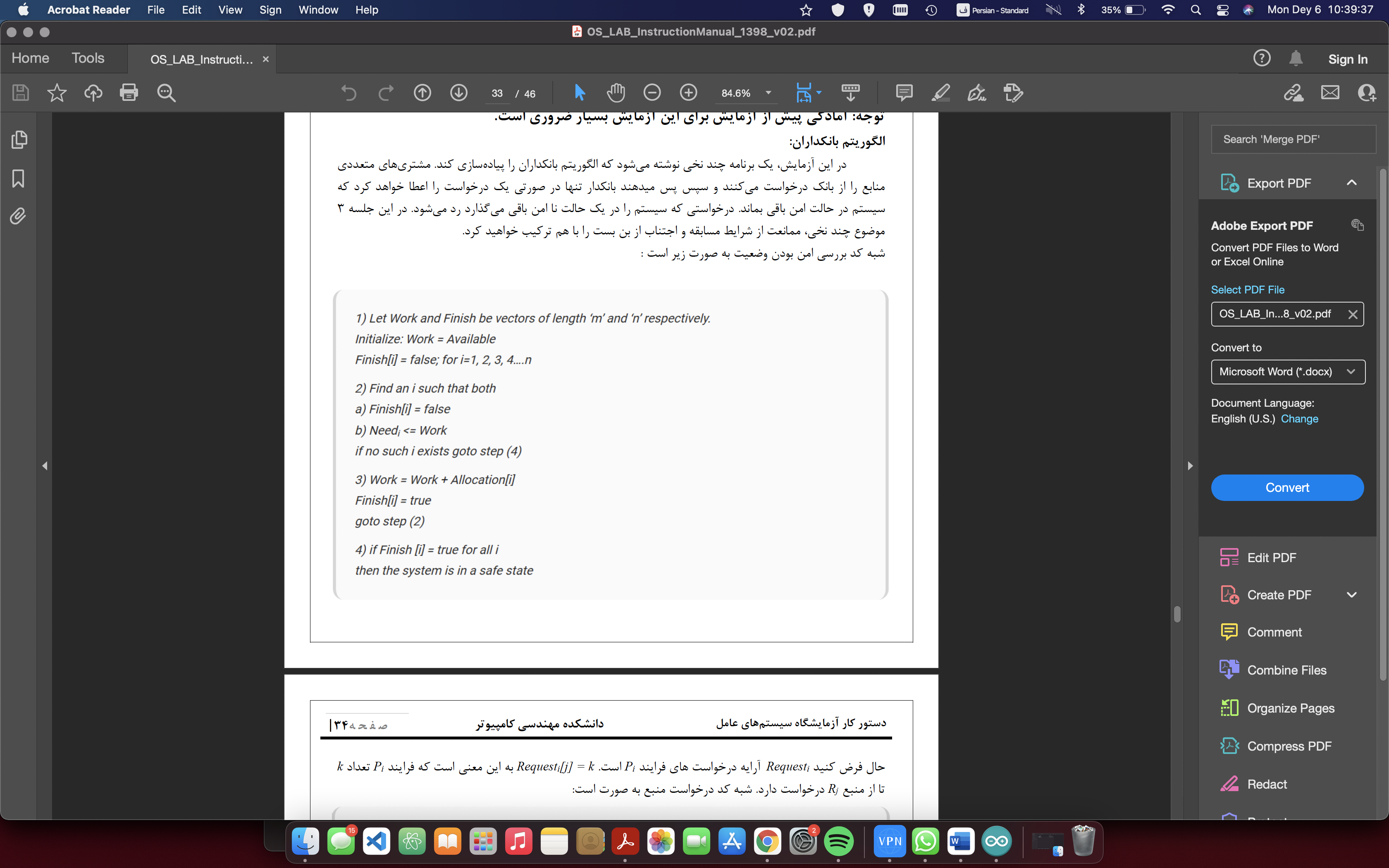
