成绩：______________

计算机科学与技术学院

《操作系统》课程大作业

学院：计算机科学与技术学院

班级：04012202

任课教师：吴桂军

学号：2022211813

姓名：陈海彬

2024 年 12 月 15 日

操作系统大作业文档提纲

1. 设计目标

模拟网络请求处理流程：通过多线程协作，模拟客户端发送请求、服务器处理请求并分配给后端服务、后端服务处理请求并返回响应的整个网络请求处理流程，帮助理解实际网络应用的工作原理。

提高请求处理效率：利用多线程并行处理请求和任务，充分发挥多核 CPU 的计算能力，提高系统的请求吞吐量和响应速度，满足高并发场景下的性能需求。

保证线程安全和数据一致性：在多线程环境下，合理使用互斥锁和条件变量等同步机制，确保共享数据（如请求队列、任务队列和响应队列）的访问安全，避免出现数据竞争和不一致的问题，保障系统的稳定性和可靠性。

增强系统的可扩展性：通过模块化的设计，将生产者、请求处理和消费者等不同功能的线程分离，使得系统在面对不同规模和类型的请求时，可以灵活地调整线程数量和任务分配策略，方便后续的扩展和优化。

2. 功能描述

生产者功能：模拟客户端，负责生成网络请求。每个生产者线程在运行期间会持续生成包含请求 ID 和请求数据的请求，并将请求放入请求队列中，供请求处理线程处理.

请求处理功能：模拟服务器，负责从请求队列中取出请求，并根据请求类型将请求分配给相应的消费者线程（后端服务）。请求处理线程会管理多个任务队列，将请求放入相应的任务队列中，以便消费者线程进行处理.

消费者功能：模拟后端服务，负责从任务队列中取出请求，处理请求并生成响应数据。每个消费者线程会处理特定类型的任务队列中的请求，处理完成后将响应数据放入响应队列中，供后续处理或展示.

同步和协调功能：通过互斥锁和条件变量等同步机制，协调生产者、请求处理线程和消费者线程之间的协作。确保在请求队列、任务队列和响应队列满或空时，线程能够正确地等待和通知对方，避免出现死锁或资源浪费的情况.

3. **总体设计**

系统架构

生产者模块：包含多个生产者线程，负责生成请求。每个生产者线程独立运行，生成的请求通过请求队列传递给请求处理模块.

请求处理模块：包含一个请求处理线程，负责从请求队列中取出请求，并根据请求类型将请求分配给相应的消费者线程。请求处理线程管理多个任务队列，每个任务队列对应一个消费者线程.

消费者模块：包含多个消费者线程，负责处理任务队列中的请求。每个消费者线程独立运行，处理完成后将响应数据放入响应队列中.

同步机制模块：提供互斥锁和条件变量等同步机制，用于协调生产者、请求处理线程和消费者线程之间的协作，确保共享数据的访问安全和线程间的正确同步.

数据流

请求生成：生产者线程生成请求，包含请求 ID 和请求数据.

请求传递：生成的请求被放入请求队列中，等待请求处理线程处理.

请求处理和分配：请求处理线程从请求队列中取出请求，根据请求类型将请求分配给相应的消费者线程，请求被放入任务队列中.

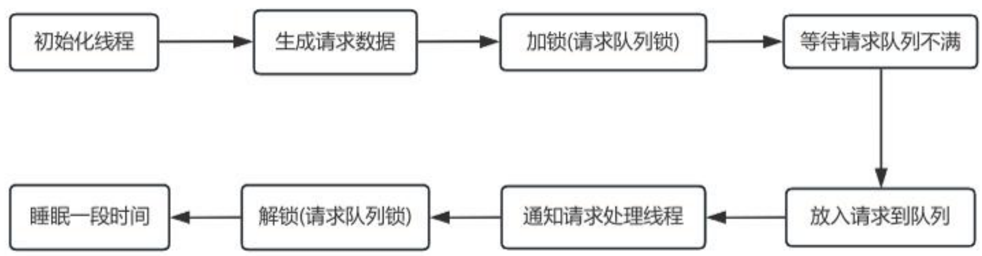
任务处理：消费者线程从任务队列中取出请求，处理请求并生成响应数据，响应数据被放入响应队列中.

响应处理：后续可以对响应队列中的数据进行进一步处理或展示.

4. 详细设计（流程图或算法步骤）

4.1 生产者线程设计

流程图：



算法步骤：

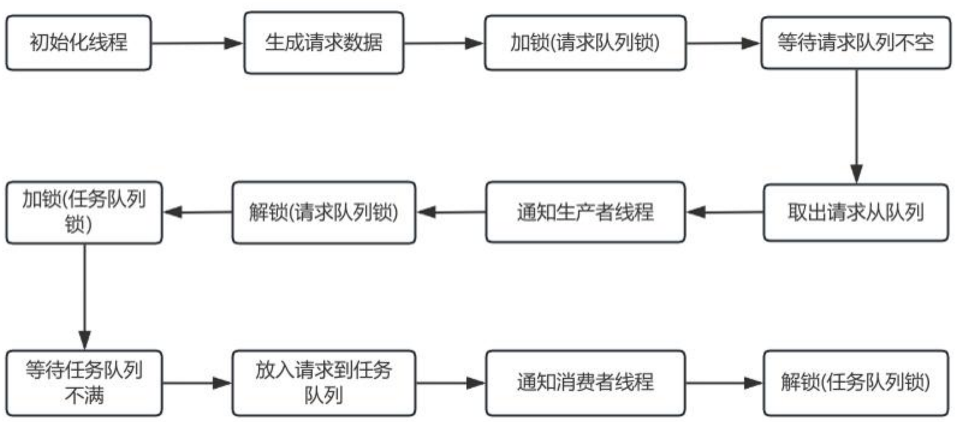
1. 初始化生产者线程，设置线程参数.
2. 生成请求数据，包括请求 ID 和请求内容.
3. 加锁请求队列的互斥锁，确保对请求队列的独占访问.
4. 等待请求队列不满，如果请求队列已满，则调用条件变量等待.
5. 将生成的请求放入请求队列中，更新队列状态.

