به نام خدا

درس: سیستم‌های عامل ۱۴۰۳۲ (۴۰۴۲۴۱)
مدرس: دکتر رسول جلیلی – ۱۴۰۴/۰۲/۳۰

تمرین شماره ۴

دانشکده مهندسی کامپیوتر

۱- راه حل پیترسون[۱] (مربوط به حل مسئله ناحیه بحرانی[۲] بین دو پردازه با استفاده از متغیرهای اشتراکی flag و turn) را در نظر بگیرید.

الف) چگونگی تأثیر بازآرایی دستورالعمل‌ها[۳] (در پردازنده‌ها یا کامپایلرهای نوین)، بر شکست این راه‌حل در حفظ انحصار متقابل را شرح دهید. یک مثال خاص از حالتی که درهم‌آمیختگی[۴] دستورالعمل‌ها به‌وضوح منجر به این شکست خواهد شد، ارائه کنید. (راهنمایی: شکل ۶.۴ کتاب مرجع)

ب) باتوجه‌به مدل‌های حافظه[۵] که به دو نوع مرتب قوی[۶] و مرتب ضعیف[۷] تقسیم می‌شوند، تعیین کنید که مشکل مطرح شده در قسمت (الف)، در کدام نوع از این مدل‌های حافظه محتمل‌تر است؟ چرا؟

ج) سازوکار سخت‌افزاری موانع حافظه[۸] یا حصارهای حافظه[۹] را شرح دهید. این سازوکار چگونه تضمین می‌کند که تغییرات انجام شده در حافظه، توسط سایر پردازنده‌ها به‌درستی و با ترتیب مورد انتظار رویت شوند؟

د) چگونه اعمال یک سد حافظه می‌تواند به راه‌حل پیترسون کمک کند تا در معماری‌های نوین (که دستورالعمل‌ها را بازآرایی می‌کنند)، به‌درستی کار کند؟ (محل مناسب برای قراردادن سد حافظه در راه‌حل پیترسون را مشخص کنید.)

۲- متغیرهای شرطی[۱۰] ابزارهای قدرتمندی برای همگام‌سازی[۱۱] ریسه‌ها هستند، که به آن‌ها اجازه می‌دهند تا زمانی که یک شرط خاص برقرار نشده است، منتظر بمانند. بااین‌حال، استفاده دقیق از آن‌ها برای جلوگیری از رفتارهای غیرمنتظره و تضمین صحت برنامه ضروری است. در اکثر پیاده‌سازی‌های استاندارد متغیرهای شرطی، اکیداً توصیه می‌شود که بررسی شرط مورد انتظار در یک حلقه while انجام شود، و نه با یک دستور if ساده. به عبارت دیگر، الگوی صحیح به صورت زیر خواهد بود.

```
pthread_mutex_lock(&mutex);
while (condition_is_not_met) {
        pthread_cond_wait(&cond, &mutex);
}
pthread_mutex_unlock(&mutex);
```

دلایل اصلی این توصیه را نام برده و توجیه کنید. (راهنمایی: مفاهیمی همچون "بیدارشدن‌های کاذب[۱۲]" و "انتظار هم‌زمان چندین ریسه برای یک شرط".)

---

[1] Peterson
[2] Critical Section Problem
[3] Instruction Reordering
[4] Interleaving
[5] Memory Models
[6] Strongly Ordered
[7] Weakly Ordered
[8] Memory Barriers
[9] Memory Fences
[10] Condition Variables
[11] Synchronization
[12] Spurious Wakeups

۳- مسئله خوانندگان-نویسندگان[13] را در نظر بگیرید، با این تفاوت که در این مسئله اولویت با نویسندگان خواهد بود. در این شرایط، به محض اینکه یک نویسنده تمایل خود را برای دسترسی به منبع اشتراکی اعلام می‌کند، هیچ خواننده جدیدی که پس از نویسنده وارد سیستم شده نباید اجازه شروع خواندن را تا زمانی که آن نویسنده و سایر نویسندگان منتظر کار خود را به اتمام برسانند پیدا کند. با استفاده از سمافورها و هر متغیرکمکی لازم، شبه‌کد[14] مربوط به توابع یا رویه‌های مربوط به خوانندگان و نویسندگان را بنویسید.

۴- در هر مورد از قطعه کدهای زیر، یک راهکار برای حل مسئله ناحیه بحرانی ارائه شده است. برای هر یک از موارد زیر با ارائه دلیل، مشخص کنید که آیا راهکار پیشنهادی مسئله بخش بحرانی را به درستی حل خواهد کرد؟ در صورتی که جواب منفی است برای حل این موضوع راهکاری پیشنهاد کنید.

