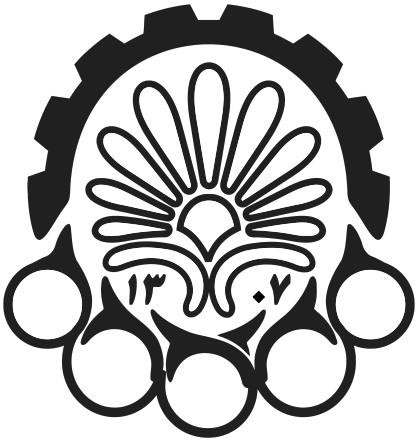
سیستم‌های عامل

دکتر زرندی



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری ششم

۱۹ آبان ۱۴۰۳

سیستم‌های عامل

تمرین سری ششم

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

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:

```
while (true) {
    entry section
        critical section
    exit section
        remainder section
}
```

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 به 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

به دستوراتی گفته می‌شود که به صورت کامل و بدون وقفه توسط فرآیند اجرا می‌شوند و نمی‌توان آن‌ها را به بخش‌های کوچکتری تقسیم کرد. این دستورات یا به‌طور کامل اجرا می‌شوند یا اصلاً اجرا نمی‌شوند، بنابراین در زمان اجرای آن‌ها، هیچ فرآیند یا رشته دیگری نمی‌تواند به منابع مشترک دسترسی پیدا کند یا آن‌ها را تغییر دهد.

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

پاسخ

دو مورد از برتری‌های Semaphore نسبت به Mutex عبارت است از:

(آ) **قابلیت استفاده برای همگام‌سازی چندین فرآیند:** Semaphoreها می‌توانند به عنوان یک شمارنده استفاده شوند و به تعداد بیش از یک فرآیند اجازه دسترسی به منبع مشترک را بدهند. برای مثال، اگر یک منبع خاص ظرفیت استفاده توسط چندین فرآیند به‌طور همزمان را داشته باشد (مانند یک محدوده حافظه با دسترسی چندگانه)، Semaphoreها می‌توانند برای مدیریت این حالت استفاده شوند، اما Mutex فقط برای کنترل دسترسی یک فرآیند در هر زمان مناسب است.

(ب) **پشتیبانی از Interprocess Synchronization:** Semaphoreها می‌توانند برای همگام‌سازی بین پردازه‌ها استفاده شوند، در حالی که Mutexها معمولاً فقط در همگام‌سازی بین رشته‌ها در یک فرآیند به‌کار می‌روند. این ویژگی باعث می‌شود Semaphore برای سناریوهایی که پردازه‌های جداگانه در یک سیستم نیاز به هماهنگی دارند مناسب‌تر باشد.

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

پاسخ

برای پشتیبانی از تعداد N فرآیند، می‌توان الگوریتم پترسون را به صورت زیر بازنویسی کرد:

ادامه پاسخ

```c
bool flag[N];
int turn;

void enter_critical_section(int i)
{
    flag[i] = true;
    turn = i;
    for (int j = 0; j < N; j++)
    {
        if (j != i)
        {
            while (flag[j] && turn == i)
            {
                /* wait until it's process i's turn.  */
            }
        }
    }
}

void exit_critical_section(int i)
{
    flag[i] = false;
}
```

Listing 1: N process Peterson algorithm

۱. **شرط Mutual Exclusion**: در این الگوریتم، شرطی وجود دارد که هر فرآیند قبل از ورود به ناحیه بحرانی بررسی می‌کند. اگر هیچ فرآیند دیگری در ناحیه بحرانی نباشد، آن فرآیند اجازه ورود پیدا می‌کند. بنابراین، همیشه تنها یک فرآیند در هر لحظه می‌تواند در ناحیه بحرانی باشد.

۲. **شرط Progress**: اگر هیچ فرآیندای در ناحیه بحرانی نباشد و چندین فرآیند آماده‌ی ورود باشند، نوبت‌دهی در این الگوریتم تضمین می‌کند که یکی از فرآیندهای آماده می‌تواند وارد ناحیه بحرانی شود، به شرطی که شرایط برایش مهیا باشد. این مسئله شرط پیشرفت را برقرار می‌کند.

۳. **شرط Bounded Waiting**: این الگوریتم تضمین می‌کند که هر فرآیند پس از تعداد محدودی از چرخش‌ها در حلقه، می‌تواند وارد ناحیه بحرانی شود. با توجه به تغییر turn در حلقه، هیچ فرآیندای به طور نامحدود منتظر نمی‌ماند. بنابراین شرط انتظار محدود نیز برقرار است.

سوال چهارم

دو فرآیند 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.