6. 通知请求处理线程请求队列不为空，唤醒等待的请求处理线程.
7. 解锁请求队列的互斥锁，释放对请求队列的访问.
8. 睡眠一段时间，模拟请求生成的间隔.
9. 重复步骤 2-8，持续生成请求.

```c
// 生产者线程函数
void* producer(void* arg) {
    int producer_id = *((int*)arg);
    int request_id = 0;
    time_t start_time = time(NULL);

    while (true) {
        if (time(NULL) - start_time >= PRODUCER_LIFETIME) {
            printf("Producer %d stopped producing requests.\n", producer_id);
            break; // 生产者线程运行时间达到上限，停止生成请求
        }

        Request request;
        request.id = request_id++;
        snprintf(request.data, sizeof(request.data), "Request from producer %d", producer_id);

        // 加锁
        pthread_mutex_lock(&request_mutex);

        // 等待请求队列不满
        while (request_count == REQUEST_QUEUE_SIZE) {
            printf("Producer %d waiting for request queue not full.\n", producer_id);
            pthread_cond_wait(&request_not_full, &request_mutex);
        }

        // 将请求放入请求队列
        request_queue[request_in] = request;
        request_in = (request_in + 1) % REQUEST_QUEUE_SIZE;
        request_count++;

        printf("Producer %d produced request %d: %s\n", producer_id, request.id, request.data);

        // 通知请求处理线程请求队列不为空
        pthread_cond_signal(&request_not_empty);

        // 解锁
        pthread_mutex_unlock(&request_mutex);

        // 睡眠一定时间
        sleep(1);
    }

    // 标记生产者线程停止生成请求
    pthread_mutex_lock(&request_mutex);
    request_produced = true;
    pthread_mutex_unlock(&request_mutex);

    return NULL;
}
```

4.2 请求处理线程设计

流程图：



算法步骤：

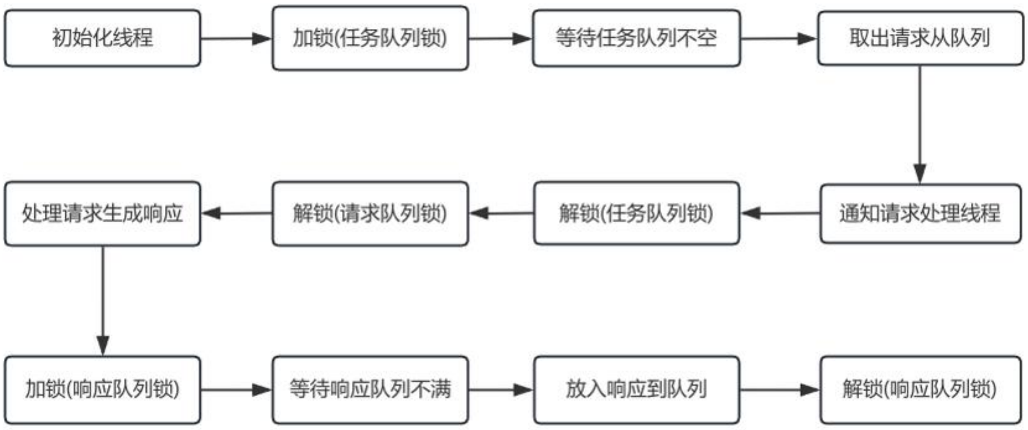
1. 初始化请求处理线程，设置线程参数.
2. 加锁请求队列的互斥锁，确保对请求队列的独占访问.
3. 等待请求队列不为空，如果请求队列为空，则调用条件变量等待.
4. 从请求队列中取出一个请求，更新队列状态.
5. 通知生产者线程请求队列不满，唤醒等待的生产者线程.
6. 解锁请求队列的互斥锁，释放对请求队列的访问.
7. 根据请求类型确定对应的消费者线程和任务队列.
8. 加锁任务队列的互斥锁，确保对任务队列的独占访问.
9. 等待任务队列不满，如果任务队列已满，则调用条件变量等待.
10. 将请求放入任务队列中，更新队列状态.
11. 通知消费者线程任务队列不为空，唤醒等待的消费者线程.
12. 解锁任务队列的互斥锁，释放对任务队列的访问.
13. 重复步骤 2-12，持续处理请求.

```c
// 请求处理线程函数
void* request_handler(void* arg) {
    while (true) {
        Request request;

        // 加锁
        pthread_mutex_lock(&request_mutex);

        // 等待请求队列不为空，或者所有生产者线程都停止生成请求且请求队列为空
        while (request_count == 0 && !request_produced) {
            printf("Request handler waiting for request queue not empty.\n");
            pthread_cond_wait(&request_not_empty, &request_mutex);
        }

        if (request_count == 0 && request_produced) {
            // 所有生产者线程都停止生成请求且请求队列为空，退出请求处理线程
            printf("Request handler stopped handling requests.\n");
            pthread_mutex_unlock(&request_mutex);
            break;
        }

        // 从请求队列中取出请求
        request = request_queue[request_out];
        request_out = (request_out + 1) % REQUEST_QUEUE_SIZE;
        request_count--;

        printf("Request handler handled request %d: %s\n", request.id, request.data);

        // 通知生产者线程请求队列不满
        pthread_cond_signal(&request_not_full);

        // 解锁
        pthread_mutex_unlock(&request_mutex);

        // 根据请求类型将请求分配给相应的消费者线程
        int consumer_id = request.id % NUM_CONSUMERS;

        // 加锁
        pthread_mutex_lock(&task_mutexes[consumer_id]);

        // 等待任务队列不满
        while (task_counts[consumer_id] == TASK_QUEUE_SIZE) {
            printf("Request handler waiting for task queue %d not full.\n", consumer_id);
            pthread_cond_wait(&task_not_full[consumer_id], &task_mutexes[consumer_id]);
        }
```

```c
        // 将请求放入任务队列
        task_queues[consumer_id][task_ins[consumer_id]] = request;
        task_ins[consumer_id] = (task_ins[consumer_id] + 1) % TASK_QUEUE_SIZE;
        task_counts[consumer_id]++;

        printf("Request handler assigned request %d to consumer %d.\n", request.id, consumer_id);

        // 通知消费者线程任务队列不为空
        pthread_cond_signal(&task_not_empty[consumer_id]);

        // 解锁
        pthread_mutex_unlock(&task_mutexes[consumer_id]);
    }
    return NULL;
}
```

