به نام خدا

گزارش پروژه شبیه‌سازی کامپیوتری

**محمد مولوی - 99105753**

**محمدجواد ماهرالنقش – 99105691**

استاد: دکتر بردیا صفایی

تابستان 1402

**فهرست موضوعات**

مقدمه ......................................................................................................................................... 3

ساختار کلی..................................................................................................................................... 4

نحوه پیاده سازی ................................................................................................................................ 5

جزئیات کد .................................................................................................................................... 6

خروجی‌ها و نتیجه گیری ........................................................................................................................ 15

منابع و تشکرها................................................................................................................................. 21

**مقدمه**

در این پروژه به پیاده‌سازی چند سیستم صف میپردازیم که به 3 سیاست مختلف رفتار خواهند کرد. همچنین از 2 هاست و 1 روتر استفاده خواهیم کرد.

در شبکه، بسته‌ها می‌خواهند از مبدا به مقصد برسند، در این مسیر از روتر عبور می‌کنند.

در این پروژه می‌خواهیم رفتار روتر و تحولات بسته‌ها را بررسی کنیم و همچنین مواردی را در پایان مصورسازی (Visualize) کنیم و همچنین تمام موارد را باید شبیه‌سازی کنیم، یعنی زمان‌های میان‌ورودی (interarrival times)، زمان‌های اجرای بسته‌ها (execution times) و همچنین فرایند تخصیص بسته‌ها به هسته‌های پردازشی باید تماما شبیه‌سازی شوند.

برای انجام این کار از زبان پایتون استفاده کردیم و کدها همزمان در فایل پایتون و فایل جوپیتر نوتبوک موجود هستند.

همچنین گزارشی از کارهای انجام‌شده در پایان نمودارسازی شده و همچنین در فایل اکسل (csv) موارد مرتبط ذخیره میشوند.

جزئیات بیشتر را در ادامه گزارش بررسی میکنیم.

این پروژه، ذیل پروژه درس شبیه‌سازی کامپیوتری تعریف شده و در تابستان 1402 توسط تیم ما پیاده‌سازی شد.

**ساختار پروژه**

* **بسته (Packet)**

این موجودیت با کلاس Task پیاده‌سازی شده است. هر بسته‌ای که وارد سیستم می‌شود، مانند وظیفه‌ای است که باید به آن رسیدگی شود. هر بسته دارای ویژگی‌های مختلفی است که نیاز است تا برای بهینگی پیاده‌سازی برای آن کلاس جداگانه تعریف کنیم. اولویت، زمان ورود، و .. از مواردی است که در این کلاس ذخیره می‌شوند. توابع (methods) نیز در این کلاس تعریف شده که در ادامه گزارش بررسی می‌شوند.

* **ساختار کلی صف (BaseQueue)**

در اینجا نیاز است تا ما یک ساختار کلی را برای صف در این سیستم طراحی کنیم. البته صف جزئیات بیشتری دارد که در کلاس‌هایی که از این کلاس ارث‌بری می‌کنند تعریف می‌شوند.

توجه کنید که برای استانداردسازی پیاده‌سازی و همچنین خوانایی بهتر، تمام سیستم‌های صف که در ادامه معرفی می‌شوند از این کلاس صف ارث‌بری می‌کنند.

* **سیاست‌های صف**

1. سیاست FIFO: برای این سیاست، یک کلاس به نام FIFO تعریف شده است که از کلاس ‌سیاست کلی صف ارث‌بری می‌کند و بسته بعدی که باید اجرا شود را به ما برمی‌گرداند و همچنین می‌توانیم زمان ورودی بسته‌های جدید را به آن اضافه کنیم.
2. سیاست WRR: برای این سیاست نیز کلاسی به همین نام تعریف کردیم که مشابها از کلاس صف اصلی ارث‌بری می‌کند. مشابها دو تا تابع قبلی در اینجا نیز پیاده‌سازی شده است.
3. سیاست NPP: برای این سیاست نیز دقیقا کارهای مشابه توضیح‌داده‌شده انجام شده است.

* **روتر (Router)**

در این کد و شبیه‌سازی، اصل کار در واقع همین روتر است که دو Host را به یکدیگر متصل می‌کند و وظیفه مدیریت رویدادها را دارد. رویدادها در ادامه تعریف می‌شوند. بر اساس رویدادهای سیستم، اختصاص بسته‌ها و پردازش‌های مربوط به آن انجام می‌شوند. تابع اصلی که کل کارها را انجام می‌دهد نیز execute\_all\_tasks است که در ادامه به طور دقیق بررسی می‌شوند.

* رویداد (Event)

برای جلوبردن زمان در شبیه‌سازی چند راه را می‌توان پیش گرفت:

1. زمان را با یک اپسیلون زمانی جلو ببریم، مثلا زمان را 0.001 ثانیه – 0.001 ثانیه جلو ببریم. این روش مشخصا بهینه نیست چراکه در بسیاری از زمان‌ها هیچ کاری قرار نیست کاری انجام شود. همچنین پیاده‌سازی کد نیز سخت می‌شود.
2. زمان را بر اساس یک سری رویداد جلو ببریم. رویداد (Event) از 2 نوع است:

1.2. حداقل یک بسته‌ای بوده که اجرایش تمام شده است و در نتیجه باید سراغ صف برویم و در صورتی که صف خالی نیست، بر اساس سیاست اولویت‌دهی‌ای که داریم به آن پردازنده خالی‌شده یک بسته اختصاص دهیم.

2.2. بسته جدیدی وارد سیستم می‌شودکه نیاز است تا بررسی شود که این بسته جدید واردشده را به کجا اختصاص دهیم.

از این 2 سیاست گفته شده، ما مورد دوم را پیاده‌سازی کرده‌ایم و در نتیجه نیاز است که موجودیتی (کلاسی) به نام EventType تعریف کنیم که 2 مقداری بگیرد که به معنای هر یک از این 2 رویداد معرفی شده در 1.2 و 2.2 است.

**نحوه پیاده‌سازی**

1. اعداد مربوط به زمان‌های میان‌ورودی تولید می‌شوند (از توزیع پوآسون).
2. اعداد مربوط به زمان‌های اجرا تولید می‌شوند (از توزیع نمایی).
3. ایجاد روتر و صدا زدن تابع execute\_all\_tasks از همین کلاس.
4. تعیین تمام زمان‌های ورودی بر اساس زمان‌های میان‌ورود به کمک تابع set\_all\_arrivals.
5. تا وقتی که زمان شبیه‌سازی تمام نشده، کارهایی که در ادامه می‌آیند انجام شود و در غیراینصورت شبیه‌سازی را تمام کنیم (از گام 9 ادامه دهیم).
6. نوع رویداد بعدی و زمان رویداد بعدی را به کمک تابع handle\_and\_get\_next\_event بگیریم.
7. اگر نوع رویداد از نوع تمام‌شدن زمان اجرای بسته است، پردازندۀ خالی را به کمک تابع get\_first\_free\_processor بگیریم. سپس تسک بعدی را به کمک service\_policy.get\_next بگیریم و در صورتی که تسکی موجود بود، به کمک execute آن را اجرا کنیم.
8. اگر نوع رویداد از نوع آمدن بسته (تسک) جدید بود، اولین پردازنده خالی به کمک get\_first\_free\_processor دریافت می‌شود و در صورتی که چنین پردازنده‌ای نبود (همه پردازنده‌ها مشغول بودند)، آن را وارد صف می‌کنیم و در غیراینصورت آن را به کمک execute اجرا می‌کنیم.
9. در صورتی که در گام 5 از زمان شبیه‌سازی فراتر رفتیم، از اینجا ادامه می‌دهیم و تابع finish\_all را صدا می‌کنیم.
10. در صورتی که تسکی شروع به اجرا کرده ولی تمام نشده (به دلیل اتمام زمان شبیه‌سازی)، آن را به کمک تابع finish تمام می‌کنیم.

