6
Reader 1 reads count: 6
Writer 1 writes count: 7
Reader 2 reads count: 7
Writer 1 writes count: 8
Reader 1 reads count: 8
Reader 2 reads count: 8
```

برنامه مربوطه را بصورت کامل نوشته و سپس اجرا کنید. به سوالات زیر پاسخ دهید:

- آیا مشکلی وجود دارد؟
- در صورت وجود ناهماهنگی چه راهکاری ارائه می‌کنید؟

بله اگر سمافور ها به طور درست استفاده نشده باشد حالت مسابقه پیش می آید و ددلاک ممکن است رخ دهد.

برای راهکار میتوان از درست بودن سمافور ها اطمینان حاصل کرد و ترتیب سمافور هارا رعایت کرد.

دستور کار آزمایشگاه سیستم های عامل

بخش سوم :

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

#define NUM_PHILOSOPHERS 5

sem_t chopstick[NUM_PHILOSOPHERS];  // Semaphores representing chopsticks

void *philosopher(void *arg) {
    int id = *((int *)arg);

    while (1) {
        // Thinking
        printf("Philosopher %d is thinking.\n", id);
        sleep(rand() % 3);

        // Picking up chopsticks
        sem_wait(&chopstick[id]);             // Pick up left chopstick
        sem_wait(&chopstick[(id + 1) % NUM_PHILOSOPHERS]); // Pick up right chopstick

        // Eating
        printf("Philosopher %d is eating.\n", id);
        sleep(rand() % 2);

        // Putting down chopsticks
        sem_post(&chopstick[id]);             // Put down left chopstick
        sem_post(&chopstick[(id + 1) % NUM_PHILOSOPHERS]); // Put down right chopstick
    }

    return NULL;
}
```

```c
int main() {
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int ids[NUM_PHILOSOPHERS];

    // Initialize semaphores
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_init(&chopstick[i], 0, 1);
        ids[i] = i;
    }

    // Create philosopher threads
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_create(&philosophers[i], NULL, philosopher, &ids[i]);
    }

    // Wait for threads to finish (they won't in this example)
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
    }

    // Destroy semaphores
    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        sem_destroy(&chopstick[i]);
    }

    return 0;
}
```

دستور کار آزمایشگاه سیستم های عامل

خروجی :

```
Philosopher 0 is thinking.
Philosopher 4 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 0 is eating.
Philosopher 0 is thinking.
Philosopher 4 is eating.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 1 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is eating.
Philosopher 3 is eating.
Philosopher 0 is thinking.
Philosopher 3 is thinking.
Philosopher 3 is eating.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 2 is thinking.
Philosopher 1 is eating.
Philosopher 1 is thinking.
Philosopher 1 is eating.
Philosopher 4 is eating.
Philosopher 4 is thinking.
Philosopher 3 is eating.
```

سوال: آیا ممکن است بن بست رخ دهد؟ در صورت امکان چگونگی ایجاد آن را توضیح دهید.

بله ددلاک میتواند اتفاق بیفتد و اگر همه فیلسوف ها چوب سمت چپ خودش را بردارد هر فیلسوف باید برای چوب راستی صبر کند(زیرا دایره ای نشسته اند).