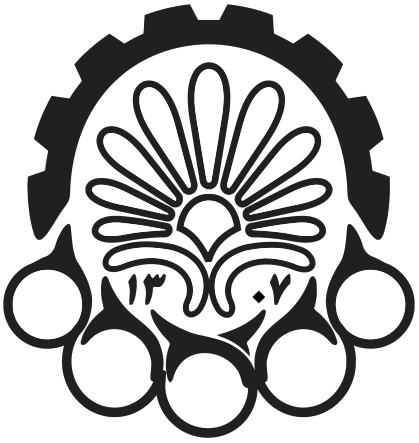
سیستم‌های عامل

دکتر زرندی



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری ششم

۱۹ آبان ۱۴۰۳

دانشکده مهندسی کامپیوتر

سیستم‌های عامل

تمرین سری ششم

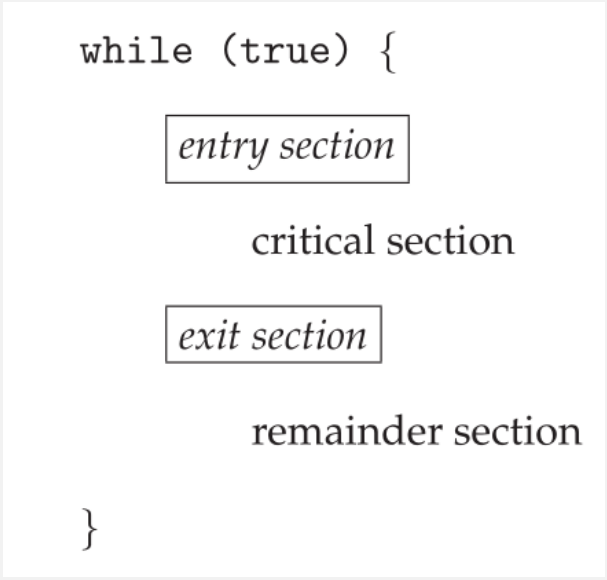
رضا آدینه پور ۴۰۲۱۳۱۰۵۵

سوال اول

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

پاسخ

مطابق با تعریف کتاب آقای silberschatz در صفحه ۲۶۰، می‌توان گفت: ناحیه بحرانی بخشی از برنامه است که در آن فرآیندها یا رشته‌ها به منابع مشترک دسترسی پیدا می‌کنند که ممکن است باعث تداخل و ناسازگاری در نتایج شود. در یک سیستم Multi task برای جلوگیری از مشکلات ناشی از دسترسی هم‌زمان به منابع مشترک، باید دسترسی به ناحیه بحرانی به‌درستی مدیریت شود. در شکل زیر این این مشکل آورده شده است:



شکل ۱: ساختار عمومی مسئله ناحیه بحرانی

در ادامه اگر منظور از شروط لازم و کافی، شروط لازم و کافی برای حل مشکل ناحیه بحرانی باشد می‌توان آن را به ۳ دست زیر تقسیم نمود:

۱. **انحصار متقابل یا Mutual exclusion:** اگر فرآیند P_i در حال اجرا در ناحیه بحرانی خود باشد، هیچ فرآیند دیگری نمی‌تواند در ناحیه بحرانی خود اجرا شود. یا یه عبارتی دیگر، در هر لحظه، فقط یک فرآیند اجازه دارد که وارد ناحیه بحرانی شود. این شرط مانع از دسترسی هم‌زمان چندین فرآیند به منابع مشترک می‌شود.

۲. **پیشرفت یا Progress:** در صورتی که هیچ فرآیندی در ناحیه بحرانی نباشد، فرآیندهای آماده‌ی ورود به ناحیه بحرانی نباید بدون دلیل منتظر بمانند. این شرط تضمین می‌کند که در صورت امکان، فرآیندهای آماده به ناحیه بحرانی دسترسی پیدا کنند.

۳. **انتظار محدود یا Bounded Waiting:** هر فرآیند نمی‌تواند برای همیشه منتظر بماند تا وارد ناحیه بحرانی شود. این شرط تضمین می‌کند که پس از مدتی محدود، هر فرآیند می‌تواند به ناحیه بحرانی دسترسی پیدا کند و به Starvation دچار نمی‌شود.

