امیرحسین بهمنی ۴۰۲۲۴۳۰۴۵

طراحی کلی سیستم

طراحی شده. این Producer-Consumer سیستم پیاده‌سازی شده یک معماری پیام‌رسان چندنخی است که بر اساس الگوی :سیستم شامل اجزای اصلی زیر می‌باشد

ساختار کلی اجزا

1. Message (پیام): واحد اساسی انتقال داده که شامل محتوا، شناسه یکتا، زمان ایجاد و TTL (Time-To-Live) اختیاری است.

2. MessageQueue (صف پیام): صف پیام با استفاده از deque پیاده‌سازی thread-safe که از bounded buffer پشتیبانی کرده و قابلیت انقضای پیام دارد.

3. Topic (موضوع): کانال ارتباطی مستقل که یک MessageQueue برای چندین fan-out داخلی دارد و از الگوی مصرف‌کننده پشتیبانی می‌کند.

4. Producer (تولیدکننده): نخ‌های مسئول تولید و انتشار پیام‌ها در موضوعات مختلف که با نرخ قابل تنظیم کار می‌کنند.

5. Consumer (مصرف‌کننده): نخ‌های مسئول دریافت و پردازش پیام‌ها از موضوعات با الگوی round-robin.

6. MessageBroker (واسط پیام): مدیر مرکزی که تمام اجزا را هماهنگ کرده و عملیات garbage collection برای پیام‌های منقضی شده انجام می‌دهد.

سناریوهای پشتیبانی شده

سیستم سه سناریوی مختلف را پشتیبانی می‌کند:

- **سناریو 1:** پایه برای پیام‌های عمومی Producer-Consumer
- **سناریو 2:** سیستم کارگر مبتنی بر اولویت با TaskProducer و WorkerConsumer
- **سناریو 3:** ترکیبی از پیام‌های دارای TTL و دائمی

استراتژی همگامسازی و پیشگیری از Race Conditions

1. قفل‌های چندسطحی (Multi-level Locking)

قفل‌های MessageQueue:

```
self._lock = threading.Lock()  # قفل اصلی
self._not_empty = threading.Condition(self._lock)  # شرط برای خالی نبودن
self._not_full = threading.Condition(self._lock)   # شرط برای پر نبودن
```

busy waiting استفاده می‌کند که امکان انتظار کارآمد نخ‌ها را بدون Condition Variables این طراحی از الگوی فراهم می‌آورد.

Topic: قفل‌های

```
self._consumer_lock = threading.Lock()  # مدیریت مصرف‌کنندگان
    self._stats_lock = threading.Lock()     # محافظت از آمار
```

Producer/Consumer: قفل‌های

```
self._stats_lock = threading.Lock()     # محافظت از آمارهای تولید/مصرف
```

2. Bounded Buffer Implementation

استفاده می‌کند: bounded buffer سیستم از الگوی کلاسیک

```
                # Producer blocking when queue is full
while (self._max_size and len(self._queue) >= self._max_size):
                    self._not_full.wait(timeout)

            # Consumer blocking when queue is empty
                    while len(self._queue) == 0:
                self._not_empty.wait(timeout)
```

3. جلوگیری از Deadlock

- استفاده از timeout در تمام عملیات blocking
- ترتیب ثابت در acquire کردن قفل‌ها
- استفاده از daemon threads برای تمیز کردن منابع

4. Thread-Safe Statistics

تمام آمارها با استفاده از قفل‌های جداگانه محافظت می‌شوند:

```
                with self._stats_lock:
self._total_produced += published_count
```

چالش‌ها و راه‌حل‌ها

1. چالش: مدیریت چندین موضوع توسط یک Consumer

مسئله: Consumer ممکن است در یک موضوع گیر کند و موضوعات دیگر را نادیده بگیرد.

راه‌حل: پیاده‌سازی الگوی round-robin با timeout کوتاه:

```
                for topic in self.topics:
    message = topic.consume(timeout=0.1)  # Short timeout
                    if message:
```

```
# Process and break to give other topics a chance
                break
```

2. چالش: Memory Leak در پیام‌های منقضی شده

مسئله: پیام‌های دارای TTL که منقضی شده‌اند در صف باقی می‌مانند.

راه‌حل: پیاده‌سازی Garbage Collector اختصاصی:

```
    def _start_garbage_collector(self):
                        def gc_worker():
                    while self._running:
          time.sleep(self._gc_interval)
                        total_cleaned = 0
        for topic in self._topics.values():
  cleaned = topic.cleanup_expired_messages()
                    total_cleaned += cleaned
```

3. چالش: Producer Blocking در صف‌های پر

مسئله: Producer ممکن است به طور نامحدود منتظر بماند.

راه‌حل: استفاده از timeout و مدیریت خطا:

```
if topic.publish(message, timeout=0.1):
                    published_count += 1
                                else:
            self._failed_productions += 1
```

4. چالش: Graceful Shutdown

مسئله: متوقف کردن سیستم بدون از دست دادن پیام‌ها.

راه‌حل: پیاده‌سازی signal handlers و shutdown تدریجی:

```
                    def shutdown(self):
                    # توقف تولیدکنندگان
        for producer in self._producers:
                        producer.stop()

                    # توقف مصرف‌کنندگان
        for consumer in self._consumers:
                        consumer.stop()

                    # بستن موضوعات
        for topic in self._topics.values():
                        topic.close()
```

5. چالش: Load Balancing بین Workers

مسئله: توزیع عادلانه کارها بین کارگرها.

راه‌حل: استفاده از الگوی First-Come-First-Served با round-robin topic access که به طور طبیعی load balancing ایجاد می‌کند.

نتیجه‌گیری

این سیستم یک پیاده‌سازی کامل و thread-safe از الگوهای استاندارد message queue concurrent programming است. استفاده از Condition Variables، timeout، و قفل‌های چندسطحی، deadlocks و race conditions سیستم را در برابر garbage، TTL مقاوم ساخته است. همچنین قابلیت‌هایی نظیر monitoring و collection آماری، آن را برای استفاده در محیط‌های production مناسب می‌سازد.