دانشکده‌ی مهندسی کامپیوتر
شماره دانشجویی: ۹۹۱۰۵۵۶۱
شایان صالحی

سیستم‌های عامل

نیم‌سال اول ۰۳-۰۲

تمرین سوم

پاسخ مسئله‌ی ۱.

الف

در اینجا می‌دانیم که برنامه a می‌تواند y و z را لاک کند همچنین برنامه b توانایی لاک کردن x و z و در نهایت برنامه c توانایی لاک کردن x و y را دارد.

برای آنکه بتوانیم حالتی را معرفی کنیم که تردها دچار Deadlock شوند. می‌توانیم ترتیبی به این صورت داشته باشیم:

$$a \to lock(y), \quad b \to lock(z), \quad c \to lock(x)$$

به این ترتیب تمامی قسمت‌ها لاک شده و برنامه توانایی پیش‌روی را نخواهد داشت.

ب

با تغییر ترتیب mutex-unlock ها در اینجا نمی‌توان جلوی رخ دادن Deadlock را گرفت زیرا که در قسمت الف، برنامه در خط اول لاک می‌شود و ترتیب اجرای دستورات mutex-unlock اهمیتی نخواهند داشت و برنامه هیچ‌گاه به آنلاک نمی‌رسد.

ج

از آنجایی که در زمان‌بندی first come first served یک پردازه تا آخر انجام می‌شود و سپس به پردازه بعدی می‌رویم. هر پردازه در زمان اجرای خود دو واحد مموری را لاک کرده و سپس آنلاک می‌کند که به این ترتیب هیچ‌گاه به مشکل Deadlock نخواهیم خورد.

د

برای آنکه بتوانیم توالی ارائه دهیم که بتوان این مشکل را برطرف کرد می‌توان برنامه را از حالتی که هرکدام یک واحد جداگانه را لاک کند دربیاورویم.

برای اینکار کافیست در پردازه a ترتیب لاک کردن y و z را عوض کنیم. به این ترتیب هر دو پردازه a و b در ابتدا تلاش می‌کند که z را لاک کنند و یکی موفق می‌شوند. حال دو حالت داریم:

- اگر پردازه c هر دو x و y را لاک کرده باشد. آنگاه پس از اجرای آن هر دو آنلاک شده و پردازه c کار خود را انجام داده و دو پردازه دیگر نیز می‌توانند بدون مشکل کار خود را به اتمام برسانند.
- x و y لاک نشده باشند. به این ترتیب پردازه a کار خود را اتمام کرده و y و z را آزاد می‌کنند. سپس b می‌تواند به انتهای خود برسد و درنهایت پس از اتمام کار این دو پردازه c بدون هیچ اشکالی می‌تواند ران شود.

به این ترتیب Deadlock در این حالت برطرف شده است.

پاسخ مسئله‌ی ۲.

برای آنکه بتوانیم مطمئن شویم گراف داده شده از توالی پردازه‌ها به درستی رعایت می‌شوند می‌توانیم از دو سمافور S_1 و S_2 استفاده کنیم. به این صورت که از سمافور اولی برای ترتیب توالی پردازه T_1 با T_2 و T_3 استفاده می‌کنیم و از سمافور دومی برای پردازه‌های منتی به T_4 استفاده خواهیم کرد.

به این ترتیب داریم:

```
void T_1 () {
    T_1 Running;
    Signal(S_1);
    Signal(S_1);
}

void T_2 () {
    Wait(S_1);
    T_2 Running;
    Signal(S_2);
}

void T_3 () {
    Wait(S_1);
    T_3 Running;
    Signal(S_2);
}

void T_3 () {
    Wait(S_2);
    T_4 Running;
}
```

در اینجا برای مقدار اولیه S_1 و S_2 داریم:

$$Value(S_1) = 0, \qquad Value(S_2) = -1$$

دلیل اینکار برای این است که پرادازه T_4 باید منتظر بماند که هردو پردازه T_3 و T_2 کار خود را به اتمام برسانند و تابع `Signal(S_2)` را صدا بزنند.

به صورت کلی دو پردازه دوم و سوم در `Wait()` S_1 خواهند ماند تا زمانی که پردازه اول کار خود را به اتمام برسانند و دوباره `Signal(S_1)` را صدا بزند تا مقدار آن برابر دو شود و پردازه‌های بعدی به اجرا در بیایند.

پاسخ مسئله‌ی ۳.

پاسخ مسئله‌ی ۴.

پاسخ مسئله‌ی ۵.

پاسخ مسئله‌ی ۶.

پاسخ مسئله‌ی ۷.

پاسخ مسئله‌ی ۸.