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

Listing 2: Code of Q4

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

Listing 3: Code of Q4

پاسخ

۱. **بررسی شرط Mutual Exclusion:** به دلیل این شرایط متفاوت، تنها یکی از پردازه‌ها می‌تواند همزمان وارد ناحیه بحرانی شود، زیرا شرطی که برای ورود هر فرآیند وجود دارد، دقیقاً خلاف شرط دیگر فرآیند است. بنابراین، شرط Mutual Exclusion برقرار است؛ چرا که اگر یکی از فرآیندها وارد ناحیه بحرانی شود، دیگری نمی‌تواند وارد شود تا زمانی که آن فرآیند از ناحیه بحرانی خارج شود و مقادیر L_1 و L_2 تغییر یابند.

۲. **بررسی شرط Progress:** در این الگوریتم، اگر یکی از فرآیندها در حال اجرای ناحیه بحرانی باشد، فرآیند دیگر منتظر می‌ماند. اما مشکل زمانی به وجود می‌آید که مقادیر L_1 و L_2 به صورتی تنظیم شوند که هر دو فرآیند در حالت نامناسب برای ورود قرار گیرند، یعنی:

- اگر P_1 شرط ورود خود را برقرار نبیند (یعنی L1==L2) و منتظر تغییر در مقدار باشد.
- و اگر P_2 شرط ورود خود را برقرار نمی‌بیند (یعنی L1!=L2) و او هم منتظر بماند.

این وضعیت می‌تواند به Deadlock منجر شود که هر دو فرآیند به صورت نامحدود در حلقه‌های انتظار خود باقی بمانند، بدون اینکه هیچ‌کدام پیشرفتی داشته باشد. به همین دلیل، شرط پیشرفت به صورت قطعی برقرار نیست و احتمال دارد فرآیندها بدون دلیل منتظر بمانند.

۳. **بررسی شرط Bounded Waiting:** در این کد، هیچ سازوکار مشخصی برای محدود کردن زمان انتظار وجود ندارد. به عنوان مثال، اگر فرآیند P_1 وارد ناحیه بحرانی شود و پس از خروج مقدار L1 != L2 را تنظیم کند، ممکن است فرآیند P_2 در حالت انتظار باقی بماند، چرا که این مقدار به شرط او نمی‌خورد. این مسئله می‌تواند باعث شود که یکی از فرآیندها برای مدت طولانی در حالت انتظار بماند. در نتیجه، شرط انتظار محدود نیز برقرار نیست و ممکن است یک فرآیند به صورت نامحدود منتظر بماند.

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 4: Code of Q5

پاسخ

کلاس را به‌صورت زیر تکمیل می‌کنیم:

```
// Semaphore class definition
class Semaphore {
    private:
    int value; // semaphore counter
    Queue<Process> queue; // queue to store waiting processes

    public:
    // Constructor to set the initial value of the semaphore
    Semaphore(int initialValue) {
        value = initialValue;
    }

    // wait method for entering the critical section
    void wait(Process process) {
        value--; // decrement the semaphore value
        if (value < 0) {
            queue.enqueue(process); // add the process to the queue
            process.block(); // block the process
        }
    }

    // signal method for exiting the critical section
    void signal() {
        value++; // increment the semaphore value
```

```
25          if (value <= 0) {
26              Process nextProcess = queue.dequeue(); // dequeue a process
27              nextProcess.wakeup(); // wake up the process
28          }
29      }
30  };
```

Listing 5: Complete code of Q5

ادامه پاسخ

- متغیر `value`: این متغیر شمارنده‌ای است که مقدار سمافور را نگه می‌دارد. اگر مقدار آن منفی شود، فرآیندها باید منتظر بمانند، که باعث برقراری شرط Mutual Exclusion می‌شود.
- سازنده `Semaphore(int initialValue)`: این سازنده مقدار اولیه سمافور را تنظیم می‌کند. این مقدار اولیه به شرط Mutual Exclusion کمک می‌کند، زیرا برای سمافور باینری تنها یک فرآیند در هر زمان می‌تواند به ناحیه بحرانی دسترسی داشته باشد.
- متد `wait(Process process)`: در این متد، اگر مقدار `value` منفی شود، فرآیند به صف اضافه شده و مسدود می‌شود. این عمل به شرط Bounded Waiting کمک می‌کند، زیرا فرآیندها به ترتیب ورود در صف قرار گرفته و به نوبت از صف خارج می‌شوند.
- متد `signal()`: این متد مقدار `value` را افزایش می‌دهد و اگر فرآیندهایی در صف منتظر باشند، فرآیند بعدی را بیدار کرده و اجازه ورود می‌دهد. این امر باعث برقراری شرط Progress و Bounded Waiting می‌شود.

این پیاده‌سازی سه شرط لازم برای مدیریت نواحی بحرانی را به‌طور کامل فراهم می‌کند و با استفاده از صف و کنترل متغیر `value`، به حفظ همگام‌سازی بین فرآیندها کمک می‌کند.