به نام خدا

پاسخ تمرین سری ششم درس سیستم های عامل

پاییز 1403

استاد درس: دکتر زرندی

سوال اول)

ناحیه بحرانی را تعریف کنید و شروط لازم و کافی را برای آن نام ببرید و به صورت مختصر توضیح دهید

ناحیه بحرانی (Critical Section) به بخشی از کد یا دستورالعمل‌های یک برنامه گفته می‌شود که دسترسی همزمان چندین فرآیند به آن می‌تواند منجر به مشکلاتی نظیر ناسازگاری داده‌ها یا شرایط رقابتی (Race Condition) شود. در این ناحیه، تنها یک فرآیند در هر لحظه باید اجازه اجرا داشته باشد.

شروط لازم برای ناحیه بحرانی:

1. **تبادل متقابل (Mutual Exclusion): در هر لحظه، تنها یک فرآیند می‌تواند وارد ناحیه بحرانی شود و سایر فرآیندها باید منتظر بمانند تا فرآیند حاضر در ناحیه بحرانی کار خود را به اتمام برساند.**
2. **پیشرفت (Progress): اگر هیچ فرآیندی در ناحیه بحرانی نباشد، فرآیندهایی که درخواست ورود دارند باید بتوانند وارد شوند و اولویت‌بندی بین فرآیندها باید تضمین شود و فرآیندها نباید در حالت انتظار دائمی بمانند.**
3. **محدودیت انتظار (Bounded Waiting): هر فرآیند منتظر ورود به ناحیه بحرانی، باید پس از تعداد محدودی تلاش از سایر فرآیندها، اجازه ورود پیدا کند. این شرط مانع از Starvation فرآیندها می‌شود.**

سوال دوم)

دو روش برای مدیریت نواحی بحرانی به صورت Preemptive و Non preemptive است این دو را توضیح دهید و برای هرکدام یک مثال بیاورید که در چه نوع سیستم هایی بهتر است استفاده شوند.

- **روش Preemptive: در این روش، سیستم می‌تواند اجرای فرآیند یا رشته‌ای که در ناحیه بحرانی قرار دارد را متوقف کند و اجازه دهد فرآیند یا رشته دیگری اجرا شود. مدیریت در این روش با استفاده از سیستم‌عامل یا زمان‌بند انجام می‌شود.**
  کاربرد: در سیستم‌های بلادرنگ (Real-Time Systems) که فرآیندهای اولویت‌دار وجود دارند و زمان پاسخ‌دهی حیاتی است.
- **روش Non Preemptive: در این روش، وقتی فرآیندی وارد ناحیه بحرانی شد، اجرای آن تا زمانی که کامل شود متوقف نمی‌شود. سایر فرآیندها مجبورند منتظر بمانند تا فرآیند جاری کار خود را تمام کند و از ناحیه بحرانی خارج شود.**
  کاربرد: در سیستم‌های تعبیه‌شده (Embedded Systems) یا سیستم‌های قدیمی که مدیریت ساده‌تر و اطمینان از عدم وقفه در اجرای فرآیند ضروری است.

سوال سوم)

در رابطه با نواحی بحرانی به سوالات زیر پاسخ دهید.

الف) دستورات Atomic به چه دستوراتی گفته می شوند؟

دستورات اتمی (Atomic) به دستوراتی گفته می‌شود که اجرای آنها به صورت غیرقابل تقسیم است؛ یعنی یا به طور کامل انجام می‌شوند یا اصلاً انجام نمی‌شوند، و در هنگام اجرای آنها هیچ‌گونه وقفه یا مداخله‌ای از سوی دیگر فرآیندها یا رشته‌ها رخ نمی‌دهد.

این دستورات به طور سخت‌افزاری یا نرم‌افزاری تضمین می‌کنند که شرایط رقابتی (Race Condition) رخ نمی‌دهد.

ب) دو مورد از برتری های استفاده از Semaphore بجای Mutex را توضیح دهید.

- **پشتیبانی از چندین فرآیند یا رشته: Semaphore می‌تواند دسترسی را به تعداد مشخصی از منابع محدود کند (مانند مقدار اولیه ۳ برای حداکثر سه فرآیند همزمان). اما Mutex تنها به یک فرآیند اجازه دسترسی می‌دهد. این ویژگی Semaphore را برای مدیریت منابع اشتراکی مناسب‌تر می‌کند.**
- **غیرمتعلق بودن به یک فرآیند: Semaphore می‌تواند توسط هر فرآیندی آزاد شود، اما Mutex تنها توسط فرآیندی که آن را قفل کرده است آزاد می‌شود. این ویژگی، انعطاف‌پذیری بیشتری به Semaphore می‌دهد.**

ج) الگوریتم پترسون را برای پشتیبانی از N پردازنده بازنویسی کنید و سپس Mutual Exclusion, Progress, Bounded Waiting را در الگوریتم خود بررسی کنید

```python
# Initialization
flag = [False] * N  # نشان‌دهنده علاقه هر پردازنده
turn = [0] * N      # تعیین نوبت برای هر پردازنده

# Code for process i
def process(i):
    while True:
        for j in range(N):  # اعلام علاقه
            if j != i:
                flag[i] = True
                turn[i] = j
                while flag[j] and turn[i] == j:
                    pass  # منتظر بمان

        # Critical Section
        print(f"Process {i} is in the critical section")

        # خروج از ناحیه بحرانی
        flag[i] = False

        # Remainder Section
        print(f"Process {i} is in the remainder section")
```

