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 ۱ 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...

```
def write(self, message):
    self.buffer[self.write_index] = message
    print(f'message \"{message}\" was wrote in index {self.write_index} successfully')
    self.write_index = (self.write_index + 1) % self.size
    self.capacity -= 1
    self.filled += 1

def check_write(self, message, nb=1):
    if nb:
        self.mutex.acquire()
        if self.capacity == 0:
            print('There is not enough space to write')
        else:
            self.write(message)
        self.mutex.release()
    else:
        self.empty.acquire()
        self.mutex.acquire()
        self.write(message)
        self.mutex.release()
        self.full.release()
```

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.

```
def read(self, reader):
    message = self.buffer[self.read_index]
    self.buffer[self.read_index] = 0
    if (reader != -1):
        print(f'message \"{message}\" was peak from index {self.read_index} successfully by \"{reader}\"')
    self.read_index = (self.read_index + 1) % self.size
    self.capacity += 1
    self.filled -= 1
    return message

def check_read(self, reader, nb):
    if nb:
        self.mutex.acquire()
        if self.filled == 0:
            print('There is no item in the buffer')
        else:
            self.read(reader)
        self.mutex.release()
    else:
        self.full.acquire()
        self.mutex.acquire()
        self.read(reader)
        self.mutex.release()
        self.empty.release()
```

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.

```
def stats(self):
    self.mutex.acquire()
    print('Number of messages in the buffer:', self.filled)
    total = ''
    for message in self.buffer:
        if message != 0:
            total += message
    print('Total length of messages:', len(total))
    memory_usage = psutil.Process().memory_info().rss / (1024 ** 2), 'MB'
    print('Total memory usage:', memory_usage)
    self.mutex.release()
    return (self.size, len(total), memory_usage)
```

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:

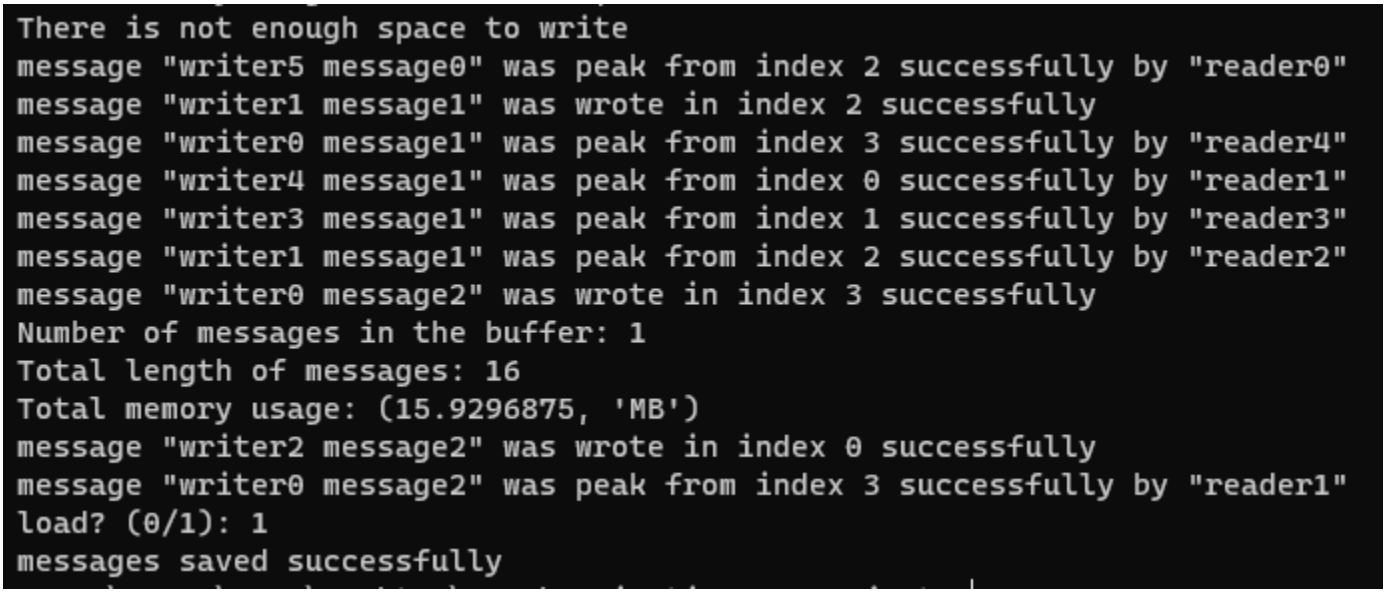
```
Writers: 3
Readers: 2
Stats: 2
Buffer size: 4
load? (0/1): 0
Benchmark:
writer id = 0 times = 2
writer id = 1 times = 1
writer id = 2 times = 2
reader id = 0 times = 1
reader id = 1 times = 1
Number of messages in the buffer: 0
Total length of messages: 0
Total memory usage: (15.76171875, 'MB')
message "writer1 message0" was wrote in index 0 successfully
message "writer1 message0" was peak from index 0 successfully by "reader0"
message "writer2 message0" was wrote in index 1 successfully
message "writer2 message0" was peak from index 1 successfully by "reader1"
message "writer0 message0" was wrote in index 2 successfully
Number of messages in the buffer: 1
Total length of messages: 16
Total memory usage: (15.828125, 'MB')
message "writer2 message1" was wrote in index 3 successfully
message "writer0 message1" was wrote in index 0 successfully
load? (0/1): 1
```

