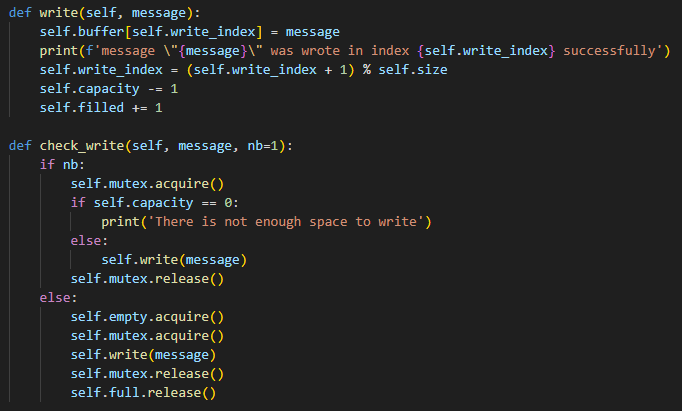
ابتدا یک کلاس Bounded Buffer تعریف میکنیم که در آن قطعات همگام سازی توسط تابع هایی درون آن پیاده سازی شده است. (در ادامه توضیح داده خواهد شد)

در تابع main ابتدا تعداد نویسنده ها و تعداد خواننده ها و تعداد بار هایی که قرار است از برنامه و Buffer اعلام وضعیت بگیریم و در نهایت حجم buffer را از کاربر میگیریم، سپس از کاربر میپرسیم که آیا نیاز به لود کردن از دیسک(فایل) داریم یا خیر که اگر نیاز به لود داشتیم این فرایند از فایل خوانده می­شود و در بافر نوشته می شود.

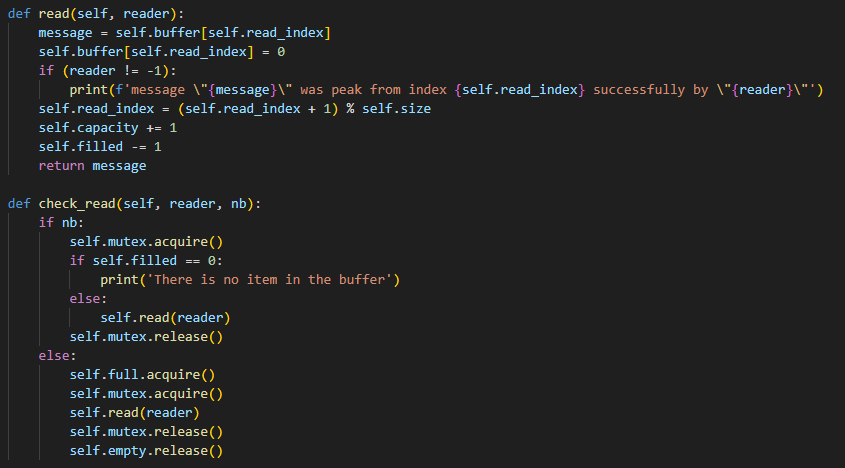
در نهایت به تعداد نویسنده ها یک عدد تصادفی که تعداد دفعات فعالیت هر یک از نویسنده ها می باشد، تولید می­شود این یعنی هر نویسنده قرار است چند مرتبه فرآیند نوشتن را انجام دهد که این مقدار در بازه [MIN\_TIMES, MAX\_TIMES] می باشد. در نهایت به ازای هرکدام از نویسنده ها یک نخ (thread) اختصاص داده می­شود که تابع send را اجرا کند و منتظر اجرا شدن میمانند. این عمل عینا برای خواننده ها هم اتفاق می افتد که توسط تابع get انجام خواهد شد. در انتهای کار هم به تابعی که قرار است status بگیرد یک نخ اختصاص میدهیم که تابع get\_status را اجرا خواهد کرد، حال تمام نخ ها را در یک لیست میریزیم و آن ها را به طور تصادفی کنار هم در یک لیست می­چیند( بُر میزند) این به این علت است که هنگام فرآیند خواندن و نوشتن نویسنده ها و خواننده ها به طور کاملا تصادفی کنار همدیگر قرار بگیرند و با هم تعامل داشته باشند. در نهایت از کاربر می­پرسیم که نیاز به ذخیره در دیسک(فایل) دارد یا خیر و برنامه را می بندیم.