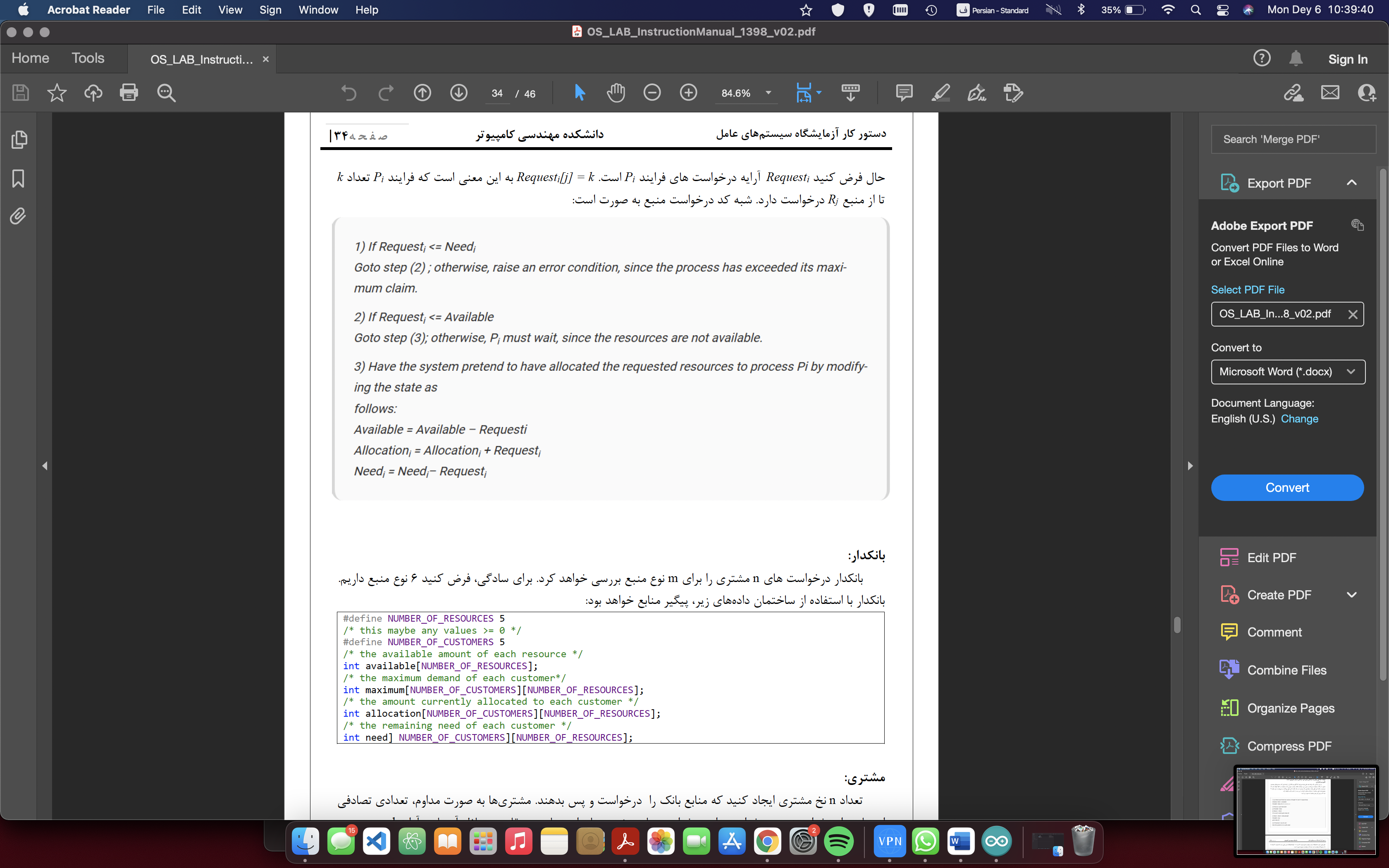
به نام خدا