- **تبادل متقابل (Mutual Exclusion): از آرایه flag و مکانیزم turn برای جلوگیری از دسترسی همزمان استفاده می‌شود. اگر یک فرآیند در ناحیه بحرانی باشد، سایر فرآیندها نمی‌توانند وارد شوند.**
- **پیشرفت (Progress): زمانی که ناحیه بحرانی خالی باشد، فرآیندهای علاقه‌مند به ترتیب وارد می‌شوند. شرط while flag[j] and turn[i] == j تضمین می‌کند که فرآیندها به ترتیب منتظر بمانند.**
- **محدودیت انتظار (Bounded Waiting): به هر فرآیند اطمینان داده می‌شود که پس از تعداد محدودی چرخش (انتظار در حلقه)، نوبت ورود به ناحیه بحرانی را خواهد داشت. این شرط با تغییر نوبت (turn) و استفاده از آرایه flag رعایت می‌شود.**

*** پاسخ های صحیح دیگر نیز مورد قبول هستند ***

سوال چهارم)

دو پردازنده برای حل مسائله ناحیه بحرانی از روش های زیر استفاده کردند (متغیر های L1 و L2 در هر دو مشترک هستند و مقدار Boolean دارند و در ابتدا به صورت تصادفی مقدار دهی شده اند) هر کدام از ۳ شرط

Mutual Exclusion, Progress, Bounded Waiting را بررسی کنید و توضیح دهید.

```
// P1
while (L1 != L2);

//Critical Section

L1 = !L2
```

```
// P2
while (L1 == L2);

//Critical Section

L1 = L2
```

Progress ندارد

فرض کنیم 2S==1S است و پردازه ی ۱ قصد ورود به ناحیه بحرانی را ندارد. در این صورت پردازه ی ۲ که قصد ورود به ناحیه

بحرانی را دارد باید منتظر پردازه ی اول بماند و نمیتواند وارد ناحیه ی بحرانی شود

Mutual Exclusion دارد

در هر لحظه یا 1S == 2S است یا نیست و در هر کدام از این حالت فقط یکی از پردازه ها میتوانند در ناحیه ی بحرانی باشند.

Bounded Waiting دارد

بعد از حداکثر یک بار ورود یک پردازه به ناحیه ی بحرانی، نوبت ورود پردازه ی دیگر میشود. به عبارتی پردازه ی اول حداکثر باید به اندازه ی یک پردازه برای ورود به ناحیه ی بحرانی صبر کند.

سوال پنجم)

کلاس زیر که پیاده سازی سمافور است را کامل کنید و توضیح دهید هر بخش از کد که اضافه می کنید، چگونه به حفظ ۳ شرط Mutual Exclusion, Progress, Bounded Waiting کمک می کند (فرض کنید که کلاس Process دو متد block و wakeup دارد)

```
class Semaphore {
    private value: number; // مقدار سمافور
    private queue: Queue<Process>; // صفی از فرآیندهای منتظر

    // سازنده کلاس
    constructor(initialValue: number) {
        this.value = initialValue; // مقدار اولیه سمافور
        this.queue = new Queue<Process>(); // صف خالی برای فرآیندها
    }

    // متد wait برای کاهش مقدار سمافور و مسدود کردن فرآیند در صورت نیاز
    wait(process: Process): void {
        this.value--; // کاهش مقدار سمافور

        if (this.value < 0) {
            // اگر مقدار سمافور منفی شد، فرآیند باید مسدود شود
            this.queue.enqueue(process); // فرآیند را به صف اضافه می‌کنیم
            process.block(); // فرآیند را مسدود می‌کنیم
        }
    }

    // متد signal برای افزایش مقدار سمافور و بیدار کردن فرآیندهای منتظر
    signal(): void {
        this.value++; // افزایش مقدار سمافور

        if (this.value <= 0) {
            // اگر فرآیندی در صف منتظر باشد، آن را بیدار می‌کنیم
            const nextProcess = this.queue.dequeue(); // بعدی را از صف خارج می‌کنیم
            if (nextProcess) {
                nextProcess.wakeup(); // فرآیند را بیدار می‌کنیم
            }
        }
    }
}
```

1. Mutual Exclusion (تبادل متقابل):

کد مرتبط: کاهش مقدار value در متد wait و مسدود کردن فرآیند در صورت منفی شدن مقدار.

چگونگی رعایت شرط:

با استفاده از مقدار value، اگر این مقدار صفر یا منفی شود، فرآیندهای جدید اجازه ورود به ناحیه بحرانی را نخواهند داشت و در صف منتظر می‌مانند. این تضمین می‌کند که فقط تعداد محدودی (یا دقیقاً یک) فرآیند بتواند در ناحیه بحرانی فعالیت کند.

2. Progress (پیشرفت):

کد مرتبط: بیدار کردن فرآیندها در متد signal.

چگونگی رعایت شرط:

در صورتی که فرآیندها در صف منتظر باشند، signal بلافاصله فرآیند اول صف را بیدار می‌کند. این تضمین می‌کند که در صورت خالی شدن ناحیه بحرانی، فرآیندها بدون وقفه وارد شوند و سیستم دچار قفل یا توقف نشود.

3. Bounded Waiting (محدودیت انتظار):

کد مرتبط: مدیریت صف فرآیندها در queue.

چگونگی رعایت شرط:

صف FIFO تضمین می‌کند که فرآیندها به ترتیب وارد ناحیه بحرانی شوند. این ساختار صف از گرسنگی (Starvation) جلوگیری می‌کند و اطمینان می‌دهد که هیچ فرآیندی برای مدت نامحدود منتظر نمی‌ماند.