گزارش پروژه‌ی شبیه سازی کامپیوتری

اعضای گروه:
محمدمهدی به نصر ۹۷۱۰۵۷۹۳
کامیار درزی لاریجانی ۹۷۱۰۵۹۱۷

استاد: دکتر پیوندی

این کد یک شبیه‌ساز سیستم ابری با استفاده از کتابخانه‌های `simpy` و `numpy` است. این شبیه‌ساز برای مدل‌سازی و تحلیل عملکرد یک سیستم ابری با تعداد مشتریان و خدمات مختلف طراحی شده است. داده‌های ورودی این سیستم شامل متغیرهای تصادفی مانند میانگین فراخوانی مشتریان (`landa`)، میانگین زمان اجرا (`miu`)، و تعداد مشتریان (`users`) است.

1. `find_kind_of_work(users)`: این تابع نوع کارهایی که مشتریان درخواست می‌دهند را بر اساس توزیع احتمال داده شده تولید می‌کند و در لیست `works` ذخیره می‌کند.
2. `calc_zamane_residan(zamane_residan, nerkhArrivalTime, users)`: این تابع زمان ورود مشتریان به سیستم را محاسبه می‌کند.
3. `calc_shoroe_serviceDehi(shoroe_serviceDehi, nerkh_serviceDehi_moshtari_schedular, users)`: این تابع شروع زمان اجرا (شروع سرویس دهی) برای هر مشتری را محاسبه می‌کند.
4. `calc_saf_schedular(saf_schedular, zamane_residan, shoroe_serviceDehi, users)`: این تابع میزان اشتباه زمانی بین ورود مشتری و شروع سرویس دهی را برای هر مشتری محاسبه می‌کند.
5. `remove_expired_users(lastList, newList)`: این تابع کاربران منقضی شده را از لیست اولیه حذف می‌کند و لیست جدید را تولید می‌کند.
6. بخش‌های پایانی کد، مقداردهی و محاسباتی انجام می‌دهند و نتایج نهایی را چاپ می‌کنند. این نتایج شامل اطلاعاتی مانند تعداد و مدت زمان کل کارها، میانگین زمان اجرا برای هر نوع کار، میانگین زمان اشتباه در صف (Scheduling Delay) و نسبت مشتریان منقضی به کل مشتریان می‌باشند.

این کد یک شبیه‌ساز سیستم ابری را پیاده‌سازی می‌کند و به تحلیل عملکرد سیستم می‌پردازد. در ادامه توضیحاتی در مورد عملکرد کلی کد آورده شده است:

1. **پارامترهای ورودی:**
   - `landa`: میانگین فراخوانی مشتریان.
   - `alpha`: پارامتری که در تولید زمان اجرا برای هر مشتری مورد استفاده قرار می‌گیرد.
   - `miu`: میانگین زمان اجرا.
   - `users`: تعداد مشتریان.
2. **تولید داده‌ها:**
   - زمان ورود مشتریان (`nerkhArrivalTime`) با استفاده از توزیع نمایی تولید می‌شود.
   - زمان اجرا برای هر مشتری (`mohlatZamani_karha_deadline`) با استفاده از توزیع نمایی تولید می‌شود.
   - نرخ سرویس دهی مشتریان (`nerkh_serviceDehi_moshtari_schedular`) با استفاده از توزیع پواسون تولید می‌شود.
3. **انجام محاسبات:**
   - توابع مختلف مانند `find_kind_of_work`، `calc_shoroe_serviceDehi`، `calc_zamane_residan` و `calc_saf_schedular` برای محاسبه متغیرهای مختلف همچون نوع کار، زمان ورود مشتری، شروع زمان اجرا، و اشتباه زمانی انجام می‌دهند.
4. **مدیریت کارها:**
   - مشتریان به دو نوع کار تقسیم می‌شوند (نوع ۱ و نوع ۲) و زمان اجرای هر کار بر اساس توزیع نمایی محاسبه می‌شود.
   - افرادی که منقضی شده‌اند از لیست مشتریان حذف می‌شوند و تخصیص منابع بر اساس موجودیت کمینه صورت می‌گیرد.
5. **گزارش‌گیری:**
   - نتایج نهایی شامل مواردی مانند تعداد و مدت زمان کل کارها، میانگین زمان اجرا برای هر نوع کار، میانگین زمان اشتباه در صف (Scheduling Delay) و نسبت مشتریان منقضی به کل مشتریان می‌باشند.

به طور کلی، این کد یک محیط شبیه‌سازی ایجاد می‌کند تا عملکرد یک سیستم ابری در مواجهه با تعداد مشتریان مختلف و تنوع در نوع کارها را بررسی کند.