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

```
writer0 message0
writer2 message1
writer0 message1
```

تست دوم:

```
Writers: 7
Readers: 5
Stats: 3
Buffer size: 4
load? (0/1): 1
message "writer0 message0" was wrote in index 0 successfully
message "writer2 message1" was wrote in index 1 successfully
message "writer0 message1" was wrote in index 2 successfully
Benchmark:
writer id = 0 times = 3
writer id = 1 times = 2
writer id = 2 times = 3
writer id = 3 times = 2
writer id = 4 times = 2
writer id = 5 times = 1
writer id = 6 times = 1
reader id = 0 times = 2
reader id = 1 times = 3
reader id = 2 times = 3
reader id = 3 times = 2
reader id = 4 times = 2
message "writer0 message0" was wrote in index 3 successfully
There is not enough space to write
message "writer0 message0" was peak from index 0 successfully by "reader2"
Number of messages in the buffer: 3
Total length of messages: 48
Total memory usage: (15.83984375, 'MB')
message "writer3 message0" was wrote in index 0 successfully
message "writer2 message1" was peak from index 1 successfully by "reader0"
message "writer2 message0" was wrote in index 1 successfully
There is not enough space to write
There is not enough space to write
message "writer0 message1" was peak from index 2 successfully by "reader3"
message "writer5 message0" was wrote in index 2 successfully
message "writer0 message0" was peak from index 3 successfully by "reader4"
message "writer3 message0" was peak from index 0 successfully by "reader1"
message "writer2 message0" was peak from index 1 successfully by "reader2"
message "writer0 message1" was wrote in index 3 successfully
message "writer4 message1" was wrote in index 0 successfully
message "writer3 message1" was wrote in index 1 successfully
Number of messages in the buffer: 4
Total length of messages: 64
Total memory usage: (15.98828125, 'MB')
```



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:

```
Writers: 4
Readers: 2
Stats: 1
Buffer size: 1024
load? (0/1): 0
Benchmark:
writer id = 0 times = 1
writer id = 1 times = 2
writer id = 2 times = 2
writer id = 3 times = 3
reader id = 0 times = 1
reader id = 1 times = 1
message "writer0 message0" was wrote in index 0 successfully
message "writer0 message0" was peak from index 0 successfully by "reader0"
There is no item in the buffer
message "writer1 message0" was wrote in index 1 successfully
message "writer2 message0" was wrote in index 2 successfully
message "writer3 message0" was wrote in index 3 successfully
Number of messages in the buffer: 3
Total length of messages: 48
Total memory usage: (15.9765625, 'MB')
message "writer2 message1" was wrote in index 4 successfully
message "writer1 message1" was wrote in index 5 successfully
message "writer3 message1" was wrote in index 6 successfully
message "writer3 message2" was wrote in index 7 successfully
load? (0/1): 1
messages saved successfully
```

فایل بعد از ذخیره سازی:

```
writer1 message0
writer2 message0
writer3 message0
writer2 message1
writer1 message1
writer3 message1
writer3 message2
```