به نام خدا



**پروژه عملی درس شبیه­سازی کامپیوتر**

**دکتر بردیا صفایی**

علی شریفی 98109601

حمیدرضا دهباشی قربانعلی 98105762

مقدمه:

در این پروژه باید یک سیستم مبتنی بر صف مطابق داک را طراحی می­کردیم.

سیستم به اینصورت طراحی شده است که شامل 7 نوع سرویس است که هر کدام از سرویس­ها می­توانند چندین نمونه داشته باشند. این 7 نوع سرویس به طور کلی در 3 لایه دسته­بندی می­شوند، به این ترتیب که لایه اول شامل (درگاه وب، درگاه موبایل)، لایه دوم شامل (مدیریت رستوران­ها، مدیریت مشتریان، مدیریت سفارش­ها) و لایه سوم نیز شامل (ارتباط با پیک، پرداخت­ها) می­باشند. چینش سرویس­ها به گونه­ای است که هیچکدام از نمونه­های سرویس­های موجود در یک لایه، از یکدیگر خدمتی را دریافت نمی­کنند.

هر درخواست حتما از لایه 1 و 2 عبور می­کند و بعضی درخواست­ها نیز به لایه سوم می­رسند و از یکی از سرویس­های لایه 3 نیز خدمتی را دریافت می­کنند.

نکته با اهمیت دیگر در طراحی این سیستم این است که تمام نمونه­های یک سرویس، صف مشترک دارند. به عبارتی سیستم از 7 صف تشکیل شده است.

درباره خدمت­رسانی سرویس­ها به یکدیگر به اینصورت است که سرویس­های موجود در لایه پایین­تر از سرویس­ها موجود در لایه بالاتر سرویس دریافت می­کنند و ترتیب خدمت­رسانی نیز از لایه سوم به سمت لایه اول است. یعنی اگر درخواست "پیگیری سفارش" داشته باشیم، سرویس "درگاه موبایل" از سرویس "مدیریت سفارش­ها" خدمت دریافت می­کند. پس ابتدا مدیریت سفارش­ها انجام می­شود و سرویس درگاه موبایل منتظر آن می­ماند سپس با تمام شدن مدیریت سفارش­ها، درگاه موبایل شروع به خدمت رسانی می­کند.

تمامی موارد امتیازی اعم از نرخ خطا، اولویت درخواست و حداکثر زمان انتظار درخواست­ها، در این پروژه پیاده­سازی شده است.

با ورودی گرفتن تعداد نمونه­های سرویس­ها، نرخ ورود داده­ها به سیستم، مدت زمان شبیه­سازی و حداکثر زمان انتظار برای هر نوع درخواست، خروجی­های زیر را به ما می­دهد:

1. پارامترهای چاپ­شده در صفحه کنسول:

* میانگین طول صف­ها (به تفکیک هر سرویس)
* میانگین زمان صرف شده در صف­ها (به تفکیک هر سرویس و به صورت کلی)
* میزان بهره­وری سرویس­ها (به تفکیک هر نمونه)
* درصد درخواست­های قبول­شده (به تفکیک هر درخواست و به صورت کلی)
* درصد درخواست­های مواجه­شده با خطا (به تفکیک هر درخواست و به صورت کلی)
* درصد درخواست­های منقضی شده (به تفکیک هر درخواست و به صورت کلی)

1. فایل­های تولید شده در پوشه پروژه:

* log.txt: این فایل تمامی اطلاعات سیستم را نگهداری می­کند به اینصورت که به ازای هر ثانیه، حاوی اطلاعات هر نمونه اعم از درخواست­های در حال اجرا و اینکه آیا خالی هستند یا نه، و همچنین حاوی اطلاعات مربوط به سرویس­ها حاوی صف هر درخواست می­باشد. برای بررسی صحت سیستم از این فایل بسیار استفاده شد.
* arrival.txt: این فایل حاوی تمامی داده­هایی است که به وارد سیستم شده­اند. ساختار آن به اینصورت است که در هر خط یک عدد وجود دارد که زمان ورود آن درخواست به سیستم است و بعد از آن نوع درخواست آورده شده است.
* pending.txt: این فایل حاوی تمام درخواست­هایی است که نیمه­کاره مانده­اند. به اینصورت که در زمان اتمام شبیه­سازی، نه به پایان رسیده­اند، نه منقضی شده­اند و نه به خطا خورده­اند. در اولین خط تعداد این درخواست­ها می­آید و در خطوط بعدی، درخواست­ها آورده می­شوند.
* نکته­ای که خوب است به آن اشاره کنم این است که ما زمان زیادی روی بهینگی سیستم نگذاشتیم زیرا جزو معیارهای نمره دهی نبود. مخصوصا راجع به نمونه­های هر سرویس با زیاد شدن آن­ها سیستم به صورت نمایی رشد می­کند. (روند اجرا در ادامه به طور دقیق توضیح داده می­شود.) مثلا به ازای ورودی­های داک، حدود 6 ساعت طول کشید تا سیستم خروجی مناسبی به ما بدهد.