الف)

```
proc (int i) {
while (TRUE) {
        compute;
        while (turn ! = i)
                ;
        Critical_section;
        turn = (i + 1) % 2;
}
}
Shared int turn;
turn = 1;
fork (proc, 1, 0);
fork (proc, 1, 1);
```

ب)

```
proc(int i) {
        while (TRUE) {
                compute;
                while (flag[(i + 1)% 2])
                        ;
                flag[i] = TRUE;
                Critical_section ;
                flag[i] = FALSE;
        }
}
Shared boolean flag[2];
flag[0] = flag[1] = FALSE;
fork(proc, 1, 0);
fork(proc, 1, 1);
```

[13] Readers-Writers Problem
[14] Pseudocode

ج)

```
proc(int i) {
        while (TRUE) {
                compute;
                flag[i] = TRUE;
                while (flag[(i + 1)% 2])
                        ;
                Critical_section;
                flag[i] = FALSE;
        }
}
Shared boolean flag[2];
flag[0] = flag[1] = FALSE;
fork(proc, 1, 0);
fork(proc, 1, 1);
```

۵- با بررسی حالات مختلف متصور برای اجرای دو پردازه‌ی P_1 و P_2، شرح دهید که کدام یک از سه شرط (۱) انحصار متقابل، (۲) پیشرفت، و (۳) انتظار محدود در اجرای این پردازه‌ها رعایت می‌شود.

```
N1, N2 : integer := 0;

task body P1 is:
begin
  loop
    Non_critical_section;
    N1 := 1;
    N1 := N2 + 1;
    while (N2 != 0 &&N1 > N2);
    Critical_section;
  end loop;
end P1;

task body P2 is:
begin
  loop
    Non_critical_section;
    N2 := 1;
    N2 := N1 + 1;
    while (N1 != 0 && N2 >= N1);
    Critical_section;
  end loop;
end P2;
```

۶-

الف) سازوکار قفل چرخشی[15] را در نظر بگیرید. این سازوکار برای چه نوع سیستم‌هایی (تک پردازنده‌ای یا چند پردازنده‌ای) و تحت چه شرایطی مدت‌زمان کوتاه یا طولانی نگه‌داشتن قفل) مناسب‌تر است؟ پاسخ خود را با ارائه دلیل، توجیه کنید.

[15] Spinlock

ب) در سیستم‌عامل لینوکس، در ماشین‌های تک پردازنده‌ای، به‌جای استفاده از قفل چرخشی واقعی، مداخله‌جویی در سطح هسته[۱۶] غیرفعال و سپس مجدداً فعال خواهد شد. چرا این رویکرد در یک سیستم تک پردازنده‌ای معادل عملکرد قفل چرخشی در یک سیستم چندپردازنده‌ای خواهد بود؟ چرا استفاده از قفل چرخشی واقعی (که شامل چرخش در یک حلقه است) در یک سیستم تک پردازنده‌ای ناکارآمد خواهد بود؟

۷- اولین راه‌حل نرم‌افزاری صحیح شناخته شده برای مسئله ناحیه بحرانی برای دو پردازه، توسط دِکِر[۱۷] توسعه داده شده است. این الگوریتم برای دو پردازه P_0 و P_1 طراحی شده است که متغیرهای زیر را به اشتراک می‌گذارند.

```
boolean flag[2]; //initially false
int turn;
```

همچنین، بر اساس این راهکار، ساختار کلی هر پردازه P_i به صورت زیر است.

```
while (true) {
  flag[i] = true;
  while (flag[j]) {
    if (turn == j) {
      flag[i] = false;
      while (turn == j) {
        //do nothing (busy wait)
      }
      flag[i] = true;
    }
  }
  //Critical Section
  turn = j;
  flag[i] = false;
  //Remainder Section
}
```

رعایت سه شرط انحصار متقابل، پیشرفت و انتظار محدود را در اجرای این پردازه‌ها بررسی کنید.

۸- یک کارگزار وب[۱۸] چندریسه‌ای می‌خواهد تعداد درخواست‌هایی را که به آن‌ها خدمت‌رسانی می‌کند[۱۹]، پیگیری کند. دو راهبرد زیر برای جلوگیری از بروز وضعیت رقابتی[۲۰] بر روی متغیر hits را در نظر بگیرید.

(۱) راهبرد اول، استفاده از یک قفل انحصار متقابل[۲۱] پایه هنگام به‌روزرسانی hits است:

```
int hits;
mutex_lock hit_lock;

hit_lock.acquire();
hits++;
hit_lock.release();
```

[16] Kernel-level Preemption
[17] Dekker
[18] Web Server
[19] Hit Count
[20] Race Condition
[21] Mutex Lock

(۲) راهبرد دوم، استفاده از یک عدد صحیح اتمی[22] است:

$$atomic\_t\ hits;$$
$$atomic\_inc(\&hits);$$

این دو راهبرد را از دیدگاه کارآمد بودن با یکدیگر مقایسه کنید.

[22] Atomic Integer