**جزئیات کد**

در ادامه در هر بخش، جرئیات نحوه پیاده‌سازی توابع را بررسی خواهیم کرد.

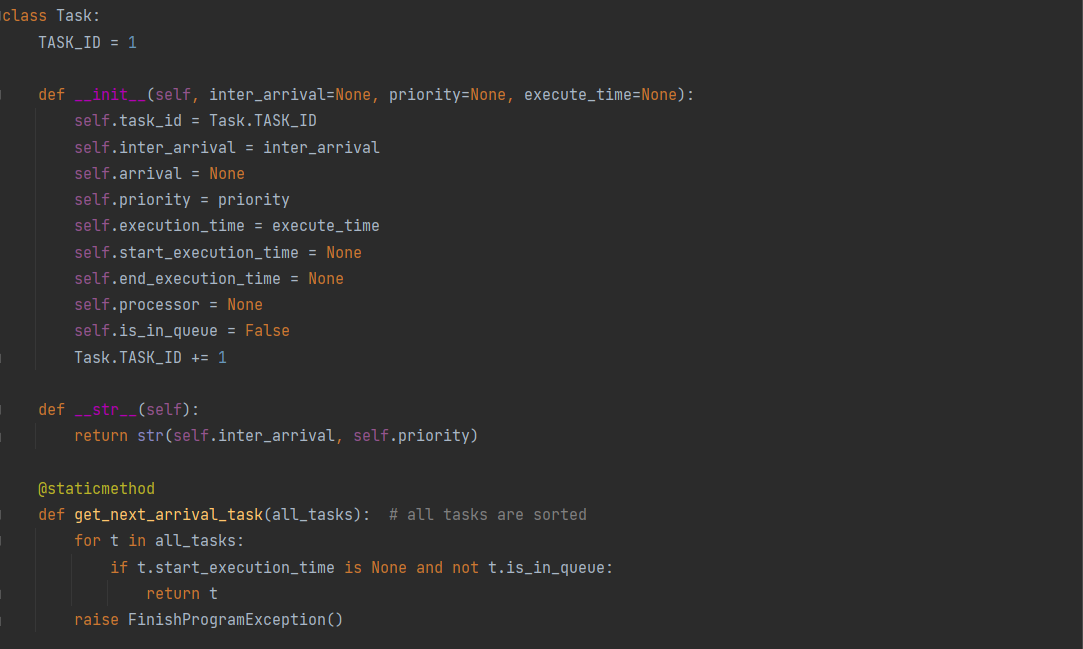
1. **موجودیت Task (بسته)**

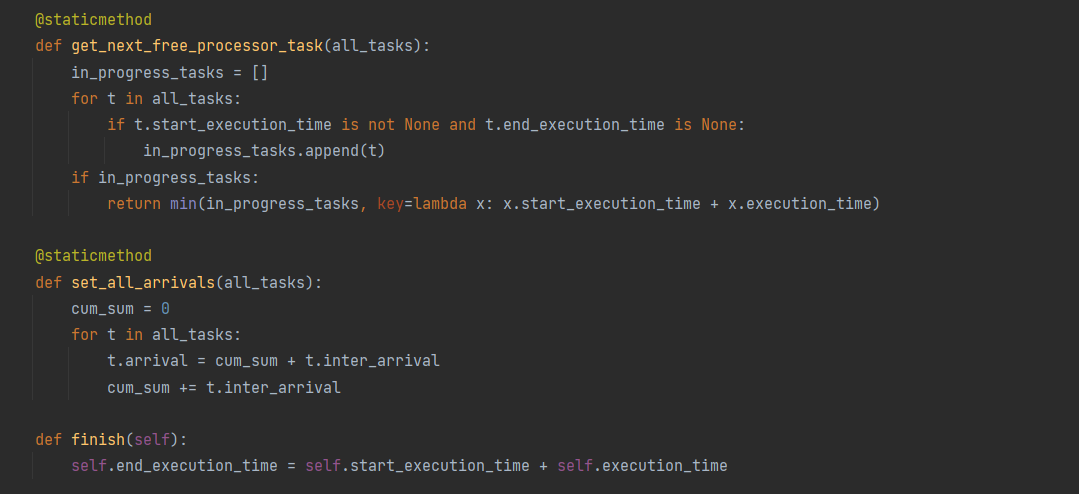
در اینجا میخواهیم کلاس Task که مربوط به بسته‌ها است را بررسی کنیم.

در هنگام نمونه‌گیری (instance گیری) از این کلاس، وارد \_\_init\_\_ شده و ویژگی های مربوط به آن مقداردهی می‌شوند.

* شناسه بسته (task\_id) به کمک یک متغیر static که در کلاس تعیین شده و یک شمارنده است مشخص میشود.
* زمان میان‌ورودی (interarrival time) به عنوان پارامتر ورودی داده می‌شود.
* متغیر arrival نیز در ابتدا خالی گذاشته می‌شود تا بعدا که زمان ورود را تعیین کردیم آن را نیز درست کنیم.
* اولویت (priority) نیز یکی از متغیرهای ورودی است که مشخص میشود بسته از کدام یک از 3 اولویت است.
* زمان اجرا (execution\_time) نیز از متغیرهای ورودی است و زمانی است که اجرای بسته طول می‌کشد.
* زمان شروع به اجرا (start\_execution\_time) در ابتدا خالی گذاشته می‌شود تا در ادامه زمان شروع اجرا را تعیین کنیم (بستگی به پردازنده‌های خالی و وظایف (بسته‌های) دیگر دارد.
* پردازنده‌ای که قرار است بر روی آن اجرا شود (processor) را در ابتدا خالی می‌گذاریم تا بعدا مشخص شود. دلیل خالی گذاشتن آن نیز مشابه start\_execution\_time است.
* مقدار بولین (متغیر دو حالتی) در ابتدا False گذاشته می‌شود چراکه هنوز معلوم نیست بسته زمان فراخوانی یا همان arrival اش رسیده باشد، در واقع باید منتظر بمانیم تا وارد سیستم شود.
* در انتها، شمارنده مربوط به تعداد بسته‌ها (TASK\_ID) را یکی زیاد می‌کنیم که در واقع یک Auto\_Counter است.



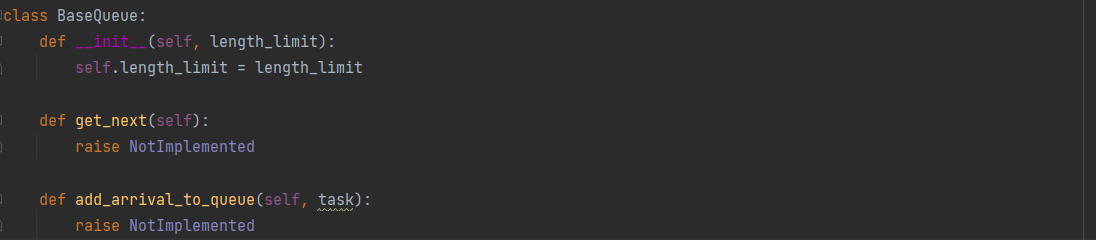


* تابع \_\_str\_\_ برای این است که زمانی که میخواهیم از یک instance به عنوان رشته استفاده کنیم (مثلا هنگام پرینت)، بدانیم چه چیزی باید return شود.
* تابع get\_static\_method که یک تابع static است (مربوط به کل کلاس است و نه یک instance خاص)، به ما تسک بعدی‌ای که باید اجرا شود را برمی‌گرداند. توجه کنید که برای اینکه بفهمیم چه تسکی را باید برگردانیم، بر روی تمام آنها loop زده و اگر تسکی زمان شروع اجرایش تعیین نشده (یعنی هنوز شروع به اجرا نکرده) و در صف نیز نیست، آن را بازمی‌گرداند. اگر چنین تسکی موچود نبود نیز پایان اجرای برنامه را اعلام می‌کند (موجودیت FinishProgramException).

