به نام خدا

دانشکده مهندسی کامپیوتر

درس:
سیستم‌های عامل ۱۴۰۳۲ (۴۰۴۲۴۱)
مدرس:
دکتر رسول جلیلی – ۱۴۰۴/۰۳/۰۶

پاسخنامه آزمونک شماره ۴

۱- مسئله خوانندگان-نویسندگان با اولویت نویسندگان را در نظر بگیرید. در این حالت، به محض اینکه یک نویسنده تمایل خود را برای دسترسی به منبع اشتراکی اعلام می‌کند، هیچ خواننده جدیدی نباید اجازه شروع خواندن را پیدا کند تا زمانی که آن نویسنده (و سایر نویسندگان منتظر) کار خود را به اتمام برسانند. شبه‌برنامه[1] زیر را با استفاده از مفاهیم نشان‌بر[2] و متغیرهای کمکی تکمیل کنید تا این سیاست اولویت‌بندی به درستی پیاده‌سازی شود. جاهای خالی (که با نقطه‌چین مشخص شده‌اند) را با کد یا عبارت مناسب پر کنید. (۶ نمره) هر مورد ۰.۵ نمره

```
// متغیرهای اشتراکی
semaphore rw_mutex = 1;  // for shared resource access
semaphore mutex = 1;     // for protecting read_count & write_count
semaphore read_try = 1;
int read_count = 0;
int write_count = 0;
```

```
// پردازه نویسنده
void writer() {
   while (true) {
      wait(mutex);
      write_count++;
      if (write_count == 1)
         wait(read_try);
      signal(mutex);

      wait(rw_mutex);

      // --- ناحیه بحرانی : Writing is performed ---

      signal(rw_mutex);

      wait(mutex);
      write_count--;
      if (write_count == 0)
         signal(read_try);
      signal(mutex); }
}
```

```
// پردازه خواننده
void reader() {
   while (true) {
      wait(read_try);          # ترتیب حائز اهمیت است

      wait(mutex);
      read_count++;
      if (read_count == 1)
         wait(rw_mutex);
      signal(mutex);
      signal(read_try);

      // --- ناحیه بحرانی : Reading is performed ---

      wait(mutex);
      read_count--;
      if (read_count == 0)
         signal(rw_mutex);
      signal(mutex); }
}
```

[1] Pseudocode
[2] Semaphore

۲-

الف) چرا استفاده از قفل‌های چرخشی[3] در سامانه‌های تک‌پردازنده‌ای نامناسب است اما ممکن است استفاده از آن‌ها در سامانه‌های چندپردازنده‌ای کارآمد باشد؟ (۳ نمره امتیازی)

اسپین‌لاک باعث می‌شود یک ریسه در یک حلقه به صورت مداوم بچرخد و پردازنده را اشغال کند تا قفل آزاد شود. در یک سیستم با یک پردازنده، اگر ریسه A در حال چرخش برای قفلی باشد که در دست ریسه B است، ریسه B هرگز فرصت اجرا شدن پیدا نمی‌کند (۱ نمره) تا قفل را آزاد کند، زیرا پردازنده توسط ریسه A اشغال شده است. این وضعیت منجر به اتلاف کامل چرخه پردازنده می‌شود و تنها پس از یک تعویض بافتار زمان‌بر، ممکن است ریسه B اجرا شود (۱ نمره). بنابراین، اسپین‌لاک در اینجا شدیداً ناکارآمد است.

در یک سیستم با چندین پردازنده، یک ریسه می‌تواند روی یک هسته در حال چرخش باشد در حالی که ریسه دیگری که قفل را در اختیار دارد، به صورت همزمان روی هسته دیگری در حال اجراست. اگر قرار باشد قفل برای مدت کوتاهی نگه داشته شود، زمان صرف شده برای چرخش ممکن است بسیار کمتر از سربار دو تعویض بافتار (یکی برای به خواب بردن ریسه و دیگری برای بیدار کردن آن) باشد (۱ نمره). بنابراین، برای قفل‌های کوتاه‌مدت، اسپین‌لاک یک راهکار بسیار کارآمد است.

ب) در پیاده‌سازی نشان‌برها، چگونه می‌توان تفکیک‌ناپذیر[4] بودن عملیات wait() و signal() را در یک سامانه چندپردازنده‌ای متقارن[5] تضمین کرد؟ چرا غیرفعال‌کردن وقفه‌ها[6] به‌تنهایی کافی نیست؟ (۲ نمره امتیازی)

عملیات wait() و signal() خود شامل یک بخش بحرانی هستند (مانند دستکاری مقدار سمافور و لیست انتظار). برای تضمین اتمی بودن این عملیات در سیستم‌های چندپردازنده‌ای، باید از راهکارهای جایگزینی استفاده کرد که سخت‌افزار از آن‌ها پشتیبانی می‌کند. رایج‌ترین این راهکارها استفاده از اسپین‌لاک‌های داخلی یا دستورالعمل‌های اتمی سخت‌افزاری مانند compare_and_swap() است (۱ نمره). این ابزارها تضمین می‌کنند که در هر لحظه فقط یک هسته می‌تواند کد داخلی wait() یا signal() را اجرا کند.

