پاسخ آزمون میانی درس سیستم‌های عامل

۱ سیستم عاملی که در یک ماشین مجازی اجرا می‌شود، به سخت‌افزار مستقیما دسترسی ندارد. پردازه‌های یک سیستم عامل نیز به صورت مستقیم به سخت‌افزار و منابع سیستم دسترسی ندارند و با استفاده از فراخوانی‌های سیستمی می‌توانند از سیستم عامل درخواست کنند تا عملیات مورد نظرشان روی سخت‌افزار انجام شود.

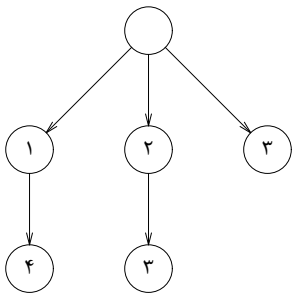
۲ «Core dump» محتویات حافظه‌ی یک پردازه (و وضعیت Register-های پردازنده) در یک لحظه از اجرای آن پردازه می‌باشد، که سیستم عامل آن را در یک فایل نوشته است. با بررسی آن توسط یک Debugger، برنامه-نویس می‌تواند با مشاهده‌ی حافظه‌ی پردازه (برای مثال مقدار متغیرها و وضعیت پشته) در زمان خطا مشکلات ممکن در برنامه را شناسایی نماید.

۳ الف) بسیاری از بخش‌های سیستم عامل در پردازه‌های فضای کاربر پیاده‌سازی می‌شوند و تغییر یا اضافه کردن بخش‌هایی به سیستم عامل (مثلا یک راه‌انداز) نیازی به تغییر و به روزی رسانی هسته ندارد. ب) با توجه به اینکه قسمت زیادی از سیستم عامل در پردازه‌های فضای کاربر اجرا می‌شوند، در صورت وجود اشکال در یکی از این پردازه‌ها، با توجه به جدا بودن فضای آدرس پردازه‌ها مشکلی برای سایر قسمت‌های سیستم عامل ایجاد نمی‌شود. ج) چون بخش‌های مختلف سیستم عامل در پردازه‌های متفاوتی پیاده‌سازی شده‌اند، این پردازه‌ها با استفاده از تبادل پیغام (Message passing) با هم ارتباط دارند که سربار قابل توجهی دارد.

در سیستم‌های عاملی که از ماژول‌ها استفاده می‌کنند الف) مشابه میکروکرنل‌ها بخش‌های زیادی از سیستم عامل در ماژول‌ها پیاده‌سازی می‌شوند که می‌توانند بدون نیاز به تغییر هسته، تغییر داده شوند یا اضافه شوند. ب) بر خلاف میکروکرنل‌ها، چون ماژول‌ها فضای آدرس مجزایی ندارند، در صورت وجود اشکال در یک بخش از سیستم عامل، کل سیستم عامل تحت تأثیر قرار می‌گیرد. ج) چون قسمت‌های متفاوت سیستم عامل به صورت مستقیم با هم ارتباط دارند، سربارهای موجود در میکروکرنل‌ها در آنها وجود ندارد.

۴ قطعه‌ی به نام Timer در بازه‌هایی که سیستم عامل مشخص می‌کند وقفه‌ای ایجاد می‌کند که باعث می‌شود پردازنده از فضای کاربر (User space) به فضای هسته (Kernel space) انتقال یابد. سپس سیستم عامل می‌تواند در انتقال به فضای کاربر، پردازه‌ی دیگری را برای اجرا انتخاب نماید.

۵ اعداد داخل رأس هر پردازه، شماره‌ی فراخوانی ()fork-ای است (با توجه به ترتیب ظاهر شدن آنها در شبه کد) که موجب ساخته شدن آن شده است.



۶ در صورتی که بافر (Buffer) لوله پر شود، پردازه‌ی نویسنده (تا زمانی که پردازنده‌ی خواننده بخشی از داده‌های لوله را بخواند) در حالت Blocking قرار می‌گیرد. در صورتی که داده‌ای در بافر لوله نباشد، پردازه‌ی خواننده (تا زمانی که پردازه‌ی نویسنده داده‌ای را در لوله بنویسد) در حالت Blocking قرار می‌گیرد.