ادامه موجودیت تسک را بررسی میکنیم که در تصویر بالا میبینیم.

* تابع get\_next\_free\_processor\_task به ما کمترین زمان مورد نیاز تا اتمام حداقل یکی از تسک ها را برمیگرداند. یعنی ابتدا تمام تسک های در حال اجرا را پیدا میکند بدین صورت که اگر تسکی زمان شروع به اجرایش مشخص است اما زمان اتمامش مشخص نیست، یعنی هنوز تمام نشده است، پس به آرایه in\_progress اضافه میکنیم. سپس میان تمام تسک‌های فعال، کمترین زمان پایان را برمیگردانیم. زمان پایان نیز به کمک جمع مقدار «زمان شروع به اجرا» و «زمان اجرا» به دست می‌آید.
* تابع set\_all\_arrivals، زمان مطلق ورود هر تسک به سیستم (absolute arrival time) را تعیین می‌کند. این کار به کمک زمان میان‌ورود (interarrival times) تعیین می شوند. توجه کنید که اعداد تصادفی‌ای که ما در ابتدا تولید می‌کنیم، زمان میان‌ورود است، یعنی فاصله میان 2 زمان ورود متوالی، پس باید با جمع تمجعی (cumulative sum)، زمان مطلق را حساب کنیم.
* تابع finish نیز به اجرای تسک پایان می‌دهد. برای پایان وظیفه نیز کافی است زمان end\_execution\_time را تعیین کنیم. همین که این مقدار دیگر None نباشد، یعنی اجرای تسک پایان یافته است (این را در تمام بخش‌های کد در نظر گرفته‌ایم).

1. **موجودیت BaseQueue**

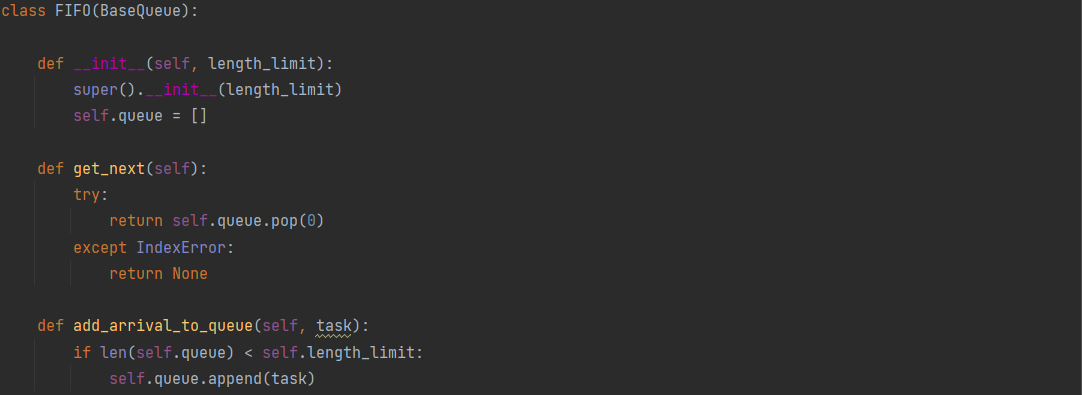


این موجودیت، همانطور که در بخش‌های ابتدایی گزارش بررسی شد، صف اصلی سیستم ما است که تمام 3 سیاست صف از آن ارث‌بری می‌کنند.

* تابع \_\_init\_\_: زمانی که instance ای ساخته می‌شود، صرفا ویژگی‌ای با عنوان length\_limit تعیین می‌شود تا اگر حجم صف پر شده بود، تسک‌های جدید drop شده و اضافه نشوند.
* توابع get\_next و add\_arrival\_to\_queue فعلا به صورت NotImplemented هستند تا در کلاس‌های فرزندشان بسته به کارکردشان پیاده‌سازی شوند. در کلاس‌های مربوط به خودشان بررسی خواهند شد.

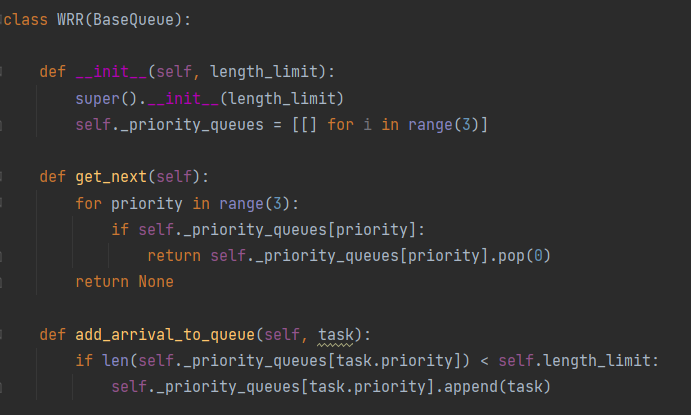
1. **موجودیت FIFO**

* تابع \_\_init\_\_: هنگام instance گیری وارد این تابع میشویم و ابتدا تابع پدر (BaseQueue) را مقداردهی اولیه میکنیم. سپس یک صف خالی ایجاد میکنیم.
* تابع get\_next: وظیفه این تابع، برگرداندن تسک بعدی از میان تسک های موجود در صف، بر اساس سیاست First In First Out یا همان First Come First Serve است. برای این کار در صورتی که صف خالی باشد (تسکی نداشته باشیم)، None برمیگردانیم، در غیراینصورت، اولین عنصر صف را برمیگردانیم و در واقع pop میکنیم.
* تابع add\_arrival\_to\_queue: وظیفه این تابع، اضافه کردن تسک جدید به صف است. البته در صورتی این اتفاق می‌افتد که به حداکثر طول صف نرسیده باشیم، که در این صورت تسک drop شده و اصلا به صف اضافه نمی‌شود.



1. **موجودیت WRR**

* تابع \_\_init\_\_: مشابه با constructor مربوط به FIFO است با این تفاوت که 3 صف ایجاد می‌کند چراکه در الگوریتم Weighted Round Robin نیاز به صف‌های مجزا برای هر یک از اولویت‌ها داریم.
* تابع get\_next: وظیفه‌اش برگرداندن تسک بعدی‌ای است که طبق این الگوریتم اجرا میشود. برای این کار، از بااولویت‌ترین صف شروع کرده (صف شماره صفر که بالاترین اولویت را دارد) و در صورتی که خالی نبود، از صف با اولویت بالاتر برداشت می‌کند.
* تابع add\_arrival\_to\_queue: مشابه با همین تابع در FIFO است با این تفاوت که به صف مربوط به اولویت خودش اضافه می‌شود.



