تمرین ۴ سیستم عامل – پویا پارسا . ۹۲۳۱۰۰۵ – دکتر زرندی – آبان ماه ۱۳۹۴

۱- **(کد ضمیمه شده – threading/race.cpp)**

نرم افزار های پیام رسان متنی بر شبکه ممکن است حداکثر تعداد کاربران یک گروه را محدود کرده باشند اما به خاطر وجود تعداد چندین دیتا سنتر و فاصله ی زمانی به روز رسانی آنها یک race condition بین درخواست ها اتفاق میفتد و ممکن است این تعداد بیشتر از حداکثر شود.

این فاصله زمانی در سورس کد با یک توقف زمانی شبیه سازی شده است.

با چند بار اجرای برنامه نتایج متفاوتی مشاهده می شود ..

۲- **(کد ضمیمه شده – threading/rw.cpp)**

بدون استفاده از قفل های اجرای مشترک ممکن بود شرایط زیر رخ دهند:

۱. ورود بیش از ۲ فرآیند Reader (بلافاصله یکی بعد از اتمام دوباره وارد شود)
۲. عدم زیاد شدن مقدار reader_counter به دلیل به وجود آمدن شرایط مسابقه
۳. عدم کم شدن مقدار reader_counter به دلیل به وجود آمدن شرایط مسابقه
۴. تغییر مقدار خوانده شده در حین عملیات Writer

۳- مشکلات راه حل پترسون:

a) هیچ ضمانتی برای صحت کارکرد متد را ندارد و صحت به معماری پردازنده وابسته است.
b) نیازمند استفاده از دستورالعمل های Atomic هست.
c) در سیستم های چند هسته ای به راحتی قابل استفاده نیست.

```
LOCK:

interested[id] = 1                interested[other] = 1
turn = other                      turn = id

while turn == other               while turn == id
  and interested[other] == 1        and interested[id] == 1


UNLOCK:

interested[id] = 0                interested[other] = 0
```

به دلیل به وجود آمدن شرایط مسابقه عملا قابل پیاده سازی نیست این الگوریتم و نیازمند متغیر های atomic است.

۴- راه حل مانیتور در سطح OOP مطرح می شود و با آن می توان کاری کرد که به طور همزمان توابع یا متد های یک کلاس توسط یک Thread صدا زده شوند.

گرچه مانیتور با سمافور ها قابل پیاده سازی است، این دو راهکار به طور اساسی تفاوت دارند:

- **مانیتور**: در یک مانیتور همزمان یک رویه قابلیت فعالیتی دارد و کنترل می شود.
- **سمافور**: کلیدی است که دسترسی به یک منبع مشترک را کنترل می کند.

مانیتور ها کلیه اتفاقات و فرآیند هایی را که به قفل کردن دسترسی وجود دارند مدیریت می کند. در واقع در راهکار های سمافور هر منبعی که به طور مشترک امکان تغییر دارد باید به طور صریح قفل شود اما روتین های مانیتور این کار را به طور خودکار انجام می دهند. (تصور کنید در یک فرآیند با چندین تابع که با متفیر های مشترک کار می کنند این کار تا چه حد دشوار خواهد شد)

مثال: در نرم افزار حساب گیری، کلاسی که داده ها را نگهداری می کند و دو رویه گزارش گیر و به روز رسانی تنها نمی توانند با استفاده از یک سمافور ساده تضمین کنند گزارش ها هنگامی که تابع به روز رسانی درحال به روز رسانی کلاس است تولید نشوند. (در این حالت گزارش گیری قطعا اشتباه است گرچه هیچگونه شرایط مسابقه ای هم وجود ندارد) اما به راحتی با ایجاد یک مانیتور روی آن کلاس می توان این مساله را حل کرد .