4.3 消费者线程设计

流程图:



算法步骤:

1. 初始化消费者线程，设置线程参数.
2. 加锁任务队列的互斥锁，确保对任务队列的独占访问.
3. 等待任务队列不为空，如果任务队列为空，则调用条件变量等待.
4. 从任务队列中取出一个请求，更新队列状态.
5. 通知请求处理线程任务队列不满，唤醒等待的请求处理线程.
6. 解锁任务队列的互斥锁，释放对任务队列的访问.
7. 处理请求，生成响应数据.
8. 加锁响应队列的互斥锁，确保对响应队列的独占访问.
9. 等待响应队列不满，如果响应队列已满，则调用条件变量等待.
10. 将响应数据放入响应队列中，更新队列状态.
11. 解锁响应队列的互斥锁，释放对响应队列的访问.
12. 重复步骤 2-11，持续处理任务.

```c
// 消费者线程函数
void* consumer(void* arg) {
    int consumer_id = *((int*)arg);

    while (true) {
        Request request;

        // 加锁
        pthread_mutex_lock(&task_mutexes[consumer_id]);

        // 等待任务队列不为空
        while (task_counts[consumer_id] == 0) {
            printf("Consumer %d waiting for task queue not empty.\n", consumer_id);
            pthread_cond_wait(&task_not_empty[consumer_id], &task_mutexes[consumer_id]);
        }

        // 从任务队列中取出请求
        request = task_queues[consumer_id][task_outs[consumer_id]];
        task_outs[consumer_id] = (task_outs[consumer_id] + 1) % TASK_QUEUE_SIZE;
        task_counts[consumer_id]--;

        printf("Consumer %d consumed request %d: %s\n", consumer_id, request.id, request.data);

        // 通知请求处理线程任务队列不满
        pthread_cond_signal(&task_not_full[consumer_id]);

        // 解锁
        pthread_mutex_unlock(&task_mutexes[consumer_id]);

        // 处理请求并生成响应数据
        Response response;
        response.id = request.id;
        snprintf(response.data, sizeof(response.data),
                "Response for request %d from consumer %d", request.id, consumer_id);

        // 加锁
        pthread_mutex_lock(&response_mutex);

        // 等待响应队列不满
        while (response_count == RESPONSE_QUEUE_SIZE) {
            printf("Consumer %d waiting for response queue not full.\n", consumer_id);
            pthread_cond_wait(&response_not_full, &response_mutex);
        }

        // 将响应放入响应队列
        response_queue[response_in] = response;
        response_in = (response_in + 1) % RESPONSE_QUEUE_SIZE;
        response_count++;

        printf("Consumer %d produced response %d: %s\n",
                consumer_id, response.id, response.data);

        // 通知主线程响应队列不为空
        pthread_cond_signal(&response_not_empty);

        // 解锁
        pthread_mutex_unlock(&response_mutex);

        // 睡眠一定时间
        sleep(2);
    }
    return NULL;
}
```