تابع `find_kind_of_work(users)` وظیفه تولید نوع کارهایی که مشتریان درخواست می‌دهند را دارد. توضیحات زیر مفصل‌تر است

```
def find_kind_of_work(users):
    while users !=0:
        items = {1: 10, 2: 90}
        works.append(  (random.choice([k for k in items for dummy in range(items[k])]))   )
        # darkhastZadan( resorces)
        users -=1
find_kind_of_work(users)
```

- در این تابع، یک دیکشنری به نام `items` تعریف شده است که هر کلید آن نوع یک کار و مقدار مرتبط با آن تعداد مشتریان متناظر است. در مثال فعلی، نوع ۱ (کار اول) با تعداد ۱۰ مشتری و نوع ۲ (کار دوم) با تعداد ۹۰ مشتری در نظر گرفته شده‌اند.
- سپس از این دیکشنری، یک نوع کار به صورت تصادفی انتخاب می‌شود. تابع `random.choice` برای انتخاب یک عنصر تصادفی از یک لیست استفاده می‌شود. لیست اینجا به وسیله لیست تکرار شده متناظر با تعداد مشتریان هر نوع کار ایجاد شده است.
- نوع کار انتخاب شده به لیست `works` اضافه می‌شود.
- تعداد مشتریان باقی‌مانده با کاهش از تعداد اولیه کم می‌شود. این عمل تا زمانی ادامه می‌یابد که تعداد مشتریان به صفر برسد.

تابع `calc_zamane_residan(zamane_residan, nerkhArrivalTime, users)` وظیفه محاسبه زمان رسیدن (ورود) مشتریان به سیستم را دارد. در زیر توضیحات این تابع آورده شده است:

```
def calc_zamane_residan(zamane_residan,nerkhArrivalTime,users):
    zamane_residan.append(0)
    ii=1
    while ii !=users:
        a=nerkhArrivalTime[ii]
        b=zamane_residan[ii-1]
        c=a+b
        zamane_residan.append(c)
        ii+=1
calc_zamane_residan(zamane_residan,nerkhArrivalTime,users)
```

- این تابع از دو لیست `zamane_residan` و `nerkhArrivalTime` به عنوان ورودی استفاده می‌کند. لیست `zamane_residan` شامل زمان رسیدن مشتریان است و لیست `nerkhArrivalTime` شامل زمان ورود مشتریان به سیستم است.

- ابتدا یک مقدار صفر به لیست `zamane_residan` اضافه می‌شود چرا که زمان رسیدن برای مشتری اول صفر است.
- سپس یک حلقه `while` اجرا می‌شود که از مشتری دوم به بعد، زمان رسیدن مشتری جدید محاسبه و به لیست `zamane_residan` اضافه می‌شود. این محاسبه بر اساس فرمول `زمان رسیدن مشتری جدید = زمان ورود مشتری + زمان رسیدن مشتری قبلی` انجام می‌شود.
- این عمل تا زمانی ادامه می‌یابد که تمام مشتریان در سیستم وارد شده باشند (`ii` تا `users - 1`).

تابع `calc_shoroe_serviceDehi(shoroe_serviceDehi, nerkh_serviceDehi_moshtari_schedular, users)` مسئول محاسبه شروع زمان اجرا (شروع سرویس دهی) برای هر مشتری است. در زیر توضیحات این تابع آورده شده است:

```
def calc_shoroe_serviceDehi(shoroe_serviceDehi,nerkh_serviceDehi_moshtari_schedular,users):
    shoroe_serviceDehi.append(0)
    ii=1
    while ii !=users:
        a= shoroe_serviceDehi[ii-1] + nerkh_serviceDehi_moshtari_schedular[ii-1]
        b=zamane_residan[ii]
        m=max(a,b)
        shoroe_serviceDehi.append(m)
        ii+=1
calc_shoroe_serviceDehi(shoroe_serviceDehi,nerkh_serviceDehi_moshtari_schedular,users)
```

- این تابع از سه ورودی استفاده می‌کند: `shoroe_serviceDehi` که شامل شروع زمان اجرا برای هر مشتری است، `nerkh_serviceDehi_moshtari_schedular` که نرخ سرویس دهی مشتریان می‌باشد، و `users` تعداد کل مشتریان.
- ابتدا یک مقدار صفر به لیست `shoroe_serviceDehi` اضافه می‌شود چرا که شروع زمان اجرا برای مشتری اول صفر است.
- سپس یک حلقه `while` اجرا می‌شود که از مشتری دوم به بعد، شروع زمان اجرا برای هر مشتری جدید محاسبه و به لیست `shoroe_serviceDehi` اضافه می‌شود.
- محاسبه این شروع زمان اجرا بر اساس فرمول `شروع زمان اجرا مشتری جدید = max(پایان سرویس دهی مشتری قبلی، زمان رسیدن مشتری)` انجام می‌شود.