۷ الف) حداکثر تعداد ریسمان‌های استفاده شده را محدود می‌کند که مانع از اتمام منابع سیستم عامل با وجود تعداد زیادی وظیفه‌ی همزمان می‌شود. ب) با توجه به اینکه تعدادی ریسمان یک بار در ابتدا ساخته می‌شوند، لازم نیست برای هر وظیفه ریسمان مجزایی ساخته شود و سربار ساختن ریسمان‌ها کاهش می‌یابد. ج) با توجه به اینکه کاربر به صورت مستقیم ریسمان‌ها را مدیریت نمی‌کند، مدیریت اجرای ریسمان‌ها برای کاربر آسان‌تر می‌شود.

۸ اول آنکه با استفاده از ریسمان‌های متفاوت برای بخش اول، در صورتی که چند پردازنده موجود باشند، عمل رمزنگاری به صورت موازی انجام می‌شود. دوم آنکه، برای انتقال داده‌ها از شبکه، عمل انتقال می‌تواند به صورت موازی انجام شود و Block شدن یکی از این ریسمان‌ها (برای ارسال فایل از شبکه) موجب توقف انتقال سایر فایل‌ها و رمزنگاری آنها نمی‌گردد.

۹ فرض کنید مقدار اولیه‌ی دو متغیر سراسری صفر باشد و تابع found() توسط دو ریسمان (ریسمان اول و دوم) به ترتیب به صورت found(1, 3) و found(1, 2) صدا زده شود. اگر ابتدا تا خط ۳ در ریسمان اول اجرا شود و سپس ریسمان دوم کل تابع را اجرا کند و پس از آن ریسمان اول اجرای خود را خاتمه دهد مقدار best_cost دو خواهد شد، در صورتی که مقدار درست آن سه بوده است.

```
1   void found(int node, int cost)
2   {
3         if (cost < best_cost) {
4               best_cost = cost;
5               best_node = node;
6         }
7   }
```

با تعریف یک Mutex با نام lock و فراخوانی acquire(lock) قبل از خط ۳ و فراخوانی release(lock) پس از خط ۶ مشکل حل می‌شود.

۱۰ در صورتی که نویسنده‌ای در حال نوشتن باشد و خواننده‌ای در حال انتظار باشد، پس از اتمام کار نویسنده، منبع به خواننده داده شود (حتی اگر نویسنده‌ی دیگری منتظر باشد). در صورتی که تعدادی خواننده در حال خواندن باشند و نویسنده‌ای منتظر باشد، خواننده‌ی دیگری نمی‌تواند از منبع بخواند و پس از اتمام کار خواننده‌هایی که در حال خواندن هستند، نویسنده از منبع استفاده خواهد کرد.

برای پیاده‌سازی یک مانیتور برای این مسئله، می‌توان دو متغیر Condition، یکی برای نویسنده‌ها و یکی برای خواننده‌ها در نظر گرفت. علاوه بر آن، چهار متغیر برای نگهداری تعداد ریسمان‌های در حال خواندن از منبع، تعداد ریسمان‌های منتظر برای خواندن، تعداد ریسمان‌های در حال نوشتن به منبع (حداکثر یک) و تعداد ریسمان‌های منتظر برای نوشتن به مانیتور اضافه کرد. با توجه به مقدار این متغیرها، می‌توان تصمیم گرفت که در پیاده‌سازی read_lock() و write_lock() آیا یک ریسمان باید با فراخوانی تابع wait() یکی از متغیرهای Condition منتظر بماند یا خیر. همچنین در read_unlock() و write_unlock() ریسمان‌های مناسب باید از حالت انتظار خارج شوند.

```
Monitor StarvationFreeRWLock {
    condition rcond;         // for waiting readers
    condition wcond;         // for waiting writers
    int rwaiting, wwaiting; // number of waiting readers and writers
```

```
    int rcount, wcount;      // number of current readers and writers
    read_lock() {
        if (wwaiting + wcount > 0) {
            rwaiting++;
            rcond.wait();
            rwaiting--;
        }
        rcount++;
    }
    write_lock() {
        if (wcount + rcount > 0) {
            wwaiting++;
            wcond.wait();
            wwaiting--;
        }
        wcount++;
    }
    read_unlock() {
        rcount--;
        if (rcount == 0 && wwaiting > 0)
            wcond.signal();
    }
    write_unlock() {
        wcount--;
        if (rwaiting > 0) {
            int i, n = rwaiting;
            for (i = 0; i < n; i++)
                rcond.signal();
        } else {
            wcond.signal();
        }
    }
}
```