در ادامه برای سه تابع get\_status, send, get که هرکدام به ترتیب به توابع stats, check\_write, check\_read اشاره می­کنند، به ترتیب به تعداد مرتبه هایی که قرار است اجرا شوند در یک حلقه اجرا می­شوند. تنها نکته ای که برای تابع send و get باید به آن اشاره کرد، مقدار rc می­باشد که بین یک و صفر مقداری را به طور تصادفی انتخاب میکند، اگر مقدار صفر باشد تابع blocking و اگر یک باشد در حالت non-blocking اجرا می­شود( برای جلوگیری از deadlock می توان مقدار آن را خالی گذاشت که به طور پیشفرض با مقدار 1 مقدار دهی می شود. و در نهایت هر تابع به مقدار کمی sleep می­شود که اولویت برای بقیه نخ ها فراهم شود.

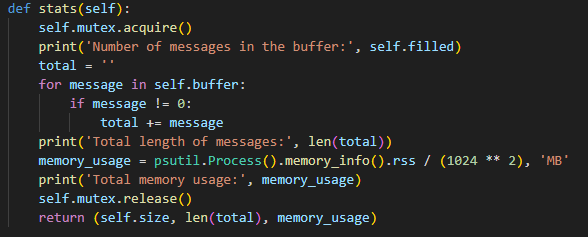
در ادامه به بررسی توابع پیاده سازی شده در کلاس Bounded Buffer می­پردازیم...



تابع check\_write بررسی می­کند که طریقه فراخوانی شده non-blocking است یا blocking، اگر حالت اول بود ابتدا با سمافور mutex آرایه را از در اختیار قرار دادن برای بقیه نخ ها محافظت میکند، سپس مقدار ظرفیت بافر را بررسی می کند اگر مقدار آن برابر صفر بود بعنی بافر پر شده است و نیاز به صبر کردن ندارد و منتظر نمی ماند یا در نهایت با تابع write در بافر پیام را می نویسد، اگر حالت دوم بود ابتدا با سمافور empty منتظر میماند و اگر فضا بود و بافر در اختیار نخ دیگری نبود، با تابع write پیام نوشته می­شود. تابع write صرفا در بافر پیام را می نویسد و همگام سازی این برنامه با تابع check\_write است.



روند کار توابع check\_read و read هم دقیقا مانند توابع check\_write و write می­باشد. با این تفاوت که در انتهای تابع read مقدار message برای استفاده های ثانویه برگردانده می­شود.



تابع stats در ابتدا بافر را برای جلوگیری از ورود بقیه نخ ها می بیندد، سپس با استفاده از ویژگی filled تعداد پیام های موجود در بافر را چاپ می­کند در انتها مجموع طول پیام ها را محاسبه و در انتها با استفاده از توابع سطح سیستم و فرآیندها، مقدار حافظه مصرف شده را به مگابایت برمیگرداند، در نهایت بافر را آزاد میکند و این سه مقدار را برمیگرداند

از آنجا که تست های این برنامه کاملا خودکار بوده نتایجی که در این گزارش آمده است، می تواند با نتایج زنده متفاوت باشد.

تست اول:

A screenshot of a computer

Description automatically generated

در ادامه دستور ذخیره در دیسک را دادیم که فایل به صورت زیر تبدیل شد

A screenshot of a computer screen

Description automatically generated

تست دوم:



A computer screen shot of a black screen

Description automatically generated

همانطور که مشاهده می­شود در بعضی جاها برنامه non-blocking عمل کرده است و پیام هایی ناشی از عدم وجود پیام برای خواندن یا عدم وجود حافظه برای نوشتن موجود است.

تست سوم:

A computer screen with white text

Description automatically generated

فایل بعد از ذخیره سازی:

A screen shot of a computer screen

Description automatically generated