روند شبیه­سازی سیستم:

برای شبیه­سازی درخواست­ها از کلاس Request و برای شبیه­سازی سرویس­ها از کلاس Service استفاده کردیم. توجه کنید این دو کلاس abstract هستند. پس از آن تمامی انواع سرویس­ها و درخواست­ها را به گونه­ای تعریف می­کنیم تا از این دو کلاس ارث ببرند.

هر نمونه از service، یک inProgress دارد که درخواست در حال خدمت­رسانی را نشان می­دهد. همچنین هر نوع Service، یک queue دارد که صف را شبیه­سازی می­کند. بعلاوه، هر نوع Service لیستی از تمامی نمونه­های خودش دارد و آن­ها را در instances ذخیره می­کند. مابقی فیلد­های سرویس، جهت ارزیابی معیار­های عملکرد و یا ثابت­هایی است که توسط کاربر ورودی داده می­شود.

هر نمونه از Request، شامل id و زمان ورود به سیستم است که با time نشان­ می­دهیم. ثابت­های مربوط به درخواست­ها مانند نرخ خطا و .. نیز در کلاس Request قابل مشاهده است. مابقی فیلدهای این کلاس، مربوط به ارزیابی عملکرد سیستم است.

\* 6 متغیر بسیار مهم داریم:

* arrival\_table: داده­­های ورودی سیستم تولید و در آن نگهداری می­شوند.
* current\_time: زمان حال حاضر در سیستم را نشان می­دهد.
* current\_requests: درخواست­ها در حال جریان (در صف یا در حال خدمت­گرفتن) را شامل می­شود.
* accepted\_requests: درخواست­های موفق­شده را نگه ­می­دارد.
* failed\_requests: درخواست­های با خطا مواجه شده را نگه می­دارد.
* timeout\_requests: درخواست­های منقضی شده را نگه می­دارد

در ابتدای برنامه تمامی داده­های ورودی با توجه به نرخ ورودی ساخته می­شوند و در جدول arrival\_table نگهداری می­شوند. سپس از current\_time=0 شروع می­کنیم و آن را در چرخه while که در خطوط 522 تا 546 وجود دارد، یکی یکی زیاد می­کنیم تا به total\_time که ماکسیمم زمان شبیه­سازی است برسد. در هر ثانیه (هر چرخه از این حلقه)، ابتدا با استفاده از تابع isIdle بررسی می­کنیم که اگر سیستم به کل خالی است، به اولین لحظه­ای برویم که سیستم در آن ورودی می­گیرد. سپس آن درخواست را با استفاده از جدول arrival\_table وارد سیستم می­کنیم. این کار با صدا زدن متد doRequest روی درخواست مورد نظر صورت می­گیرد. به اینصورت داده مورد نظر در arrival\_table در زمان مناسب وارد سیستم می­شود. (خطوط 523 تا 532)

سپس در خطوط 533 تا 539، فیلدهای مورد نظر برای ارزیابی عملکرد پر می­شوند. مثلا queue\_sums آرایه ­ایست که طول صف هر سرویس را نگه می­دارد و در هر ثانیه آن را تغییر می­دهیم.

سپس متد checkTimeOuts است که در آن درخواست­هایی که منقضی شده­اند مشخص می­شوند. پس از آن handleProgressed را داریم که در آن درخواست­هایی که با موفقیت تمام شده­اند شناسایی می­شوند و از لیست درخواست­های جاری حذف می­شوند. پس از آن نیز handleFaults اجرا می­شود که مسئول شناسایی دستور­هایی است که در طول آن ثانیه ممکن است با خطا مواجه شوند.

پس از صدا زدن این 3 تابع، یک­سری از نمونه­ها خالی شده­اند به این معنا که دیگر خدمتی ارائه نمی­دهند. پس باید در صورت وجود درخواست دیگری در صف، سریعا به آن درخواست خدمت ارائه دهند. handleQueues این­ کار را برای ما انجام می­دهد به اینصورت که نمونه­هایی را که بیکار هستند شناسایی می­کند و در صورت لزوم آن­ها را با درخواست­های مرتبط پر می­کند.

در انتها نیز log با استفاده از تابع showDebug صورت می­گیرد و زمان یک واحد افزایش می­یابد.

در آخر کد نیز معیارهای مورد نیاز در خروجی با استفاده از فیلدهایی که در طی شبیه­سازی پر شده­اند، مقداردهی می­شوند(خطوط 622 تا 663). پیش از خط 622 و پس از حلقه نیز، داده­های مورد نیاز که قبلا در طول شبیه­سازی تجمیع نشده بودند، به صورت تجمیعی در لیست­های a, b, c, d جمع می­شوند.