«آزمایشگاه سیستم عامل»

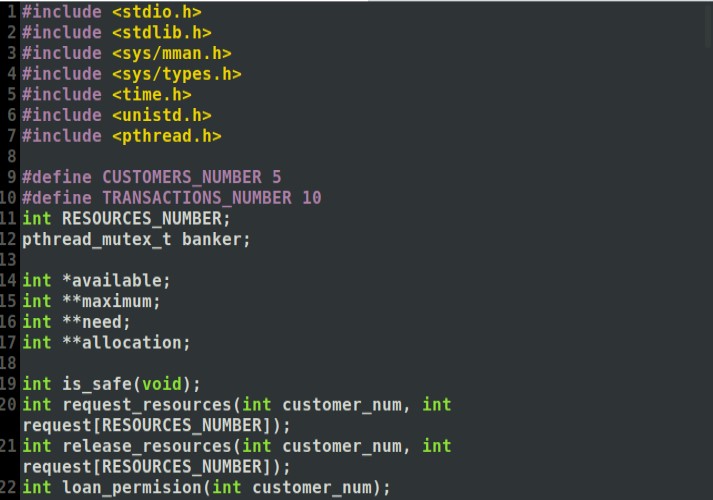
**آزمایش۷**

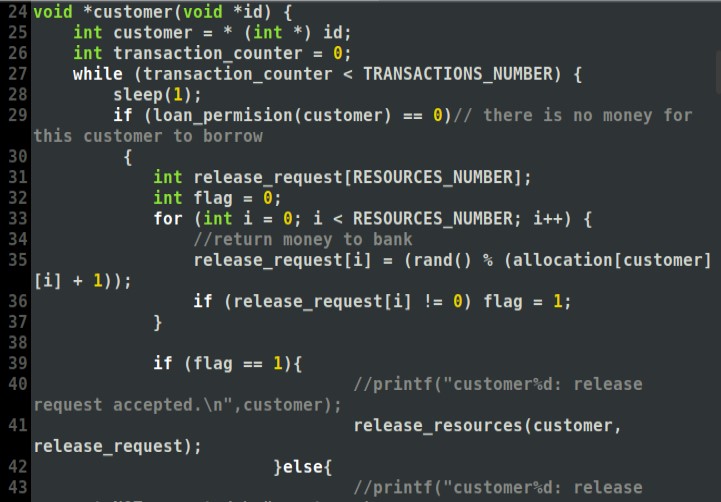
**اعضای گروه: هلیاسادات هاشمی پور۹۸۳۱۱۰۶-روژینا کاشفی ۹۸۳۱۱۱۸**

در این آزمایش در اصل یک مساله resource allocation داریم که می خواهیم به مشکل deadlock و race condition نخورد و برای حل این منظور از mutexکمک می گیریم و از pthread استفاده می کنیم.کدی که نوشتیم با توجه به شبه کدی که در دستورکار آمده است پایده سازی شده است:



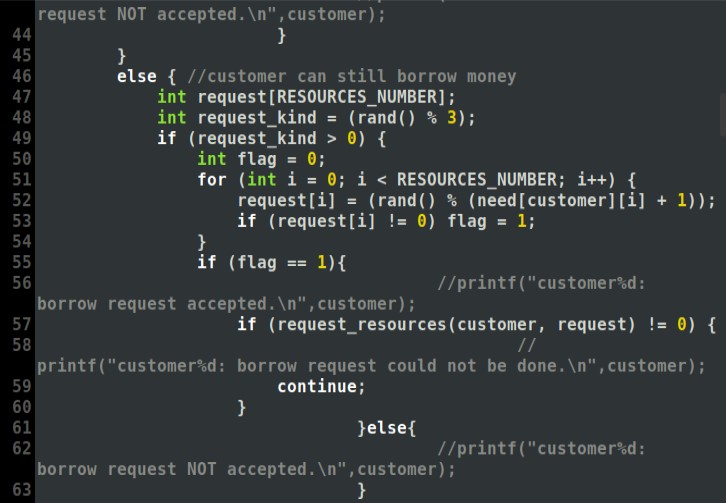
**کد**

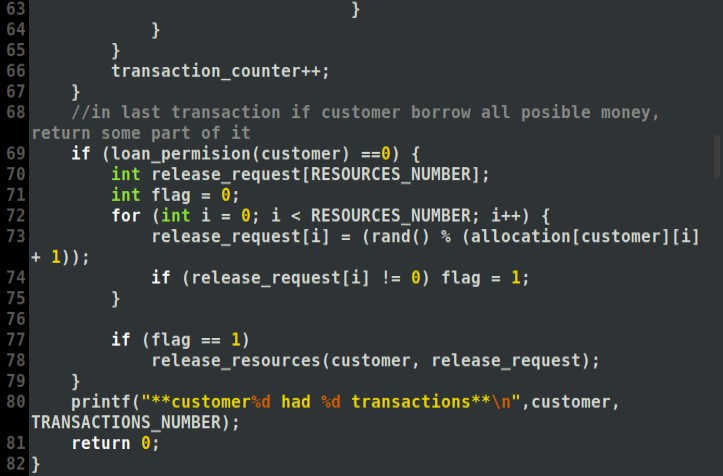
در ابتدا کتابخانه های مورد نیاز را include می کنیم و تعداد مشتری ها و تعداد تراکنش های آن را مشخص می کنیم و mutex به نام banker را به شکل global تعریف می کنیم. متغیرهای available و maximum و need و allocation را به شکل global تعریف مرده ایم و در تابع main آن ها را مقداردهی می کنیم.



