



## 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:

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

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 پردازش تا آخر انجام می‌شود و سپس به پردازش بعدی می‌رویم. هر پردازش 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 دربیاریم.

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

- اگر پردازش  $c$  هر دو  $x$  و  $y$  را لاک کرده باشد. آنگاه پس از اجرای آن هر دو آنلاک شده و پردازش  $c$  کار خود را انجام داده و دو پردازش دیگر نیز می‌توانند بدون مشکل کار خود را به اتمام برسانند.

- $x$  و  $y$  لاک نشده باشند. به این ترتیب پردازش  $a$  کار خود را اتمام کرده و  $y$  و  $z$  را آزاد می‌کنند. سپس  $b$  می‌تواند به انتهای خود برسد و در نهایت پس از اتمام کار این دو پردازش  $c$  بدون هیچ اشکالی می‌تواند ران شود.

به این ترتیب Deadlock در این حالت برطرف شده است.

پاسخ مسئله‌ی ۲.

پاسخ مسئله‌ی ۳.

پاسخ مسئله‌ی ۴.

پاسخ مسئله‌ی ۵.

پاسخ مسئله‌ی ۶.

پاسخ مسئله‌ی ۷.

پاسخ مسئله‌ی ۸.