1. **موجودیت NPPS**

* تابع \_\_init\_\_ آن که دقیقا مشابه با همین تابع در FIFO است.
* تابع get\_next نیز دقیقا همان تابع در FIFO است.
* تابع add\_arrival\_to\_queue بدین گونه تعریف شده است که اگر طول صف به ماکسیمم خودش نرسیده بود، تسک را به صف اضافه می‌کند اما صف را نیز بر اساس اولویت و سپس بر اساس زمان ورودی مرتب میکند (در کد نهایی، زمان ورود در نظر گرفته شده نه میان ورود).



1. **موجودیت EventType**

* این موجودیت در مراحل قبلی گزارش بررسی شده است، اما به طور کلی تعیین میکند که نوع رویدادی که در سیستم رخ داده چیست.

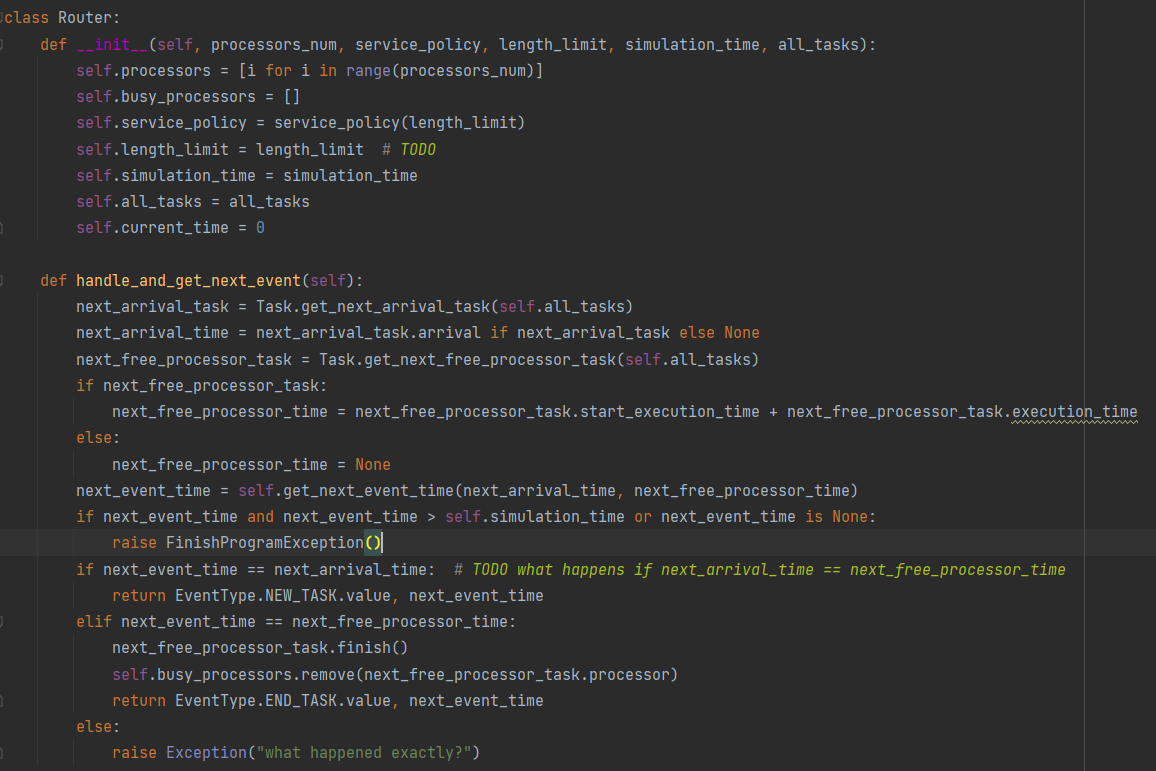


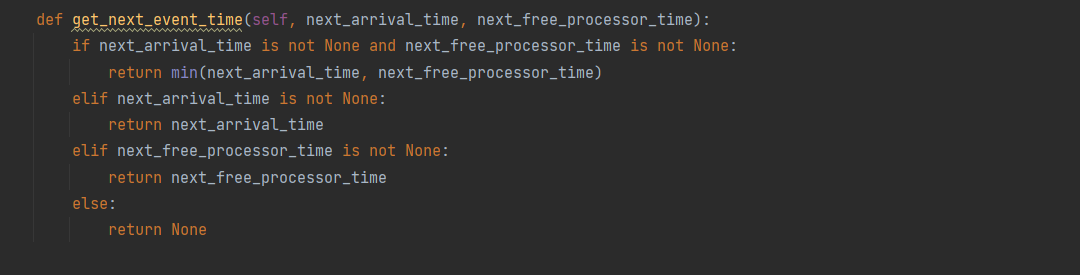
1. **موجودیت Router**

* این موجودیت در بخش «نحوه پیاده‌سازی» بررسی شده است اما اینجا نیز بررسی خواهد شد.
* تابع \_\_init\_\_ بدین صورت کار میکند که تمام مقادیر مربوط به روتر مقداردهی میشوند. پردازنده ها به تعدادی که در ورودی داده شده تولید میشوند، آرایه busy\_processors ایجاد میشود که پردازنده های مشغول به کار را مشخص میکند که در ابتدا خالی است، صف بر اساس سیاستی که در ورودی تعیین شده و به حداکثر طولی که در ورودی داده شده ایجاد میشود (صف سیاست دار یکی از FIFO – WRR – NPP) است. مدت زمان کل شبیه سازی که روتر باید درگیر باشد مشخص میشود و همچنی تمام تسک ها و زمان فعلی (که در ابتدا صفر است) مشخص میشوند.
* تابع handle\_and\_get\_next\_event: ابتدا تسک بعدی ای که از راه میرسد و وارد سیستم میشود را دریافت میکنیم و زمان ورود آن را نیز ذخیره میکنیم. سپس پردازنده خالی بعدی را دریافت میکنیم. در صورتی که پردازنده خالی بعدی داشته باشیم، زمانی که خالی میشود را حساب میکنیم. در غیر اینصورت این مقدار را None میگذاریم.

حالا زمان آن است که تابع get\_next\_event\_time را فراخوانی کنیم تا بر اساس نزدیکترین زمان خالی شدن پردازنده ها و نزدیکترین زمان آزادشدن تسک جدید، به ما بگوید رویداد بعدی کی قرار است رخ دهد.

حال در صورتی که زمان ایونت بعدی نامشخص بود (یعنی دیگر ایونتی نداشتیم) و یا زمان ایونت بعدی از مدت زمان شبیه سازی ما فراتر میرفت، پایان شبیه سازی را اعلام میکنیم.در غیراینصورت مطمئن هستیم که قرار است رویدادی رخ دهد و تنها لازم است که نوع آن را مشخص کنیم.

در صورتی که از نوع خالی شدن پردازنده باشد، پردازنده را خالی کرده و آن را از لیست پردازنده های مشغول خارج میکنیم. همچنین نوع رویداد بعدی و زمان وقوع رویداد بعدی را return میکنیم.

* تابع get\_next\_event\_time: زمان رویداد بعدی را برمیگرداند. اگر هم تسک جدیدی دارد وارد میشود و هم پردازنده ای دارد خالی میشود، در این صورت مینیمم آنها را برمیگردانیم، در غیر اینصورت آن که None نیست را برمیگردانیم. اگر هیچ یک نبود نیز خالی برمیگردانیم.
* کدهای اینجا نیز قبلتر بررسی شده اما مجددا توضیحاتی داده میشود.

تا زمانی که زمان شبیه سازی تمام نشده است، زمان را خرده خرده جلو میبریم. زمان را بر اساس رویکرد دومی که در بخش های ابتدایی گزارش توضیح دادیم جلو میبریم بدین شکل که بر اساس رویدادها (Event) جلو میرویم.

