پروژه شماره ۶: تخصیص وظایف آگاه به سطوح بحرانیت در سامانه‌های لبه

فروزان ایرجی (۹۹۱۰۵۲۷۲) و فاطمه‌السادات لاجوردی (۴۰۰۱۰۵۲۱۷)

نحوه‌ی پیاده‌سازی:

1. **تعریف مدل‌ها** (models.py)، شامل کلاس‌ها و فیلدها و توضیحات مربوط به هر فیلد به صورت زیر است:

Task:

***id**: int*
***number_of_clocks**: int*
***data_amount**: float*
***deadline**: int*
***criticality**: CriticalityLevel (Enum: S0, S1, S2, S3)*
***arrivel_time**: int به طور متوسط در هر ثانیه یک وظیفه وارد استیشن می‌شود*
***execution_time**: int = number_of_clocks / server.processing_frequency*
***productivity**: float = execution_time / deadline*
***response_time**: int*
***is_missed**: boolean آیا موقع اجرا، وظیفه به ددلاین خود می‌رسد یا نه*
***server**: Server سروری که وظیفه به آن اختصاص داده شده است*

- همچنین تاخیر ارسال به سرور برای هر وظیفه با فرمول *data_amount / server.data_transmission_rate* به دست می‌آید.
- فیلد استاتیک ***task_count_based_on_criticality*** هم به صورت یک دیکشنری تعریف شده که در هر شبیه‌سازی تعداد وظایف تعریف‌شده با سطح بحرانیت مشخص را ذخیره می‌کند.

Server:

***id**: int*
***processing_frequency**: float*
***data_transmission_rate**: float*
***number_of_cores**: int*
***productivity**: float مجموع بهره‌وری وظایفی که به سرور اختصاص داده شده است*
***number_of_assigned_tasks:** dict دیکشنری‌ای که تعداد وظایف اختصاص‌یافته به سرور را با توجه به سطح بحرانیت آن ذخیره می‌کند*
***assigned_tasks:** list لیست وظایفی که سرور باید آن‌ها را اجرا کند*

BaseStation:

***servers**: list لیست سرورهایی که استیشن باید وظایف را بین آن‌ها تقسیم کند*

2. پیاده‌سازی توابع:

- ***generate_tasks*:** در این تابع تعداد کل وظایف و سطح بحرانیت آن‌ها گرفته می‌شود و وظایف با فیلدهایی که مقدار آن‌ها به صورت رندم مشخص می‌شود، ساخته می‌شوند. اگر سطح بحرانیت به تابع داده نشود، مقدار آن به صورت رندم از بین مقادیر S0, S1, S2, S3 انتخاب می‌شود.
- ***generate_tasks_based_on_criticality*:** این تابع تعداد تسک‌ها را بر اساس سطح بحرانیت آن‌ها می‌گیرد و با صدا زدن تابع *generate_tasks کل وظایف را می‌سازد. (برای خروجی‌هایی که تعداد تسک‌ها را بر اساس سطح بحرانیت آن‌ها مشخص کرده است.)*
- ***initialize_base_station*:** این تابع یک instance جدید از استیشن می‌سازد (هر شبیه‌سازی‌ای استیشن خود را دارد) و بعد به تعداد سرورهای داده‌شده در آرگومان تابع، سرورها را با مقادیر فیلدهای رندم می‌سازد و به استیشن اضافه می‌کند.
- ***simulate*:** در این تابع ابتدا استیشن و سرورها تعریف شده‌اند و بعد از آن استیشن، وظایف را بر اساس معیارهای مشخص‌شده در شرح پروژه بین سرورها تقسیم کرده و در نهایت هر سروری وظایف تخصیص‌یافته به خود را اجرا می‌کند تا خروجی‌های response time, deadline miss ratio را بدست آورد.
- ***create_schedule_table*:** در این تابع از استیشن، وظایف به سرورها با توجه به معیارهای مشخص‌شده در شرح پروژه اختصاص یافته می‌شوند. روی لیست وظایف ساخته‌شده پیمایش می‌شود و ابتدا سرورهای موجود را بر اساس بهره‌وری‌شان مرتب می‌کنیم. اگر بهره‌وری سرور + بهره‌وری وظیفه کمتر مساوی از تعداد هسته‌های سرور باشد آن وظیفه به سرور اختصاص می‌یابد، در غیر این صورت به این معنی است که سرور نمی‌تواند وظیفه را در موعد زمانی مقرر به اتمام برساند. در این صورت دو حالت داریم:
  - در نظر گرفتن سطح بحرانیت وظیفه: وظیفه را به سروری اختصاص می‌دهیم که کمترین تعداد وظیفه‌ی تخصیص‌داده‌شده با سطح بحرانیت بیشتر مساوی وظیفه‌ی فعلی را دارد.
  - نادیده گرفتن سطح بحرانیت وظیفه: وظیفه را به سروری با کمترین میزان بهره‌وری اختصاص می‌دهیم.

- ***execute_tasks:*** در این تابع از سرور، وظایف به ترتیب زمان رسیدن‌شان به استیشن (با در نظر گرفتن تاخیر ارسال به سرور) و با تعیین کردن اینکه سطح بحرانیت در اجرای وظایف در نظر گرفته شود یا نه، اجرا می‌شوند.

3. **نحوه‌ی اجرای برنامه**: برنامه از ***main*** شروع می‌شود که در ابتدای آن تعداد سرورهای مورد نظر را مشخص می‌کنیم (***number_of_servers***) و بعد به ترتیب با توجه به خروجی‌های مورد نظر، ابتدا وظایف را با تابع ***generate_tasks*** تولید می‌کنیم و بعد تابع ***simulate*** روی تسک‌ها اجرا می‌کنیم و در نهایت هم تابع plot را با توجه به خروجی مورد نظر (Average response time, Deadline miss ratio) صدا می‌زنیم تا نمودارهای خروجی را نشان دهد. (برای مجموعه وظایف متفاوت باید تابع simulate را جداگانه اجرا کنیم)