5. 实现（给出编译截图、运行截图）

```
chenhaibin@chenhaibin-PC:~/计算机作业/操作系统/理论大作业$ vi network_request_system.c
chenhaibin@chenhaibin-PC:~/计算机作业/操作系统/理论大作业$ gcc -o network_request_system network_request_system.c -lpthread
chenhaibin@chenhaibin-PC:~/计算机作业/操作系统/理论大作业$ ./network_request_system
```

```
Producer 0 produced request 0: Request from producer 0
Producer 4 produced request 0: Request from producer 4
Producer 1 produced request 0: Request from producer 1
Request handler handled request 0: Request from producer 0
```

```
Request handler handled request 0: Request from producer 0
Request handler assigned request 0 to consumer 0.
Producer 2 produced request 0: Request from producer 2
Consumer 0 consumed request 0: Request from producer 0
Consumer 0 produced response 0: Response for request 0 from consumer 0
Consumer 1 waiting for task queue not empty.
Producer 3 produced request 0: Request from producer 3
Consumer 2 waiting for task queue not empty.
Request handler handled request 0: Request from producer 4
Request handler assigned request 0 to consumer 0.
Request handler handled request 0: Request from producer 1
Request handler assigned request 0 to consumer 0.
Request handler handled request 0: Request from producer 2
Request handler assigned request 0 to consumer 0.
Request handler handled request 0: Request from producer 3
Request handler assigned request 0 to consumer 0.
Request handler waiting for request queue not empty.
Producer 0 produced request 1: Request from producer 0
Request handler handled request 1: Request from producer 0
Request handler assigned request 1 to consumer 1.
Request handler waiting for request queue not empty.
Producer 4 produced request 1: Request from producer 4
Consumer 1 consumed request 1: Request from producer 0
Consumer 1 produced response 1: Response for request 1 from consumer 1
Producer 1 produced request 1: Request from producer 1
Request handler handled request 1: Request from producer 4
Request handler assigned request 0 to consumer 0.
Request handler handled request 0: Request from producer 1
Request handler assigned request 0 to consumer 0.
Request handler handled request 0: Request from producer 2
Request handler assigned request 0 to consumer 0.
Request handler handled request 0: Request from producer 3
Request handler assigned request 0 to consumer 0.
Request handler waiting for request queue not empty.
Producer 0 produced request 1: Request from producer 0
Request handler handled request 1: Request from producer 0
Request handler assigned request 1 to consumer 1.
Request handler waiting for request queue not empty.
Producer 4 produced request 1: Request from producer 4
Consumer 1 consumed request 1: Request from producer 0
Consumer 1 produced response 1: Response for request 1 from consumer 1
Producer 1 produced request 1: Request from producer 1
Request handler handled request 1: Request from producer 4
Request handler assigned request 1 to consumer 1.
Request handler handled request 1: Request from producer 1
Request handler assigned request 1 to consumer 1.
Request handler waiting for request queue not empty.
Producer 2 produced request 1: Request from producer 2
Producer 3 produced request 1: Request from producer 3
Request handler handled request 1: Request from producer 2
Request handler assigned request 1 to consumer 1.
Request handler handled request 1: Request from producer 3
Request handler assigned request 1 to consumer 1.
```

```
Request handler waiting for request queue not empty.
Consumer 0 consumed request 0: Request from producer 4
Consumer 0 produced response 0: Response for request 0 from consumer 0
Producer 0 produced request 2: Request from producer 0
Request handler handled request 2: Request from producer 0
Request handler assigned request 2 to consumer 2.
Producer 4 produced request 2: Request from producer 4
Consumer 2 consumed request 2: Request from producer 0
Consumer 2 produced response 2: Response for request 2 from consumer 2
Request handler handled request 2: Request from producer 4
Request handler assigned request 2 to consumer 2.
Request handler waiting for request queue not empty.
Producer 1 produced request 2: Request from producer 1
Producer 2 produced request 2: Request from producer 2
Producer 3 produced request 2: Request from producer 3
Request handler handled request 2: Request from producer 1
Request handler assigned request 2 to consumer 2.
Request handler handled request 2: Request from producer 2
Request handler assigned request 2 to consumer 2.
Request handler handled request 2: Request from producer 3
Request handler assigned request 2 to consumer 2.
Request handler waiting for request queue not empty.
Producer 0 produced request 3: Request from producer 0
Request handler handled request 3: Request from producer 0
Request handler assigned request 3 to consumer 0.
Request handler waiting for request queue not empty.
Consumer 1 consumed request 1: Request from producer 4
Consumer 1 produced response 1: Response for request 1 from consumer 1
Producer 4 produced request 3: Request from producer 4
Producer 1 produced request 3: Request from producer 1
Producer 2 produced request 3: Request from producer 2
Producer 3 produced request 3: Request from producer 3
Request handler handled request 3: Request from producer 4
Request handler assigned request 3 to consumer 0.
Request handler handled request 3: Request from producer 1
Request handler assigned request 3 to consumer 0.
Request handler handled request 3: Request from producer 2
Request handler assigned request 3 to consumer 0.
Request handler handled request 3: Request from producer 3
Request handler assigned request 3 to consumer 0.
Request handler waiting for request queue not empty.
Consumer 0 consumed request 0: Request from producer 1
Consumer 0 produced response 0: Response for request 0 from consumer 0
Producer 0 produced request 4: Request from producer 0
Consumer 2 consumed request 2: Request from producer 4
Consumer 2 produced response 2: Response for request 2 from consumer 2
Request handler handled request 4: Request from producer 0