تابع handle\_and\_get\_next\_event صدا زده میشود و رویداد بعدی را به همراه نوع آن دریافت میکند.

در صورتی که رویداد از جنس ورود تسک جدید بود، به دنبال یک پردازنده جدید میگردیم تا آن تسک را به آن پردازنده اختصاص دهیم.

تسک را نیز به کمک get\_next\_arrival\_task دریافت میکنیم.

در صورتی که پردازنده ای موجود بود، اجرای تسک را به کمک execute شروع میکنیم.

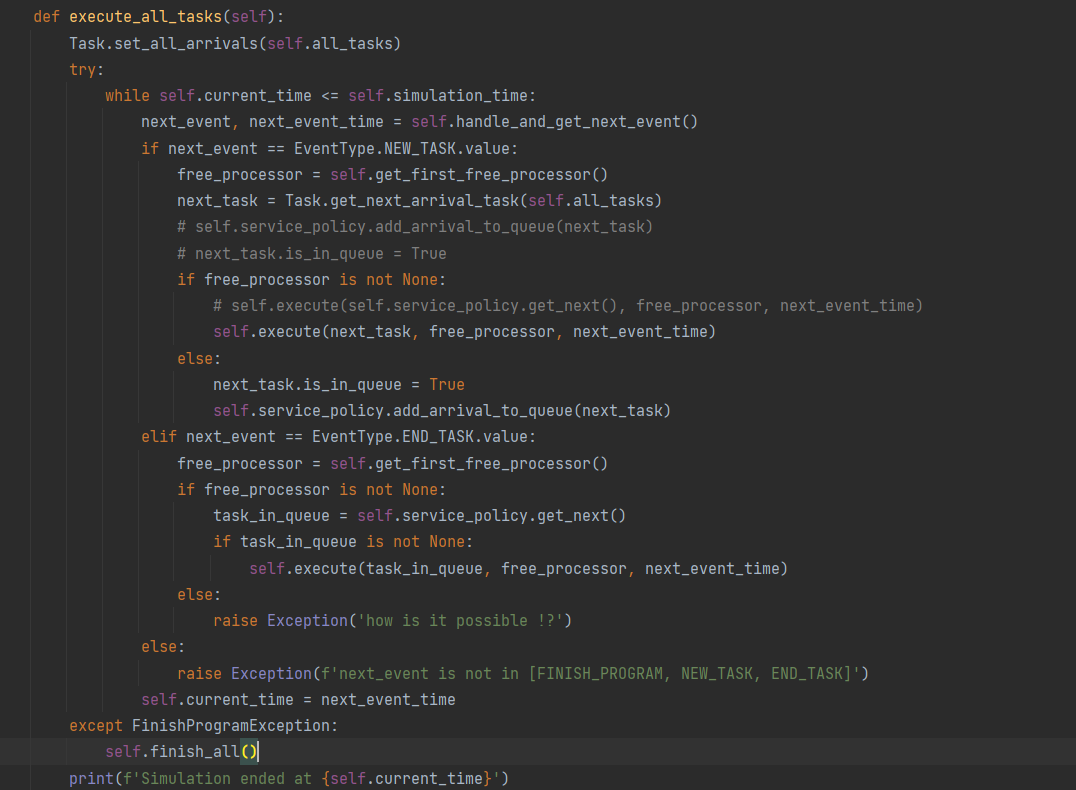
در صورتی که پردازنده ای موجود نبود، تسک را بر اساس policy ای که داریم (یکی از FIFO-WRR-NPPS) به صف اضافه میکنیم (طبعا اگر ظرفیت صف پر نشده باشد).

در صورتی که رویداد از نوع خالی شدن پردازنده باشد، شماره پردازنده را دریافت کرده و در صورتی که None بود Exception میدهیم چراکه وقتی وارد اینجا شدیم یعنی مطئنیم که پردازنده ای خالی بوده اما وقتی None میگیریم یعنی پردازنده خالی نداریم که این یعنی Exception.

در صورتی که خطا نداد، تسک بعدی ای که قرار است اجرا شود را دریافت کرده و آن را از صف گرفته (بر اساس سیاستی که داریم) و تسک را شروع به اجرا میکنیم.

زمان فعلی را نیز به زمان رویداد بعدی به روزرسانی میکنیم.

در صورتی که به Exception خوردیم نیز تمام اجرا را خاتمه میدهیم.

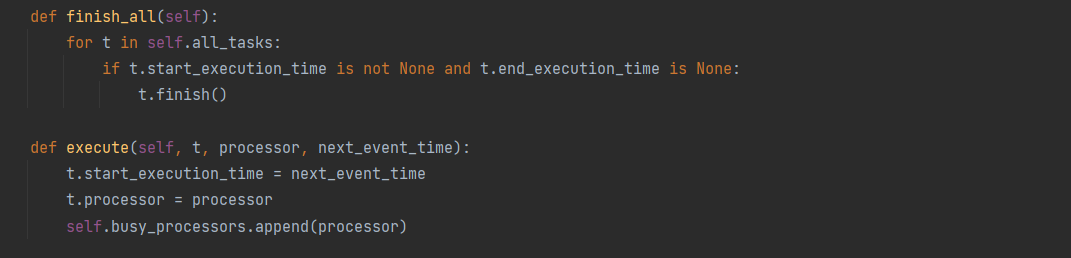


* در پایان این موجودیت نیز دو تابع زیر را مشاهده میکنیم.

تابع finish\_all اجرای تمام تسک های تمام نشده را تمام میکند.

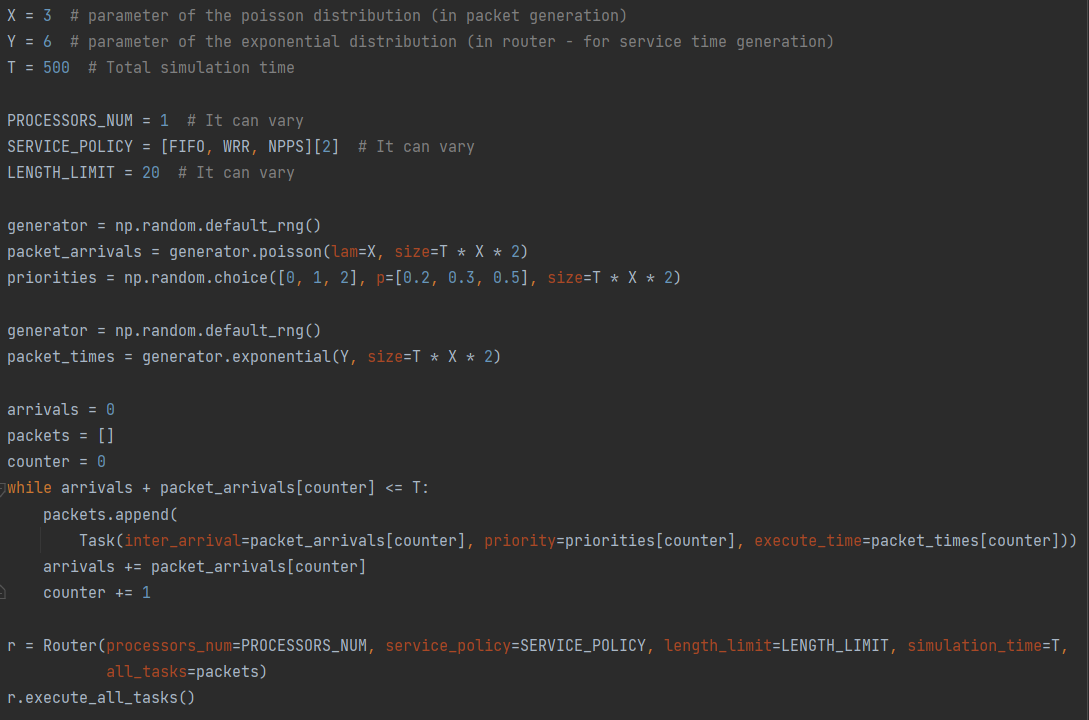
تابع execute نیز تسک را اجرا میکند. اجرای تسک نیز بدین معنی است که زمان شروع اجرای تسک را مشخص کنیم (همانطور که قبلتر توضیح دادیم، این بدین معنی است که اجرای این تسک شروع شده است) و همچنین پردازنده متناظر با آن را مشخص کنیم.