تابع `calc_saf_schedular(saf_schedular, zamane_residan, shoroe_serviceDehi, users)` مسئول محاسبه میزان اشتباه زمانی بین زمان ورود مشتری و شروع زمان اجرا (سرویس دهی) برای هر مشتری است. در زیر توضیحات این تابع آورده شده است:

```
def calc_saf_schedular(saf_schedular,zamane_residan,shoroe_serviceDehi,users):
    ii=0
    while ii !=users:
        a=abs(zamane_residan[ii]-shoroe_serviceDehi[ii])
        saf_schedular.append(a)
        ii+=1
calc_saf_schedular(saf_schedular,zamane_residan,shoroe_serviceDehi,users)
```

- این تابع از سه ورودی استفاده می‌کند: `saf_schedular` که شامل اشتباه زمانی برای هر مشتری است، `zamane_residan` که زمان ورود مشتریان به سیستم را نشان می‌دهد، و `shoroe_serviceDehi` که شروع زمان اجرا (سرویس دهی) برای هر مشتری است.
- یک حلقه `while` اجرا می‌شود که برای هر مشتری، اشتباه زمانی محاسبه و به لیست `saf_schedular` اضافه می‌شود.
- محاسبه اشتباه زمانی از اختلاف مطلق بین زمان ورود مشتری و شروع زمان اجرا انجام می‌شود و به لیست اضافه می‌شود.

تابع `remove_expired_users(lastList, newList)` مسئول حذف کاربران منقضی شده از یک لیست و ایجاد یک لیست جدید بدون این کاربران است. در زیر توضیحات این تابع آورده شده است:

- این تابع دو لیست به نام‌های `lastList` و `newList` را به عنوان ورودی دریافت می‌کند.
- برای هر عنصر در `lastList`، اگر مقدار آن برابر با `-11111` نباشد (یعنی کاربر منقضی نشده باشد)، این عنصر به لیست `newList` اضافه می‌شود.
- به این ترتیب لیست جدید `newList` بدون کاربران منقضی شده ایجاد می‌شود.

سایر بخش‌های کد که باقی مانده‌اند، در زیر توضیح داده شده‌اند:

تعداد کل کارها:

```python
numberOf1 = works.count(1)
numberOf2 = works.count(2)
```

در این بخش، تعداد کل کارهای نوع ۱ (`numberOf1`) و ۲ (`numberOf2`) محاسبه می‌شود.

محاسبه زمان اجرا و اشتباه زمانی برای هر نوع کار:

```python
countOfTime_one = 0
countOfTime_two = 0

if 1 in works:
    all_indexes_work1 = [a for a in range(len(works)) if works[a] == 1]
    for i in all_indexes_work1:
        countOfTime_one += mohlatZamani_karha_deadline[i]

if 2 in works:
    all_indexes_work2 = [a for a in range(len(works)) if works[a] == 2]
    for i in all_indexes_work2:
        countOfTime_two += mohlatZamani_karha_deadline[i]
```

این بخش مسئول محاسبه مجموع زمان اجرا برای هر نوع کار است. اگر نوع کاری انجام شده باشد (مثلاً نوع ۱)، این بخش زمان اجرا را برای آن نوع کار محاسبه می‌کند.

محاسبه زمان اجرا متوسط برای هر نوع کار:

```
mianginZamaneKare1 = countOfTime_one / numberOf1
mianginZamaneKare2 = countOfTime_two / numberOf2
```

در این بخش، میانگین زمان اجرا برای هر نوع کار محاسبه می‌شود.

محاسبه میانگین زمان اشتباه در صف (Scheduling Delay):

```
mianginZamanSafDarSchedularKare1 = safWork1 / numberOf1
mianginZamanSafDarSchedularKare2 = safWork2 / numberOf2
```

این بخش مسئول محاسبه میانگین زمان اشتباه در صف (Scheduling Delay) برای هر نوع کار است.

محاسبه نسبت کارهای منقضی به کل کارها:

```
nesbateKarhaye1KeExpiredShodanBeKoleKare1 = (monghazi1 / kooleWork1) * 100
nesbateKarhaye2KeExpiredShodanBeKoleKare2 = (monghazi2 / kooleWork2) * 100
```

در این بخش، نسبت تعداد کارهای منقضی به تعداد کل کارها برای هر نوع کار محاسبه می‌شود.

محاسبه نسبت کارهای منقضی به کل مشتریان:

```
nesbateKarhaye1KeExpiredShodanBeKoleKareha = (monghazi1 / users) * 100
nesbateKarhaye2KeExpiredShodanBeKoleKareha = (monghazi2 / users) * 100
```

این بخش مسئول محاسبه نسبت تعداد کارهای منقضی به تعداد کل مشتریان برای هر نوع کار است.

محاسبه میانگین طول صف سرورها:

```
safCores = 0
for i, j in cores.items():
    safC
```