Request handler assigned request 4 to consumer 1.
Request handler waiting for request queue not empty.
Producer 4 produced request 4: Request from producer 4
Producer 1 produced request 4: Request from producer 1
Request handler handled request 4: Request from producer 4
Request handler assigned request 4 to consumer 1.
```

```
Producer 2 produced request 4: Request from producer 2
Request handler handled request 4: Request from producer 1
Request handler assigned request 4 to consumer 1.
Request handler handled request 4: Request from producer 2
Request handler assigned request 4 to consumer 1.
Request handler waiting for request queue not empty.
Producer 3 produced request 4: Request from producer 3
Request handler handled request 4: Request from producer 3
Request handler assigned request 4 to consumer 1.
Request handler waiting for request queue not empty.
Producer 0 produced request 5: Request from producer 0
Consumer 1 consumed request 1: Request from producer 1
Consumer 1 produced response 1: Response for request 1 from consumer 1
Request handler handled request 5: Request from producer 0
Request handler assigned request 5 to consumer 2.
Request handler waiting for request queue not empty.
Producer 4 produced request 5: Request from producer 4
Producer 1 produced request 5: Request from producer 1
Producer 3 produced request 5: Request from producer 3
Producer 2 produced request 5: Request from producer 2
Request handler handled request 5: Request from producer 4
Request handler assigned request 5 to consumer 2.
Request handler handled request 5: Request from producer 1
Request handler assigned request 5 to consumer 2.
Request handler handled request 5: Request from producer 3
Request handler assigned request 5 to consumer 2.
Request handler handled request 5: Request from producer 2
Request handler assigned request 5 to consumer 2.
Request handler waiting for request queue not empty.
Consumer 0 consumed request 0: Request from producer 2
Consumer 0 produced response 0: Response for request 0 from consumer 0
Consumer 2 consumed request 2: Request from producer 1
Consumer 2 produced response 2: Response for request 2 from consumer 2
Producer 0 produced request 6: Request from producer 0
Request handler handled request 6: Request from producer 0
Request handler assigned request 6 to consumer 0.
Request handler waiting for request queue not empty.
Producer 4 produced request 5: Request from producer 4
Producer 1 produced request 5: Request from producer 1
Producer 3 produced request 5: Request from producer 3
Producer 2 produced request 5: Request from producer 2
Request handler handled request 5: Request from producer 4
Request handler assigned request 5 to consumer 2.
Request handler handled request 5: Request from producer 1
Request handler assigned request 5 to consumer 2.
Request handler handled request 5: Request from producer 3
Request handler assigned request 5 to consumer 2.
Request handler handled request 5: Request from producer 2
Request handler assigned request 5 to consumer 2.
Request handler waiting for request queue not empty.
Consumer 0 consumed request 0: Request from producer 2
Consumer 0 produced response 0: Response for request 0 from consumer 0
Consumer 2 consumed request 2: Request from producer 1
```

```
Consumer 2 produced response 2: Response for request 2 from consumer 2
Producer 0 produced request 6: Request from producer 0
Request handler handled request 6: Request from producer 0
Request handler assigned request 6 to consumer 0.
Request handler waiting for request queue not empty.
Producer 4 produced request 6: Request from producer 4
Producer 3 produced request 6: Request from producer 3
Request handler handled request 6: Request from producer 4
Request handler assigned request 6 to consumer 0.
Request handler handled request 6: Request from producer 3
Request handler assigned request 6 to consumer 0.
Producer 1 produced request 6: Request from producer 1
Request handler handled request 6: Request from producer 1
Request handler assigned request 6 to consumer 0.
Request handler waiting for request queue not empty.
Producer 2 produced request 6: Request from producer 2
Request handler handled request 6: Request from producer 2
Request handler assigned request 6 to consumer 0.
Request handler waiting for request queue not empty.
Consumer 1 consumed request 1: Request from producer 2
Consumer 1 produced response 1: Response for request 1 from consumer 1
Producer 0 produced request 7: Request from producer 0
Request handler handled request 7: Request from producer 0
Request handler assigned request 7 to consumer 1.
Request handler waiting for request queue not empty.
Producer 4 produced request 7: Request from producer 4
Request handler handled request 7: Request from producer 4
Request handler assigned request 7 to consumer 1.
Request handler waiting for request queue not empty.
Producer 3 produced request 7: Request from producer 3
Request handler handled request 7: Request from producer 3
Request handler assigned request 7 to consumer 1.
Request handler waiting for request queue not empty.
Producer 1 produced request 7: Request from producer 1
Request handler handled request 7: Request from producer 1
Request handler assigned request 7 to consumer 1.
Request handler waiting for request queue not empty.
Producer 2 produced request 7: Request from producer 2
Request handler handled request 7: Request from producer 2
Request handler assigned request 7 to consumer 1.
Request handler waiting for request queue not empty.
Consumer 0 consumed request 0: Request from producer 3
Consumer 0 produced response 0: Response for request 0 from consumer 0
Consumer 2 consumed request 2: Request from producer 2
Consumer 2 produced response 2: Response for request 2 from consumer 2
Producer 0 produced request 8: Request from producer 0
Request handler handled request 8: Request from producer 0
Request handler assigned request 8 to consumer 2.
Request handler waiting for request queue not empty.