سوال دوم

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

پاسخ

همانطور که در صورت سوال نیز بیان شد، دو روش کلی برای مدیریت نواحی بحرانی در سیستم‌عامل‌ها استفاده می‌شود: روش Preemptive و روش Non-Preemptive

۱. در روش Preemptive سیستم عامل می‌تواند یک فرآیند را در هنگام اجرای ناحیه بحرانی به‌صورت خودکار متوقف کند و کنترل را به فرآیند دیگری واگذار کند. در این حالت، فرآیند می‌تواند با قطع ناگهانی (که اصطلاحا به این کار Preempt کردن گفته می‌شود) از ناحیه بحرانی خارج شود. این روش انعطاف‌پذیر است و امکان اجرای همزمان چند فرآیند را فراهم می‌آورد. این روش در سیستم‌های Time-Sharing مانند سیستم‌های عامل دسکتاپ (Windows Linux macOS) که نیاز به مدیریت هم‌زمان چندین برنامه را دارند، بسیار مناسب است. به دلیل نیاز به پاسخ‌دهی سریع به تعاملات کاربر و اجرای همزمان برنامه‌ها، این سیستم‌ها از روش پیش‌دستانه بهره می‌برند تا اطمینان حاصل شود که هیچ فرآیندی به طور نامحدود در ناحیه بحرانی باقی نمی‌ماند.

۲. روش None-Preemptive فرآیند پس از ورود به ناحیه بحرانی بدون امکان قطع توسط سیستم عامل تا پایان کارش در ناحیه بحرانی باقی می‌ماند. در این روش، کنترل به فرآیند دیگری منتقل نمی‌شود مگر اینکه فرآیند به‌طور کامل کار خود را به پایان رسانده و ناحیه بحرانی را ترک کند. این روش برای سیستم‌هایی که نیاز به کنترل دقیق در دسترسی به منابع مشترک دارند، مناسب است. این روش در سیستم‌های Real-Time که به زمان‌بندی دقیق و پیش‌بینی‌پذیر نیاز دارند، استفاده می‌شود، مانند سیستم‌های کنترل صنعتی یا سیستم‌های کنترل پرواز. در این سیستم‌ها، پیش‌بینی‌پذیری اهمیت بالایی دارد و قطع شدن فرآیندها در حین اجرای ناحیه بحرانی ممکن است به نتایج ناخواسته و خطرناک منجر شود.

سوال سوم

در رابطه با نواحی بحرانی به سوالات زیر پاسخ دهید.

۱. دستورات Atomic به چه دستوراتی گفته می‌شود؟

پاسخ

۲. دو مورد از برتری‌های استفاده از Semaphore به‌جای Mutex را توضیح دهید.

پاسخ

۳. الگوریتم پترسون را برای پشتیبانی از N پردازنده بازنویسی کنید و سپس برقراری سه شرط Mutual exclusion وProgress و Bounded waiting را در الگوریتم خود بررسی کنید.

پاسخ

سوال چهارم

دو پردازنده برای حل مسائل ناحیه بحرانی از روش‌های زیر استفاده کرده‌اند (متغیرهای L1 و L2 در هر دو مشترک هستند و مقدار Boolean دارند و در ابتدا به صورت تصادفی مقداردهی شده‌اند). هر کدام از سه شرط Mutual Exclusion و Progress و Bounded Waiting را بررسی کنید و توضیح دهید.

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

Listing 2: Code of Q4

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

Listing 1: Code of Q4

پاسخ

سوال پنجم

کلاس زیر که پیاده‌سازی سمافور است را کامل کنید و توضیح دهید هر بخش از کد که اضافه می‌کنید چگونه به حفظ سه شرط Mutual Exclusion و Progress و Bounded Waiting کمک می‌کند (فرض کنید که کلاس Process دو متد block و wakeup دارد).

```
class Semaphore
{
    queue :Queue<Process>
    // other class Properties

    constructor Semaphore(initialValue: int){
    }

    wait(process: Process){
    }

    signal(){
    }
}
```

Listing 3: Code of Q5

پاسخ