تمامی عملیات های یک مشتری انجام می شود. Printfهایی که کامنت شده اند برای تست کردن می باشند.

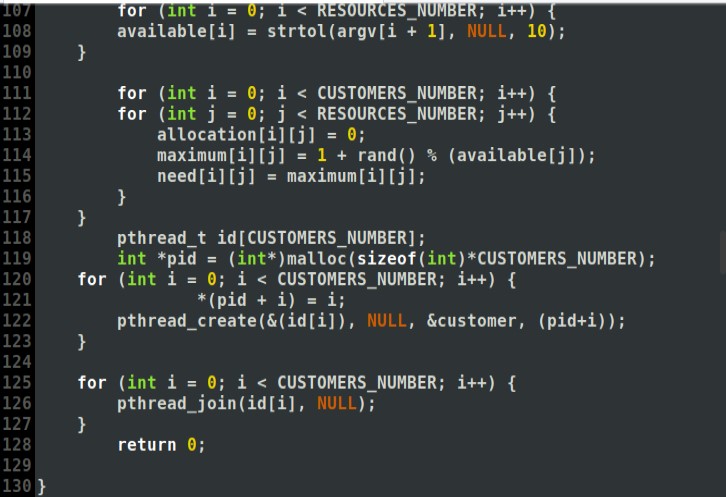
در یک حلقه که شرط اتمام آن انجام تراکنش ها به میزان کافی است، مشتری همواره درخواست انجام تراکنش می دهد و در صورت قبول شدن درخواست مشتری، جابجایی پول ها انجام می شود و آرایه ها به روز رسانی می شود.اگر مشتری درخواست وام داشته باشد با احتمالی درخواست آن را بررسی میشود و تابع request\_resource صدا زده میشود و از هر منبع می تواند مقداری از needازآن را بردارد. اگر مشتری امکان درخواست وام را نداشته باشد در ifاول، باید قسط پرداخت کند مبلغ آن به طور رندوم با توجه به مقدار allocation برای آن مشتری، انتخاب میشود و تابع صدا زده می شود.بعد از حلقه تمام خانه های need مربوط به مشتری برابر صفر بود و او مجبور است قسطی پرداخت کند که دقیقا کدی شبیه قسمت نخست است.



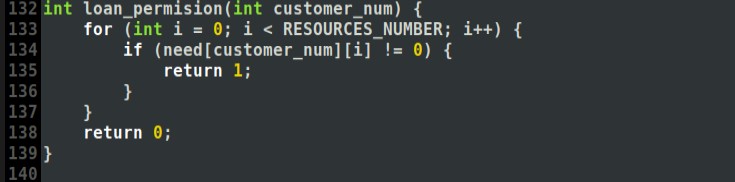


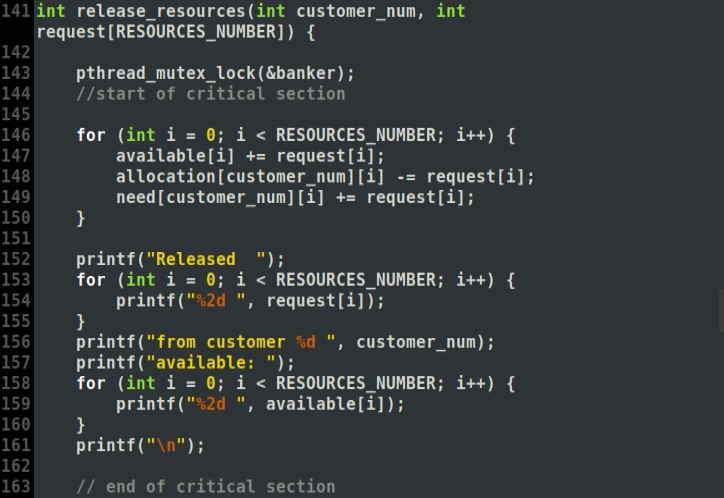


ابتدا منابع را که از طریق args در ورودی داده می شود می خوانیم و سپس با توجه به این تعداد و تعداد مشتریان آرایه های مورد نیاز را مقدار دهی می کنیم. در نهایت به تعداد مشتریان thread می سازیم و هر ترد تابع customer را اجرا کند. در انتها با pthread\_join صبر می کنیم تا اجرای همه تردها انجام شود و در نهایت اجرای برنامه به پایان می رسد.