همچنین پردازنده ای که درگیر اجرای آن شده را به لیست پردازنده های درگیر اضافه کنیم.



1. اجرای شبیه سازی

* در قطعه کد زیر نیز، روند اجرا را مشاهده میکنیم که X و Y و Z همانطور که کامنت شده اند، پارامترهای توزیع پوآسون، توزیع نمایی و همچنین مدت زمان شبیه سازی هستند.
* تعداد پردازنده ها، سیاست اولویت دهی به تسک ها، و حداکثر طول صف نیز تعیین میشود.
* سپس زمان های میان ورود و اجرا از توزیع های مناسب خواسته شده تولید شده و packets تولید میشود.
* توجه کنید که تعداد بسته ها را بیش از میزان مورد نیاز در نظر گرفتیم تا هیچ گاه این اتفاق نیفتد که شبیه سازی زودتر از موعد تمام شود.
* البته برای اینکه اضافات این بسته ها دور ریخته شود، یک loop زده و تا زمانی که از زمان شبیه سازی فراتر نمیرویم این بسته ها را اضافه میکنیم.
* در نهایت Router را ایجاد کرده و شبیه سازی را شروع میکنیم.



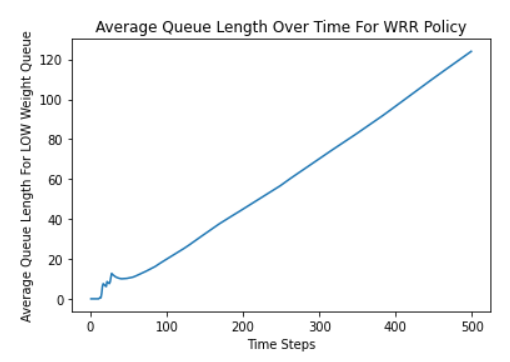
1. در نهایت مطابق با قطعه کد زیر، اطلاعات شبیه سازی در فایل csv ذخیره میشوند.

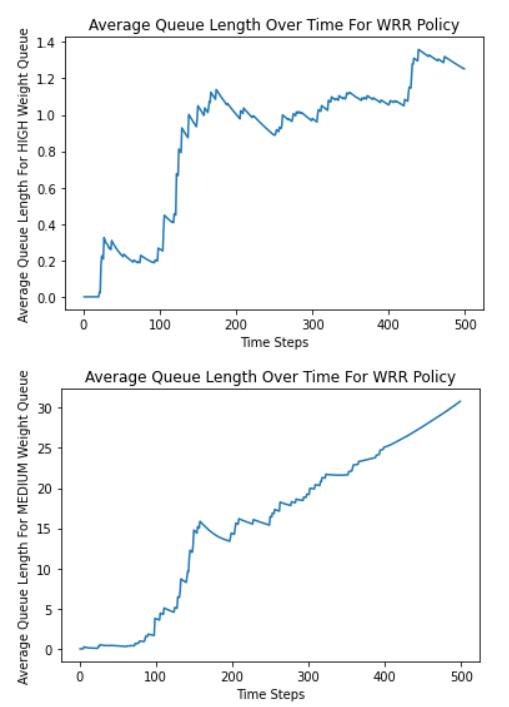
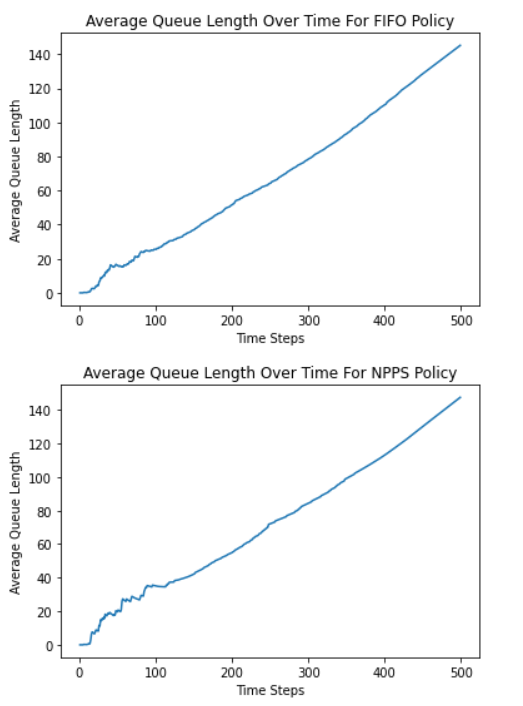
* به ازای تمام تسک ها، شناسه تسک، زمان میان ورود، زمان ورود، اولویت، زمان اجرا، زمان شروع اجرا، زمان پایان اجرا، شماره پردازنده ای که بر روی آن اجرا شده، و اینکه آیا اصلا به روتر ارسال شده یا نه، و اینکه آیا تسکی بوده که به دلیل پربودن صف drop شده، همگی در فایل اکسل ذخیره میشوند.



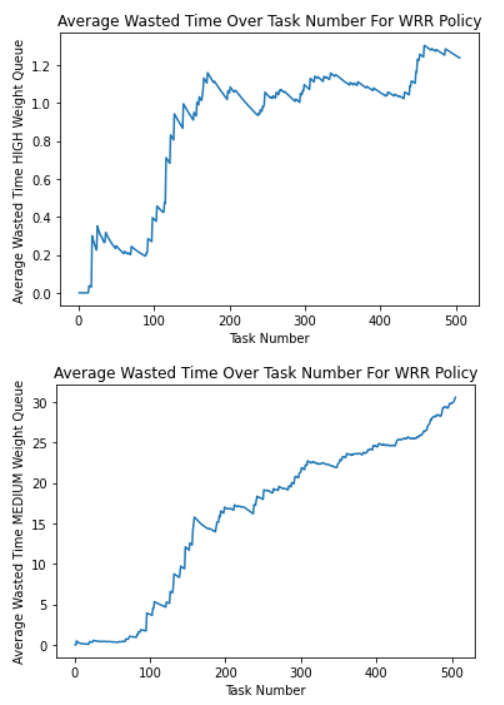
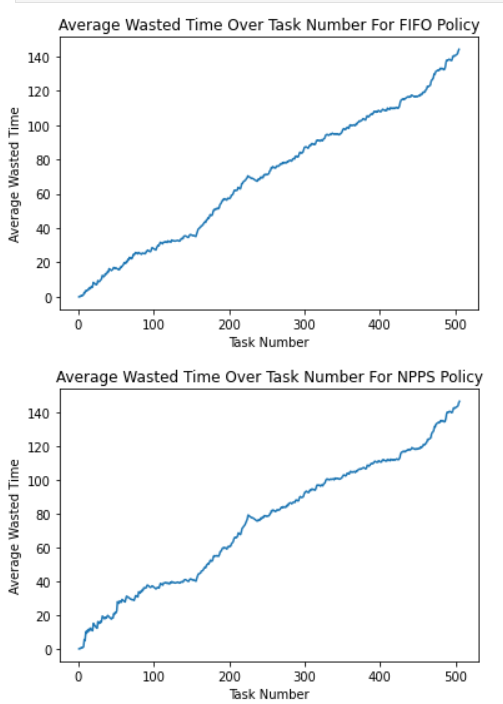
**خروجی‌ها و نتیجه‌گیری**

در پایان، خروجی های خواسته شده رسم میشوند.