Producer 4 produced request 8: Request from producer 4
Request handler handled request 8: Request from producer 4
Request handler assigned request 8 to consumer 2.
Request handler waiting for request queue not empty.
```

```
Request handler waiting for request queue not empty.
Producer 1 produced request 8: Request from producer 1
Request handler handled request 8: Request from producer 1
Request handler assigned request 8 to consumer 2.
Request handler waiting for request queue not empty.
Consumer 1 consumed request 1: Request from producer 3
Consumer 1 produced response 1: Response for request 1 from consumer 1
Producer 0 produced request 9: Request from producer 0
Producer 2 produced request 9: Request from producer 2
Request handler handled request 9: Request from producer 3
Request handler assigned request 9 to consumer 0.
Request handler handled request 9: Request from producer 2
Request handler assigned request 9 to consumer 0.
Request handler waiting for request queue not empty.
Producer 1 produced request 9: Request from producer 1
Request handler handled request 9: Request from producer 1
Request handler assigned request 9 to consumer 0.
Request handler waiting for request queue not empty.
Consumer 2 consumed request 2: Request from producer 3
Consumer 2 produced response 2: Response for request 2 from consumer 2
Consumer 0 consumed request 3: Request from producer 0
Consumer 0 produced response 3: Response for request 3 from consumer 0
Producer 0 stopped producing requests.
Producer 4 stopped producing requests.
Producer 3 stopped producing requests.
Producer 2 stopped producing requests.
Producer 1 stopped producing requests.
Consumer 1 consumed request 4: Request from producer 0
Consumer 1 produced response 4: Response for request 4 from consumer 1
Consumer 2 consumed request 5: Request from producer 0
Consumer 2 produced response 5: Response for request 5 from consumer 2
Consumer 0 consumed request 3: Request from producer 4
Consumer 0 produced response 3: Response for request 3 from consumer 0
Consumer 1 consumed request 4: Request from producer 4
Consumer 1 produced response 4: Response for request 4 from consumer 1
Consumer 2 consumed request 5: Request from producer 4
Consumer 2 produced response 5: Response for request 5 from consumer 2
Consumer 0 consumed request 3: Request from producer 1
Consumer 0 produced response 3: Response for request 3 from consumer 0
Consumer 1 consumed request 4: Request from producer 1
Consumer 1 produced response 4: Response for request 4 from consumer 1
Consumer 2 consumed request 5: Request from producer 1
Consumer 2 produced response 5: Response for request 5 from consumer 2
Consumer 0 consumed request 3: Request from producer 2
Consumer 0 produced response 3: Response for request 3 from consumer 0
Consumer 1 consumed request 4: Request from producer 2
Consumer 1 produced response 4: Response for request 4 from consumer 1
Consumer 2 consumed request 5: Request from producer 3
Consumer 2 produced response 5: Response for request 5 from consumer 2
Consumer 0 consumed request 3: Request from producer 3
Consumer 0 produced response 3: Response for request 3 from consumer 0
Consumer 1 consumed request 4: Request from producer 3
```

```
Consumer 1 produced response 4: Response for request 4 from consumer 1
Consumer 2 consumed request 5: Request from producer 2
Consumer 2 produced response 5: Response for request 5 from consumer 2
Consumer 0 consumed request 6: Request from producer 0
Consumer 0 produced response 6: Response for request 6 from consumer 0
Consumer 1 consumed request 7: Request from producer 0
Consumer 1 produced response 7: Response for request 7 from consumer 1
Consumer 2 consumed request 8: Request from producer 0
Consumer 2 produced response 8: Response for request 8 from consumer 2
Consumer 0 consumed request 6: Request from producer 4
Consumer 0 produced response 6: Response for request 6 from consumer 0
Consumer 1 consumed request 7: Request from producer 4
Consumer 1 produced response 7: Response for request 7 from consumer 1
Consumer 2 consumed request 8: Request from producer 4
Consumer 2 produced response 8: Response for request 8 from consumer 2
Consumer 0 consumed request 6: Request from producer 3
Consumer 0 produced response 6: Response for request 6 from consumer 0
Consumer 1 consumed request 7: Request from producer 3
Consumer 1 produced response 7: Response for request 7 from consumer 1
Consumer 2 consumed request 8: Request from producer 3
Consumer 2 produced response 8: Response for request 8 from consumer 2
Consumer 0 consumed request 6: Request from producer 1
Consumer 0 produced response 6: Response for request 6 from consumer 0
Consumer 1 consumed request 7: Request from producer 1
Consumer 1 produced response 7: Response for request 7 from consumer 1
Consumer 2 consumed request 8: Request from producer 2
Consumer 2 produced response 8: Response for request 8 from consumer 2
Consumer 0 consumed request 6: Request from producer 2
Consumer 0 produced response 6: Response for request 6 from consumer 0
Consumer 1 consumed request 7: Request from producer 2
Consumer 1 produced response 7: Response for request 7 from consumer 1
Consumer 2 consumed request 8: Request from producer 1
Consumer 2 produced response 8: Response for request 8 from consumer 2
Consumer 0 consumed request 9: Request from producer 0
Consumer 0 produced response 9: Response for request 9 from consumer 0
Consumer 1 waiting for task queue not empty.
Consumer 2 waiting for task queue not empty.
Consumer 0 consumed request 9: Request from producer 4
Consumer 0 produced response 9: Response for request 9 from consumer 0
Consumer 0 consumed request 9: Request from producer 3
Consumer 0 produced response 9: Response for request 9 from consumer 0
Consumer 0 consumed request 9: Request from producer 2
Consumer 0 produced response 9: Response for request 9 from consumer 0
Consumer 0 consumed request 9: Request from producer 1
Consumer 0 produced response 9: Response for request 9 from consumer 0
Consumer 0 waiting for task queue not empty.
```