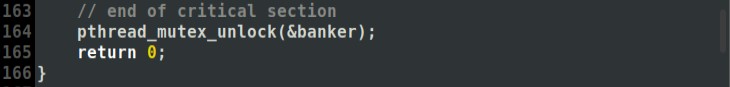


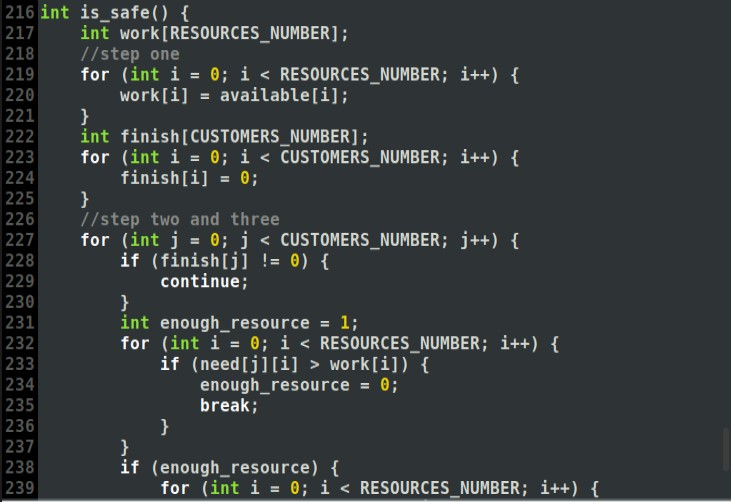
چک می شود که أیا مشتری می تواند در خواست وام بدهد یا خیر،اگر همه خانه های need[customer\_num]برایش صفر باشد نمی تواند درخواست وام بدهد و باید قسط پرداخت کند.



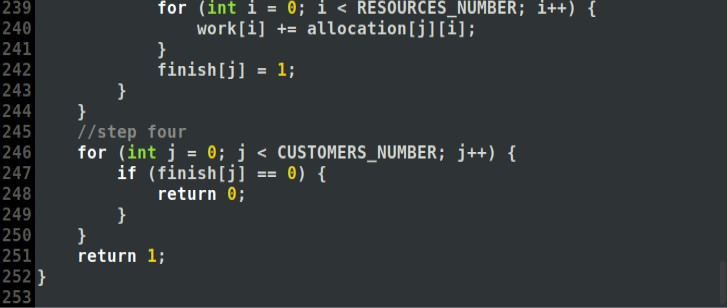


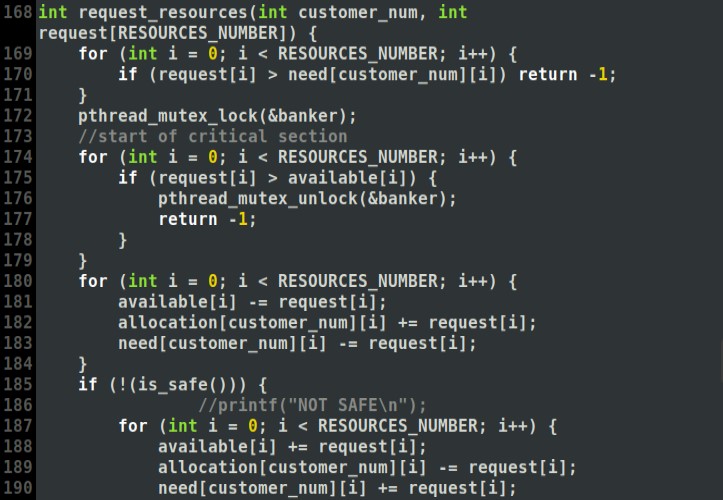
در این تابع مشتری می تواند قسط خود را پرداخت کند. در ابتدا چون می خواهیم critical section شویم دسترسی بقیه تردها را به حافظه ها و آرایه ها لاک می کنیم تا race condition پیش نیاید و سپس مقدار آرایه ها را به روزرسانی میکنیم و پیام مناسب را چاپ میکنیم.بعد چون از critical section خارج میشویم ،banker را آنلاک می کنیم و بقیه تردها اجازه دسترسی می دهیم.