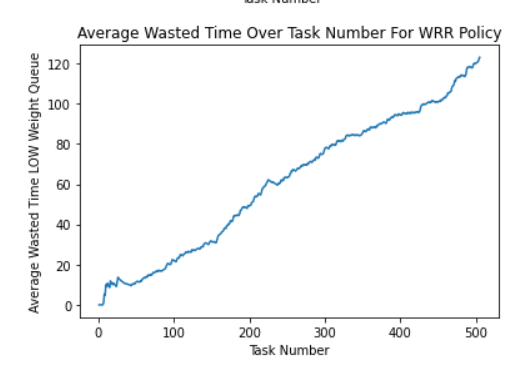
1. **متوسط طول صف**
   1. کد
   2. خروجی

**2. متوسط زمان تلف شده در همه صف ها**

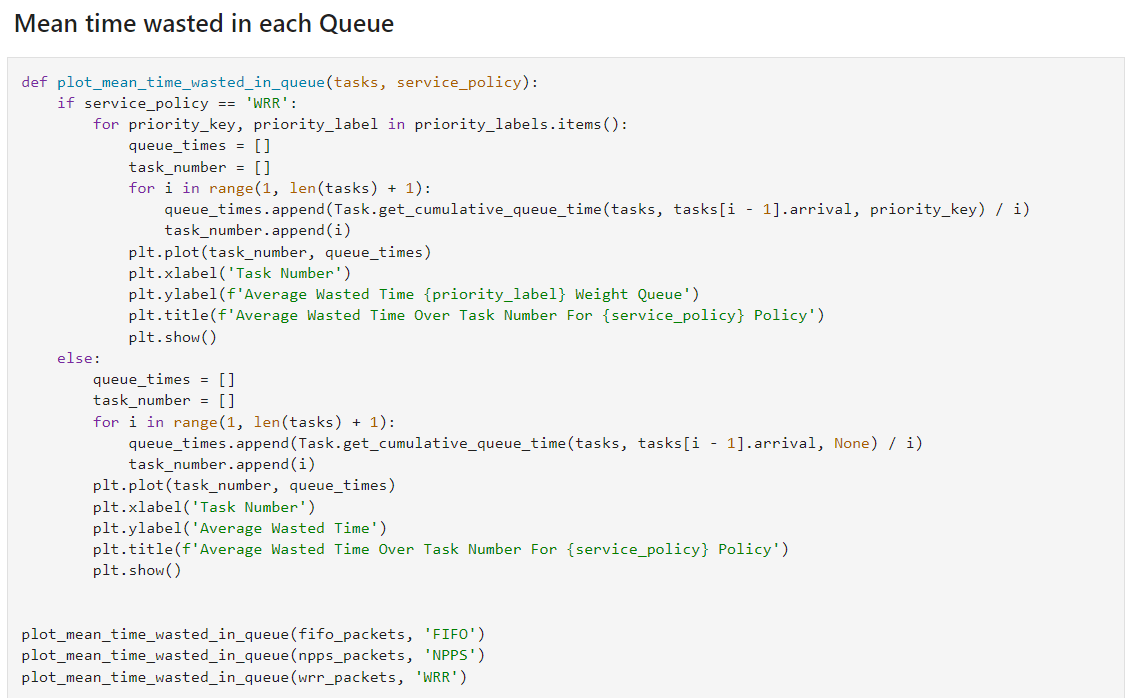
1.2. کد

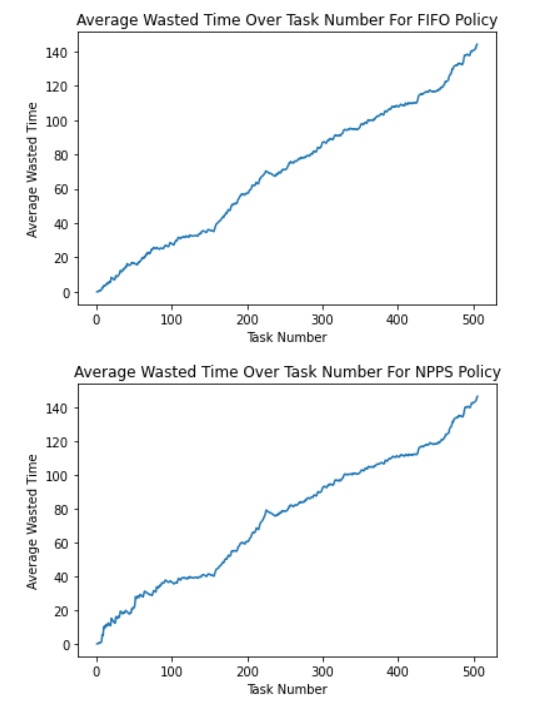
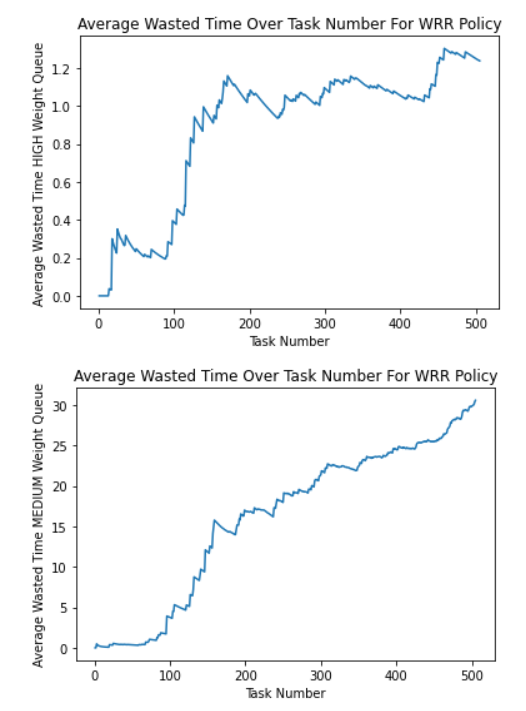