6. 参考资料（3-5 个，文献或链接）

《Linux 多线程编程》：介绍了 Linux 环境下多线程编程的基本概念、互斥锁、条件变量等同步机制的使用方法，为本系统的设计提供了理论基础和编程指导.

《操作系统原理》：详细阐述了操作系统中进程和线程的概念、调度算法、同步和通信机制等，帮助理解多线程协作的原理和实现方式.
《C 语言程序设计》：提供了 C 语言的基本语法和编程技巧，是编写多线程程序的基础.

附件：源代码

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>

#define REQUEST_QUEUE_SIZE 100
#define TASK_QUEUE_SIZE 50
#define RESPONSE_QUEUE_SIZE 100
#define NUM_PRODUCERS 5
#define NUM_CONSUMERS 3
#define PRODUCER_LIFETIME 10 // 生产者线程运行时间（秒）

// 请求结构体
typedef struct {
    int id;
    char data[100];
} Request;

// 响应结构体
typedef struct {
    int id;
    char data[100];
} Response;

// 请求队列
Request request_queue[REQUEST_QUEUE_SIZE];
int request_count = 0;
int request_in = 0;
```

```c
int request_out = 0;
bool request_produced = false; // 标记是否所有生产者线程都停止生成请求

// 任务队列
Request task_queues[NUM_CONSUMERS][TASK_QUEUE_SIZE];
int task_counts[NUM_CONSUMERS] = {0};
int task_ins[NUM_CONSUMERS] = {0};
int task_outs[NUM_CONSUMERS] = {0};

// 响应队列
Response response_queue[RESPONSE_QUEUE_SIZE];
int response_count = 0;
int response_in = 0;
int response_out = 0;

// 互斥锁和条件变量
pthread_mutex_t request_mutex;
pthread_cond_t request_not_empty;
pthread_cond_t request_not_full;

pthread_mutex_t task_mutexes[NUM_CONSUMERS];
pthread_cond_t task_not_empty[NUM_CONSUMERS];
pthread_cond_t task_not_full[NUM_CONSUMERS];

pthread_mutex_t response_mutex;
pthread_cond_t response_not_empty;
pthread_cond_t response_not_full;

// 生产者线程函数
void* producer(void* arg) {
    int producer_id = *((int*)arg);
    int request_id = 0;
    time_t start_time = time(NULL);
```

```
    while (true) {
        if (time(NULL) - start_time >= PRODUCER_LIFETIME) {
            printf("Producer %d stopped producing requests.\n", producer_id);
            break; // 生产者线程运行时间达到上限，停止生成请求
        }

        Request request;
        request.id = request_id++;
        snprintf(request.data, sizeof(request.data), "Request from producer %d", producer_id);

        // 加锁
        pthread_mutex_lock(&request_mutex);

        // 等待请求队列不满
        while (request_count == REQUEST_QUEUE_SIZE) {
            printf("Producer %d waiting for request queue not full.\n", producer_id);
            pthread_cond_wait(&request_not_full, &request_mutex);
        }

        // 将请求放入请求队列
        request_queue[request_in] = request;
        request_in = (request_in + 1) % REQUEST_QUEUE_SIZE;
        request_count++;

        printf("Producer %d produced request %d: %s\n", producer_id, request.id, request.data);

        // 通知请求处理线程请求队列不为空
        pthread_cond_signal(&request_not_empty);

        // 解锁
        pthread_mutex_unlock(&request_mutex);
```

```
        // 睡眠一定时间
        sleep(1);
    }

    // 标记生产者线程停止生成请求
    pthread_mutex_lock(&request_mutex);
    request_produced = true;
    pthread_mutex_unlock(&request_mutex);

    return NULL;
}

// 请求处理线程函数
void* request_handler(void* arg) {
    while (true) {
        Request request;

        // 加锁
        pthread_mutex_lock(&request_mutex);

        // 等待请求队列不为空，或者所有生产者线程都停止生成请求且请求队列为空
        while (request_count == 0 && !request_produced) {
            printf("Request handler waiting for request queue not empty.\n");
            pthread_cond_wait(&request_not_empty, &request_mutex);
        }

        if (request_count == 0 && request_produced) {
            // 所有生产者线程都停止生成请求且请求队列为空，退出请求处理线程
            printf("Request handler stopped handling requests.\n");
            pthread_mutex_unlock(&request_mutex);
            break;
        }
```

```
        // 从请求队列中取出请求
        request = request_queue[request_out];
        request_out = (request_out + 1) % REQUEST_QUEUE_SIZE;
        request_count--;

        printf("Request handler handled request %d: %s\n", request.id,
request.data);

        // 通知生产者线程请求队列不满
        pthread_cond_signal(&request_not_full);

        // 解锁
        pthread_mutex_unlock(&request_mutex);

        // 根据请求类型将请求分配给相应的消费者线程
        int consumer_id = request.id % NUM_CONSUMERS;

        // 加锁
        pthread_mutex_lock(&task_mutexes[consumer_id]);

        // 等待任务队列不满
        while (task_counts[consumer_id] == TASK_QUEUE_SIZE) {
            printf("Request handler waiting for task queue %d not full.\n",
consumer_id);
            pthread_cond_wait(&task_not_full[consumer_id],
&task_mutexes[consumer_id]);
        }