غیرفعال کردن وقفه تنها بر روی هسته‌ای که دستور را اجرا می‌کند تأثیر دارد (۱ نمره). در یک سیستم چندپردازنده‌ای، اگر ریسه A روی هسته ۱ وقفه‌ها را غیرفعال کند و شروع به تغییر سمافور کند، ریسه B روی هسته ۲ کاملاً بدون تأثیر باقی می‌ماند و می‌تواند همزمان برای دسترسی به همان سمافور تلاش کند که منجر به بروز وضعیت رقابتی[7] می‌شود. برای اینکه این روش کار کند، باید وقفه‌ها روی تمام هسته‌ها غیرفعال شوند که این کار فرآیندی کند و پیچیده است و کارایی سیستم را به شدت کاهش می‌دهد.

---

[3] Spinlocks
[4] Atomic
[5] Symmetric Multi-Processor (SMP)
[6] Disabling Interrupts
[7] Race Condition

۳- در هر مورد از قطعه برنامه‌های زیر، یک راهکار برای حل مسئله ناحیه بحرانی ارائه شده است. برای هر یک از این موارد، با ذکر دلیل مشخص کنید که آیا راهکار پیشنهادی، مسئله ناحیه بحرانی را به درستی حل می‌کند؟ در صورتی که جواب منفی است، مشخص کنید که کدام یک از شروط انحصار متقابل[8]، پیش‌روی[9]، یا انتظار محدود[10] رعایت نخواهد شد؟ یک مثال از حالتی که شرط انتخاب شده را نقض می‌کند، به عنوان دلیل ذکر کنید. (۷ نمره)

الف)

```
// متغیر اشتراکی
boolean flag[2] = {false, false};

// پردازه Pi
while (true) {
   while (flag[1-i]) ; // Wait if other process wants to enter
   flag[i] = true;
   // ناحیه بحرانی
   flag[i] = false; }
```

ب)

```
// متغیر اشتراکی
int turn = 0;

// پردازه Pi
while (true) {
   while (turn != i) ; // Busy wait for its turn
     // ناحیه بحرانی
   turn = 1 - i; }
```

الف) خیر (۰.۵ نمره) - انحصار متقابل (۱ نمره) - فرض کنید P0 و P1 هر دو می‌خواهند وارد ناحیه بحرانی شوند و در ابتدا flag[0] و flag[1] هر دو false هستند. P0 اجرا می‌شود: P0 شرط while (flag[1]) را بررسی می‌کند. چون flag[1] برابر false است، از حلقه خارج می‌شود. تعویض بافتار: دقیقاً در همین لحظه، قبل از اینکه P0 بتواند دستور flag[0] = true را اجرا کند، سیستم‌عامل پردازه را به P1 تغییر می‌دهد. P1 اجرا می‌شود: P1 شرط while (flag[0]) را بررسی می‌کند. چون P0 هنوز flag[0] را true نکرده، این شرط نیز false است و P1 از حلقه خارج می‌شود. P1 دستور flag[1] = true را اجرا کرده و وارد بخش بحرانی خود می‌شود. تعویض بافتار: حال پردازه به P0 برمی‌گردد. P0 ادامه می‌دهد: P0 از همان نقطه‌ای که متوقف شده بود، ادامه می‌دهد و دستور flag[0] = true را اجرا کرده و وارد بخش بحرانی خود می‌شود. (۲ نمره)

ب) خیر (۰.۵ نمره) – پیش‌روی (۱ نمره) - فرض کنید turn = 0 است. P0 اجرا می‌شود، شرط while (turn != 0) را false می‌یابد و وارد بخش بحرانی می‌شود. P0 از بخش بحرانی خارج شده و turn را برابر ۱ قرار می‌دهد (turn = 1 - 0). P0 بلافاصله دوباره می‌خواهد وارد ناحیه بحرانی شود. این بار شرط while (turn != 0) را بررسی می‌کند. چون turn برابر ۱ است، P0 در این حلقه گیر می‌کند و منتظر می‌ماند. در همین حین، P1 در بخش باقی‌مانده خود در حال اجرای کدی طولانی است و اصلاً قصد ورود به ناحیه بحرانی را ندارد. (۲ نمره)

۴- با استفاده از دستورالعمل تفکیک‌ناپذیر 'int compare_and_swap(int *value, int expected, int new_value)'، یک قفل چرخشی ساده پیاده‌سازی نمائید. شبه‌برنامه توابع acquire() و release() را برای این قفل چرخشی بنویسید. (فرض کنید یک متغیر اشتراکی int lock با مقدار اولیه صفر وجود دارد. مقدار صفر برای متغیر lock، به معنای آزاد بودن قفل و مقدار ۱ به معنای دردسترس‌نبودن آن است.) (۴ نمره)

```
void acquire(int *lock) {

   while(compare_and_swap(lock, 0, 1) != 0);

}
```

```
void release(int *lock) {

   *lock=0;

}
```

[8] Mutual exclusion
[9] Progress
[10] Bounded waiting