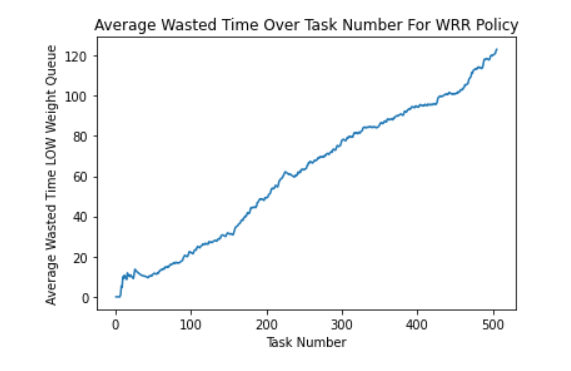
2.2. خروجی



**3. متوسط زمان تلف شده در هر صف**

1.3. کد

2.3. خروجی

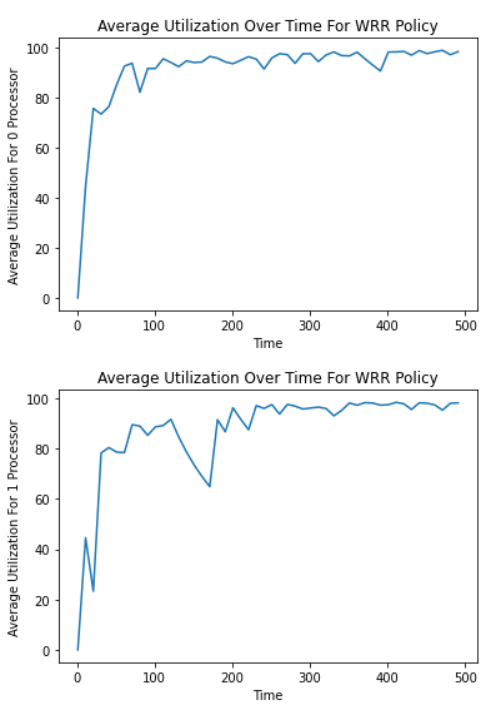
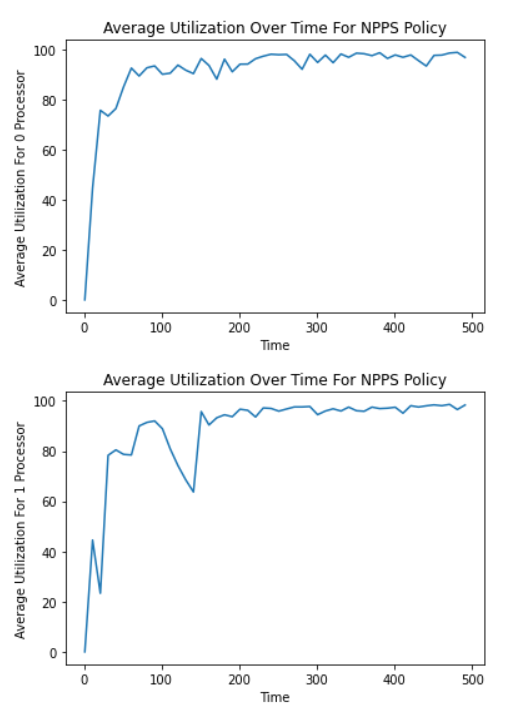
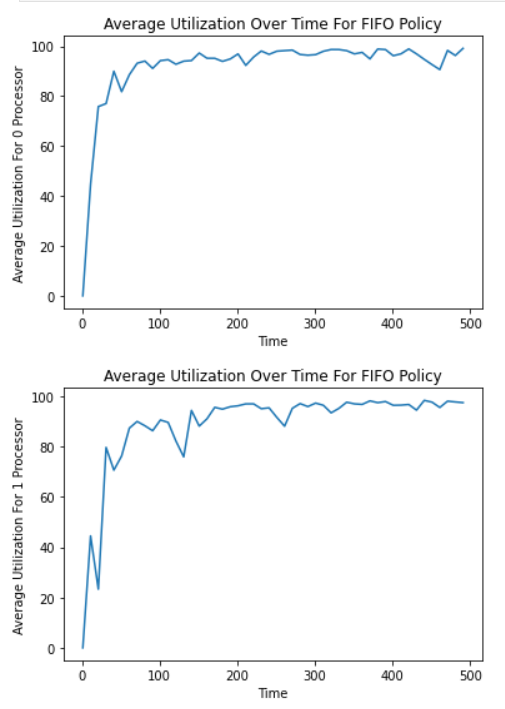


**4. متوسط Utilization هر پردازنده**

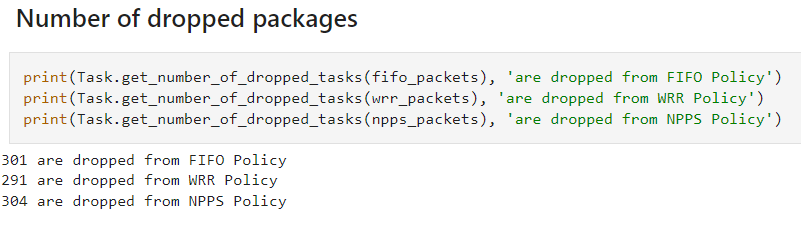
1.4. کد



2.4. خروجی

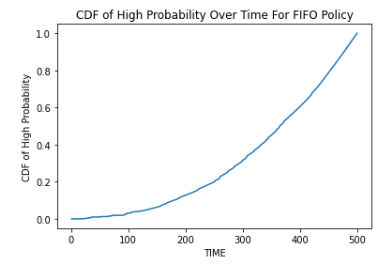


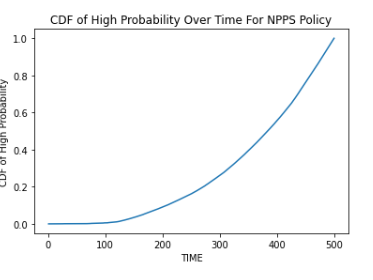
**5. تعداد بسته های drop شده**

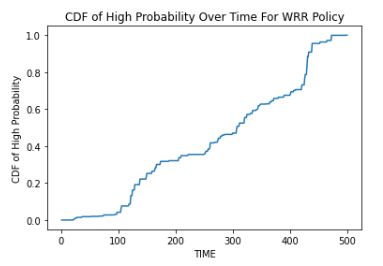


**6. نمودارهای CDF مربوط به تسک های با اولویت بالا**

1.6. کد

2.6. خروجی





**7. چگونه میتوانیم utilization سیستم را بالا ببریم؟**

\* تقسیم سنگینی (Load Balancing): تقسیم کردن مناسب تسک ها میان پروسسورها

\* پردازش موازی: از مولتی ترد و چندپردازشی استفاده کنیم تا از ریسورس های موجود حداکثر استفاده را بکنیم.

\* بهینه سازی منابع: اختصاص منابع را بهینه سازی کنیم. این کار به کمک تحلیل و بررسی عملکرد سیستم و مشخص کردن ریسورس هایی که کم یا زیاد از حد استفاده شده اند به دست می آید، بدین صورت میتوانیم توازن بهتری در تقسیم تسک ها نیز داشته باشیم که به همان Load Balancingنیز مرتبط میشود.

\* الگوریتم های زمانبندی: از الگوریتم های مختلف زمانبندی یا همان Task Scheduling های مختلف استفاده کنیم تا اولویت بندی مناسب کرده و بتوانیم زمان های بیکاری را کمینه و استفاده از منابع را بیشینه کنیم. این کار به کمک در نظر گرفتن موارد ی همچون وابستگی ها، اولویت ها و پیش نیازها قابل دستیابی است.

**8. با توجه به داده های به دست آمده، با در نظر گرفتتن دو معیار تعداد بسته هایی که drop میشوند و تعداد دسته هایی که به دست host میرسند، کدام یک از سیاست های نوبت دهی در روتر منطقی تر و بهتر است که استفاده کنیم؟ توضیح دهید.**

سیاست WRRبهتر است چراکه بسته های کمتری را drop میکند و سایر نتایج و خروجی های مقایسه ای نیز در نمودارهای بالا نمایش داده شده اند.

**منابع و تشکرها**

برای انجام این پروژه از مطالب درس شبیه سازی که در کلاس گفته شد استفاده کردیم.

همچنین جست و جوهای مختلفی نیز انجام دادیم و ودر صورت نیاز از گروه های دیگر نیز مشورت و همفکری گرفتیم.

با تشکر از استاد و تیم تدریس که ما را در انجام این پروژه یاری کردند.