        // 将请求放入任务队列
        task_queues[consumer_id][task_ins[consumer_id]] = request;
        task_ins[consumer_id] = (task_ins[consumer_id] + 1) %
TASK_QUEUE_SIZE;
        task_counts[consumer_id]++;
```

```
        printf("Request handler assigned request %d to consumer %d.\n",
request.id, consumer_id);

        // 通知消费者线程任务队列不为空
        pthread_cond_signal(&task_not_empty[consumer_id]);

        // 解锁
        pthread_mutex_unlock(&task_mutexes[consumer_id]);
    }
    return NULL;
}

// 消费者线程函数
void* consumer(void* arg) {
    int consumer_id = *((int*)arg);

    while (true) {
        Request request;

        // 加锁
        pthread_mutex_lock(&task_mutexes[consumer_id]);

        // 等待任务队列不为空
        while (task_counts[consumer_id] == 0) {
            printf("Consumer %d waiting for task queue not empty.\n",
consumer_id);
            pthread_cond_wait(&task_not_empty[consumer_id],
&task_mutexes[consumer_id]);
        }

        // 从任务队列中取出请求
        request = task_queues[consumer_id][task_outs[consumer_id]];
        task_outs[consumer_id] = (task_outs[consumer_id] + 1) %
TASK_QUEUE_SIZE;
```

```
        task_counts[consumer_id]--;

        printf("Consumer %d consumed request %d: %s\n", consumer_id, request.id, request.data);

        // 通知请求处理线程任务队列不满
        pthread_cond_signal(&task_not_full[consumer_id]);

        // 解锁
        pthread_mutex_unlock(&task_mutexes[consumer_id]);

        // 处理请求并生成响应数据
        Response response;
        response.id = request.id;
        snprintf(response.data, sizeof(response.data), "Response for request %d from consumer %d", request.id, consumer_id);

        // 加锁
        pthread_mutex_lock(&response_mutex);

        // 等待响应队列不满
        while (response_count == RESPONSE_QUEUE_SIZE) {
            printf("Consumer %d waiting for response queue not full.\n", consumer_id);
            pthread_cond_wait(&response_not_full, &response_mutex);
        }

        // 将响应放入响应队列
        response_queue[response_in] = response;
        response_in = (response_in + 1) % RESPONSE_QUEUE_SIZE;
        response_count++;

        printf("Consumer %d produced response %d: %s\n", consumer_id, response.id, response.data);
```

```c
        // 通知主线程响应队列不为空
        pthread_cond_signal(&response_not_empty);

        // 解锁
        pthread_mutex_unlock(&response_mutex);

        // 睡眠一定时间
        sleep(2);
    }
    return NULL;
}

// 主函数
int main() {
    pthread_t producer_threads[NUM_PRODUCERS];
    pthread_t request_handler_thread;
    pthread_t consumer_threads[NUM_CONSUMERS];
    int producer_ids[NUM_PRODUCERS];
    int consumer_ids[NUM_CONSUMERS];

    // 初始化互斥锁和条件变量
    pthread_mutex_init(&request_mutex, NULL);
    pthread_cond_init(&request_not_empty, NULL);
    pthread_cond_init(&request_not_full, NULL);

    for (int i = 0; i < NUM_CONSUMERS; i++) {
        pthread_mutex_init(&task_mutexes[i], NULL);
        pthread_cond_init(&task_not_empty[i], NULL);
        pthread_cond_init(&task_not_full[i], NULL);
    }

    pthread_mutex_init(&response_mutex, NULL);
    pthread_cond_init(&response_not_empty, NULL);
```

```c
    pthread_cond_init(&response_not_full, NULL);

    // 创建生产者线程
    for (int i = 0; i < NUM_PRODUCERS; i++) {
        producer_ids[i] = i;
        pthread_create(&producer_threads[i], NULL, producer, &producer_ids[i]);
    }
    // 创建请求处理线程
    pthread_create(&request_handler_thread, NULL, request_handler, NULL);

    // 创建消费者线程
    for (int i = 0; i < NUM_CONSUMERS; i++) {
        consumer_ids[i] = i;
        pthread_create(&consumer_threads[i], NULL, consumer, &consumer_ids[i]);
    }

    // 等待生产者线程结束
    for (int i = 0; i < NUM_PRODUCERS; i++) {
        pthread_join(producer_threads[i], NULL);
    }

    // 等待请求处理线程结束
    pthread_join(request_handler_thread, NULL);

    // 等待消费者线程结束
    for (int i = 0; i < NUM_CONSUMERS; i++) {
        pthread_join(consumer_threads[i], NULL);
    }

    // 销毁互斥锁和条件变量
    pthread_mutex_destroy(&request_mutex);
    pthread_cond_destroy(&request_not_empty);
```

```
    pthread_cond_destroy(&request_not_full);

    for (int i = 0; i < NUM_CONSUMERS; i++) {
        pthread_mutex_destroy(&task_mutexes[i]);
        pthread_cond_destroy(&task_not_empty[i]);
        pthread_cond_destroy(&task_not_full[i]);
    }

    pthread_mutex_destroy(&response_mutex);
    pthread_cond_destroy(&response_not_empty);
    pthread_cond_destroy(&response_not_full);

    return 0;
}
```