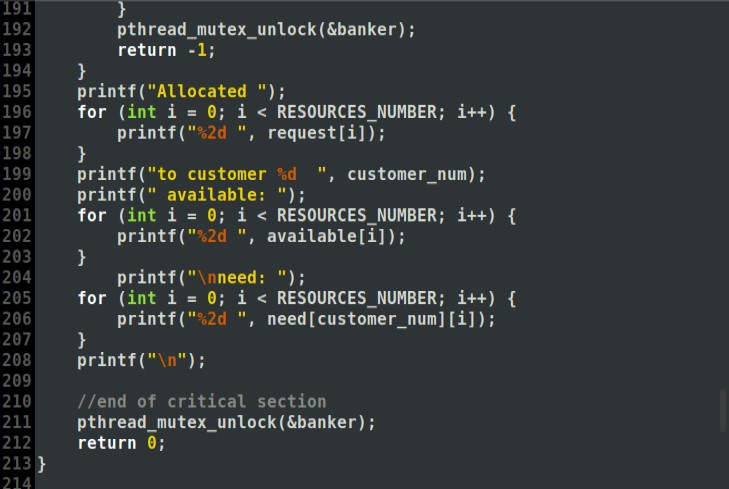


همه ۴ گام ذکر شده در دستورکار این تابع پیاده سازی شده است.این تابع مشخص می کند که آیا درخواست وام یک مشتری بانکدار را در آینده دچار مشکل می کند یا خیر. در واقع تشخیص میدهد که آیا ممکن است در آینده با ددلاک بخوریم یا خیر.



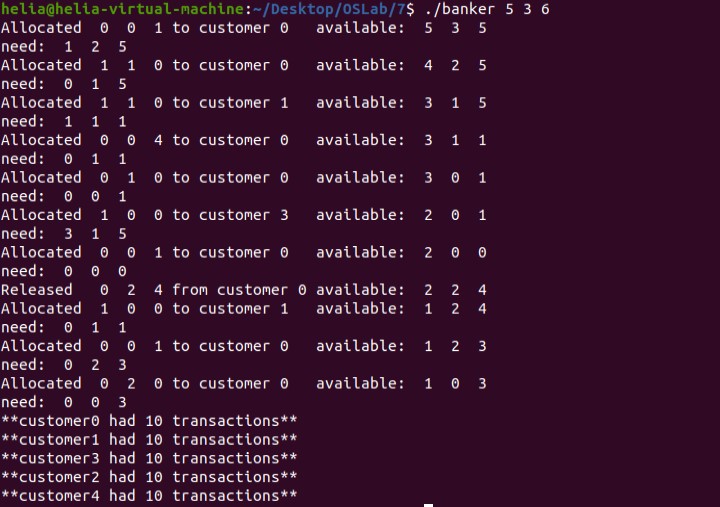


این تابع درخواست وام مشتری را بررسی میکند و در صورت قابل قبول بودن درخواست تابع is\_safe را صدا میزند و در نهایت اگر همه شرایط محیا بود وارد critical section می شود و آرایه ها را آپدیت میکند.در ابتدا چم می شود مشتری بیش از حد مجاز خود درخواست وام نداده باشد و اگر بیشتر بود درخواست رد می شود و return -1 میکند. اگر درخواست مورد قبول بود وارد ناحیه بحرانی می شود و به کمک mutex دسترسی بقیه تردها به این قسمت از کد را بلاک می کند و از race condition جلوگیری می شود. سپس چک می شود که منابع به اندازه کافی برای درخواست وام مشتری وجود داشته باشد و اگر وجود داشت ، مقدار درخواستی مشتری از available کم می شود و به allocation اضافه می شود و بقیه آرایه های مربوطه نیز به روزرسانی می شود. سپس تابع is\_safe صدا زده می شود.اگر خروجی این تابع ۱ بود پام داده می شود و پیام های مناسب چاپ می شود . banker آنلاک می شود وگرنه یعنی ممکن است با دادن وام بعدا به ددلاک بخوریم و وام داده نمی شود و آرایه ها را به حالت اولیه باز می گردد و banker آنلاک می شود. درحالتی که منابع کافی وجود نداشت وام داده نمی شود و banker آنلاک می شود.

****

**خروجی**

برای ۱۰ تراکنش

****