فرایندهای متفاوت در شبیه­سازی سیستم:

اضافه کردن درخواست به سیستم:

با استفاده از صدازدن doRequest روی درخواست مورد نظر صورت می­گیرد. در متد doRequest، مقداردهی اولیه به درخواست صورت می­گیرد و متدی به نام SpecifyHandleInstanceصدا زده می­شود. وظیفه این متد، تخصیص نمونه مناسب به درخواست مورد نظر است. در این متد از تابع serve نیز استفاده شده است. وظیفه تابع serve در حقیقت، هندل کردن خدمت­رسانی سرویس­ها به یکدیگر است. به اینصورت که در ابتدا برای هر درخواست، به ترتیب رو pipeline آن درخواست حرکت می­کند و رابطه­ای برای خدمت­گرفتن و خدمت­رسانی روی سرویس­ها مشخص می­کند. برای مثال اگر درخواست (ثبت سفارش با وب) وارد سیستم شود، مراحل زیر انجام می­شود:

1. doRequest روی درخواست (ثبت سفارش با وب) صدا زده می­شود و مقداردهی­های اولیه به آن صورت می­گیرد.
2. در متد doRequest، متد SpecifyHandleInstance صدا زده می­شود که وظیفه دارد با توجه به پارامترهای ورودی، اگر نمونه­ای (درگاه وب) خالی است این درخواست (ثبت سفارش با وب) را به آن assign کند و در غیر اینصورت آن را در صف (درگاه وب) قرار می­دهد.
3. به محض ورود درخواست به (درگاه وب) جهت خدمت گرفتن، خود سرویس (درگاه وب) نیز باید از سرویس (مدیریت سفارش­ها) خدمتی را دریافت کند پس تابع serve صدا زده می­شود.
4. در تابع serve، بسته به پایپ­لاین، SpecifyHandleInstance روی سرویس مرحله بعد صدا که (مدیریت سفارش­ها) است صدا زده می­شود. این فرایند رفت و برگشتی میان serve و specifyHandleInstance تا زمانی ادامه پیدا می­کند که به انتهای پایپ­لاین برسیم. در این صورت در تابع serve یک زمان سرویس­دهی با توجه به پارامتر­های ورودی اختصاص داده می­شود و درخواست به این صورت به سیستم اضافه می­شود.

اضافه کردن به صف:

از آن­جا که صف ما دارای اولیت است و یک سری از درخواست­ها به نسبت بقیه درخواست­ها اولویت بیشتری دارند، اضافه کردن درخواست به صف با اولویت موجود در متد addToQueues صورت می­گیرد به صورتیکه همواره درخواست­های موجود در هر صف اولا بر اساس اولویت و ثانیا بر اساس زمان مرتب شده­اند.

انجام درخواست توسط نمونه­ها:

پس از به اتمام رسیدن درخواست توسط هر نمونه (با بررسی کردن زمان حال حاضر و زمان مورد نیاز آن نمونه از سرویس برای خدمت­رسانی)، این اتفاق توسط handleProgressed شناسایی شده و تابع serve\_back روی آن نمونه صدا زده می­شود. این تابع به اینصورت است که با استفاده از درخواست در حال اجرا که زمان آن به اتمام رسیده است، نمونه­ای از آن سرویسی را که باید به آن خدمت ارائه می­کرده است را پیدا می­کند و روی آن نیز serve\_back را صدا می­زند. سپس مقدار inProgress برای نمونه برابر None می­شود. با این­ کار نمونه­ای که خدمت را دریافت کرده است، شروع به خدمت­رسانی می­کند و زمان سرویس­دهی مناسب به آن assign می­شود.اگر سرویس نمونه مورد نظر در لایه 1 باشد به این معناست که درخواست با موفقیت پشت سر گذاشته شده است و به درخواست­های موفق افزوده می­شود.

خطا خوردن درخواست­ها:

به اینصورت است که هر نمونه­ای که خطا بخورد، درخواست inProgress آن نمونه با خطا مواجه شده و آن درخواست به لیست درخواست­های خطا اضافه می­شود و آن درخواست از تمامی صفوف و نمونه­هایی که آن درخواست را انجام می­دهند، پاک می­شود و به اینصورت آن درخواست نیز از سیستم خارج می­شود.. این کار در متد fail هندل شده است.

منقضی شدن درخواست­ها:

در هر ثانیه تمامی درخواست بررسی می­شوند و اگر زمان انقضای آن­ها رسیده باشد، به لیست درخواست­های منقضی اضافه می­شوند. همچنین آن درخواست نیز از سیستم خارج می­شود. این کار در متد time\_out صورت گرفته است.

تحلیل نتایج سیستم: