TCT-기술인증테스트
시스템&솔루션개발
실기형 문제지

[2022년 #차]

사번		성명	
유의 사항	1. 공정한 평가를 위해 동료를 도와주는 행위, 보여주는 행위를 금지하고 있습니다. 2. 부정행위 적발 시, 응시한 평가는 0점 처리됩니다. 3. 본 시험지는 응시장 외부로 유출할 수 없으며, 시험 종료 후 감독관에게 제출해야 합니다.		

LG CNS

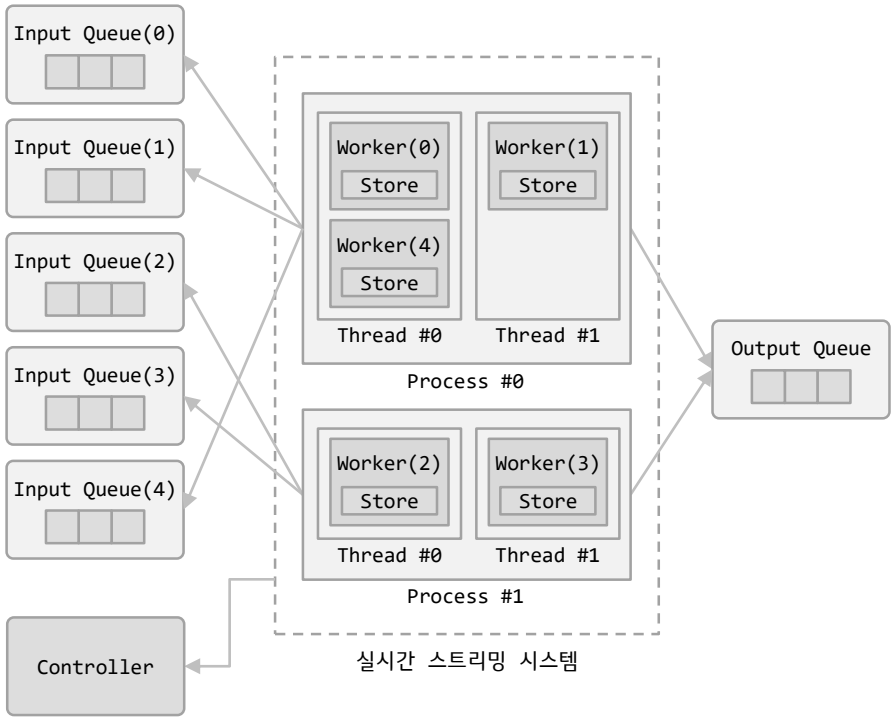
실시간 스트리밍 시스템

개요

해당 시스템 구현은 동해 요구사항 분석, 데이터 관리, HTTP Client 구현, Process/Thread 관리 등의 기술역량 및 프로그램 구현 역량을 측정하기 위한 문제입니다.

설명

본 프로그램은 Input Queue에서 데이터를 가져와 다수의 Process/Thread를 활용하여 Worker를 실행하고 그 결과를 Output Queue로 출력하는 '실시간 스트리밍 시스템'입니다.



[기능 요약]

- '실시간 스트리밍 시스템'은 콘솔 또는 Input Queue로부터 데이터를 가져와 Worker를 실행하고 그 결과를 콘솔 또는 Output Queue로 출력하는 기능을 반복한다.
- '실시간 스트리밍 시스템'은 Controller가 제공하는 정보에 따라 Process/Thread/Worker를 생성하고 Worker를 Process/Thread에 할당하여 실행한다.

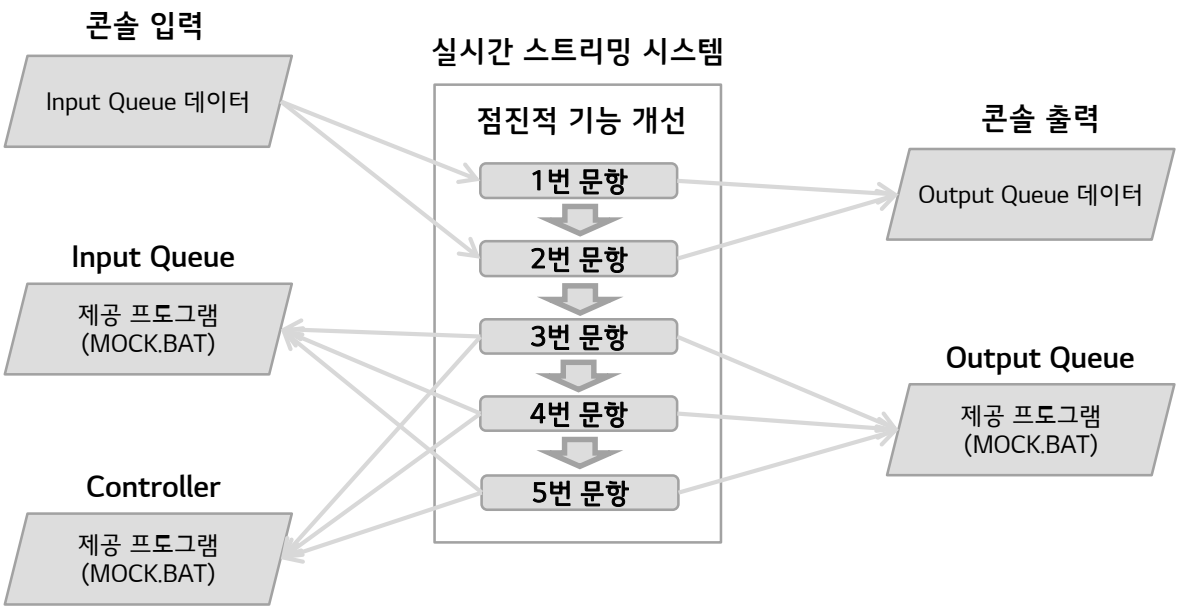
- Input Queue : 입력 데이터를 제공하는 HTTP 서버 (1, 2번 소문항에서는 콘솔)
- Worker : Input Queue로부터 입력된 데이터를 처리하여 Output Queue로 출력할 데이터를 생성하는 구현체 (라이브러리 및 소스코드도 제공됨)
- Output Queue : Worker 실행 결과를 출력하는 HTTP 서버 (1, 2번 소문항에서는 콘솔)
- Store : Worker가 입력 데이터를 저장하기 위해 사용하는 문자열 List (Worker마다 존재)
- Controller : Input/Output Queue 정보, Process/Thread 정보 등을 제공하는 HTTP 서버

주의사항

실행 결과로 평가하고 부분점수는 없으므로 아래사항을 필히 주의해야 함

- 구현된 프로그램은 실행 완결성 필수 (명확한 실행&종료 처리, 정확한 결과 출력, 동상의 실행 시간)
- 소 문항별 결과 검수 필수 (선행문항 오류 시, 후속문항 전체에 오류가 발생할 수 있음)
- 제시된 조건이 없는 한 선행요구사항 유지 필수 (소 문항별 입출력 관계도 참고)
- 프로그램 실행 위치 및 실행결과출력 (위치, 파일명, 데이터포맷)은 요구사항과 정확히 일치 필수
- 제시된 모든 위치는 상대경로 사용 필수 (프로그램 실행 위치 기준)
- 종료조건에 맞는 자동종료 처리 필수 (불필요한 종료방해처리(pause/입력대기 등)를 하면 안됨)
- 모든 문자는 대소문자 구분 필수

문항 관계



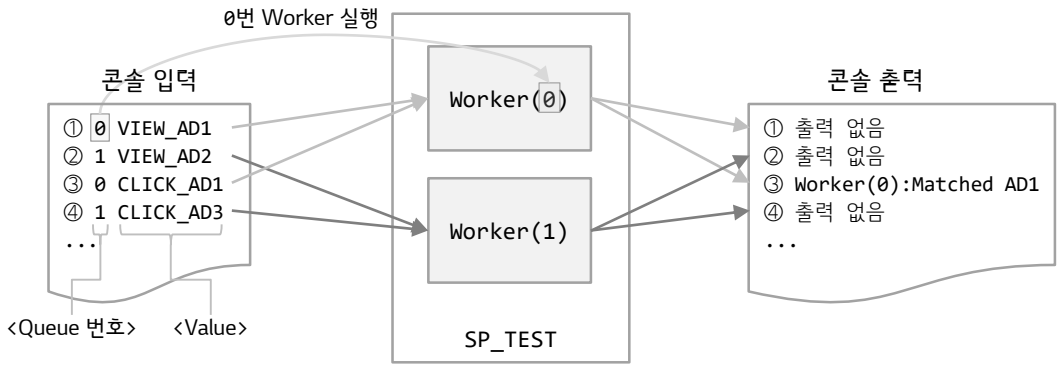
문제

아래 제시된 문항은 문항번호가 증가할 수록 점진적 개선을 요구하는 방식으로 구성되어 있으며, 제시된 문항번호 별로 각각 **구현된 소스와 컴파일 된 실행파일을 제출**하시오.
cf) 1번 구현 → 1번 소스복사 → 2번 구현 → 2번 소스복사 → ...

1. Input Queue 데이터가 콘솔을 통해 입력되면 Worker(라이브러리 제공)를 호출하고 결과를 Output Queue(콘솔)로 출력하는 동작을 반복하는 '실시간 스트리밍 시스템'을 구현하시오. (20점)

   1) Worker 생성 - Worker 인스턴스 2개 생성
   2) Input Queue 입력 - 콘솔을 통해 Input Queue 데이터를 입력 받아 Parsing 수행
   3) Worker 실행 - Parsing된 Input Queue 데이터로 Worker 실행
   4) Output Queue 출력 - Worker 실행 결과(리턴된 문자열)가 Null이 아니면 콘솔(Output Queue)로 출력

상세설명



※ Worker(라이브러리로 제공되며 이미 프로젝트에 포함되어 있음) 생성
- 구현 프로그램이 시작되면 <Queue 번호(0, 1)>를 파라미터로 하여 2개의 Worker 인스턴스 생성 (언어별 상세 사항은 소문항 홈 아래 README.TXT 참조)

```
(Java) public Worker(int queueNo)
(C#) public Worker(int queueNo)
(C) Worker* New_Worker(int queue_no)
```

※ Input Queue 입력 (콘솔 입력)
- Input Queue 데이터는 아래의 형식으로 콘솔을 통해 입력됨
- 입력 형식 : <Queue 번호> + " " + <Value>
  예> 0 VIEW_AD1 (<Queue 번호>: 0, <Value>: VIEW_AD1)

※ Worker(라이브러리로 제공되며 이미 프로젝트에 포함되어 있음) 실행
- 콘솔을 통해 Input Queue 데이터가 입력되면 <Value>를 파라미터로 하여 <Queue 번호>에 해당하는 Worker의 아래 메소드/함수 호출 (언어별 상세 사항은 소문항 홈 아래 README.TXT 참조)

```
(Java) public String run(String value)
(C#) public string Run(string value)
(C) char* (*run)(Worker* this, char* value)
```

※ Output Queue 출력 (콘솔 출력)
- Worker 실행 결과(리턴된 문자열)가 Null이 아니면 콘솔(Output Queue)로 출력

※ 콘솔 입/출력

- 입력 포맷 : Input Queue 데이터 (형식: <Queue 번호> + " " + <Value>)
- 출력 포맷 : Worker 실행 결과 (형식: <Worker 실행 결과>)

```
C:\>SP_TEST<엔터키>                    ← 구현한 프로그램 실행 (Argument 없음)
0 VIEW_AD1                            ← 콘솔 입력
1 VIEW_AD2                            ← 콘솔 입력
0 CLICK_AD1                           ← 콘솔 입력
Worker(0):Matched AD1                 ← 콘솔 출력
1 CLICK_AD3                           ← 콘솔 입력
1 CLICK_AD2                           ← 콘솔 입력
Worker(1):Matched AD2                 ← 콘솔 출력
...
```

평가대상 프로그램 정상 실행, 콘솔 출력 결과

문제

2. 위 1번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 '실시간 스트리밍 시스템'을 구현하시오. (15점)

   - Input Queue 입력 변경 – Input Queue 데이터에 Timestamp 추가
   - Worker 실행 변경 – 파라미터에 Timestamp 추가
   - 만료된 Store Item 제거

상세설명

※ Input Queue 입력 변경

- Input Queue 데이터에 Timestamp 추가됨 ('※ 콘솔 입/출력' 참조)
- 입력 형식: <Timestamp> + " " + <Queue 번호> + " " + <Value>
  예> `1000 0 VIEW_AD1 (<Timestamp>: 1000, <Queue 번호>: 0, <Value>: VIEW_AD1)`

※ Worker(라이브러리로 제공되며 이미 프로젝트에 포함되어 있음) 실행 변경

- Worker의 아래 메소드/함수 호출 시 Timestamp 파라미터 추가 (언어별 상세사항은 소문항 홈 아래 README.TXT 참조)

```
(Java) public String run(long timestamp, String value)
(C#) public string Run(long timestamp, string value)
(C) char* (*run)(Worker* this, long timestamp, char* value);
```

❖ 참고 : Worker 실행 로직

① 만료된 Store Item 제거 – 구현 필요 ('※ 만료된 Store Item 제거' 참조)
② 입력된 파라미터와 Store의 내용을 이용하여 로직 처리
③ 입력된 파라미터를 Store에 추가

※ 만료된 Store Item 제거 (제공되는 Worker의 소스코드에 구현 필요)

- Store는 Worker가 입력 데이터를 저장하기 위해 사용하는 문자열 List
- Store Item 저장 형식: <Timestamp> + "#" + <Value>
- 입력된 Timestamp와 Store Item의 Timestamp간의 차이가 만료시간(3000)을 **초과**하면 Store에서 제거하는 기능을 아래의 메소드/함수에 구현 (언어별 상세 사항은 소문항 홈 아래 README.TXT 참조)

```
(Java) public void removeExpiredStoreItems(long timestamp, List<String> store)
(C#) protected override void RemoveExpiredStoreItems(long timestamp, List<string>
     store)
(C) void removeExpiredStoredItems(long timestamp, char* store[1000], int* store_count)
```

※ 콘솔 입/출력

- 입력 포맷 : Input Queue 데이터 (형식: <Timestamp> + " " + <Queue 번호> + " " + <Value>)
- 출력 포맷 : Worker 실행 결과 (형식: <Worker 실행 결과>)

```
C:\>SP_TEST<엔터키>                    ← 구현한 프로그램 실행 (Argument 없음)
1000 0 VIEW_AD1                        ← 콘솔 입력
1500 0 VIEW_AD2                        ← 콘솔 입력
2000 1 VIEW_AD3                        ← 콘솔 입력
2500 1 VIEW_AD4                        ← 콘솔 입력
4000 0 CLICK_AD1                       ← 콘솔 입력
Worker(0):Matched AD1                  ← 콘솔 출력
5000 1 CLICK_AD4                       ← 콘솔 입력
Worker(1):Matched AD4                  ← 콘솔 출력
6000 0 CLICK_AD2                       ← 콘솔 입력
7000 1 CLICK_AD3                       ← 콘솔 입력
...
```

<Timestamp> <Queue 번호> <Value>

- 위 콘솔 입/출력 시 Worker(0)/Worker(1)의 Store 변경 내역

콘솔 입력	콘솔 출력	Worker(0) Store	Worker(1) Store
1000 0 VIEW_AD1		1000#VIEW_AD1	
1500 0 VIEW_AD2		1000#VIEW_AD1 1500#VIEW_AD2	
2000 1 VIEW_AD3			2000#VIEW_AD3
2500 1 VIEW_AD4			2000#VIEW_AD3 2500#VIEW_AD4
4000 0 CLICK_AD1	Worker(0):Matched AD1	1000#VIEW_AD1 1500#VIEW_AD2 4000#CLICK_AD1	
5000 1 CLICK_AD4	Worker(1):Matched AD4		2000#VIEW_AD3 2500#VIEW_AD4 5000#CLICK_AD4
6000 0 CLICK_AD2		~~1000#VIEW_AD1~~ ~~1500#VIEW_AD2~~ 4000#CLICK_AD1 6000#CLICK_AD2	
7000 1 CLICK_AD3			~~2000#VIEW_AD3~~ ~~2500#VIEW_AD4~~ 5000#CLICK_AD4 7000#CLICK_AD3

입력된 Timestamp(6000)와 Store Item의 Timestamp(1000,1500)의 차이가 만료시간(3000)을 초과하여 제거됨

Store에 추가

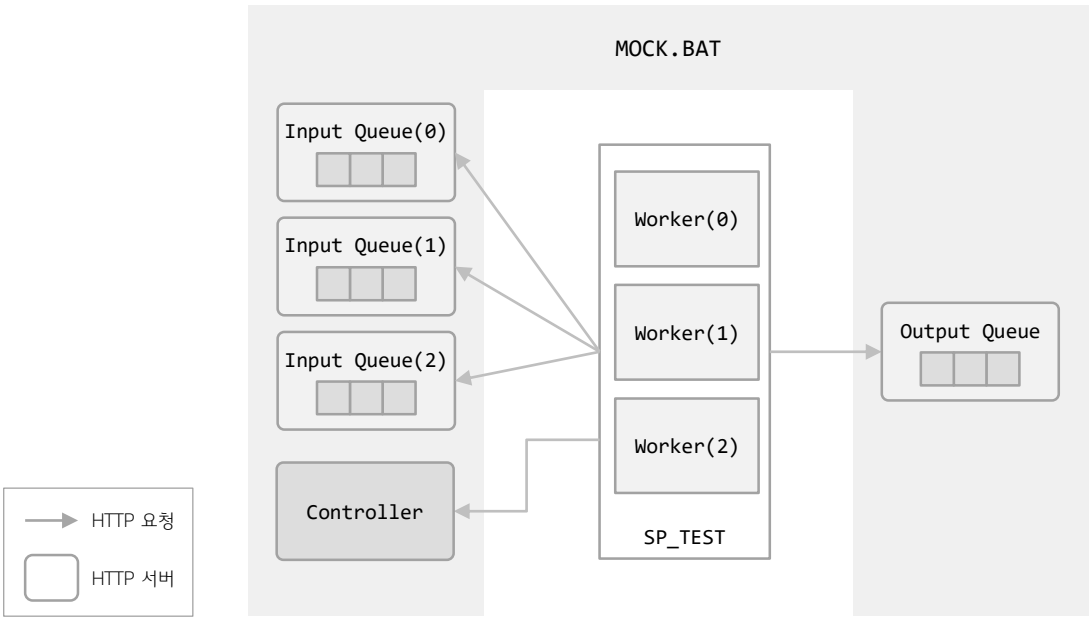
평가대상 프로그램 정상 실행, 콘솔 출력 결과

문제

3. 위 2번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 '실시간 스트리밍 시스템'을 구현하시오. (15점)

- Controller로부터 Input/Output Queue 정보 수신하도록 변경
- Worker 생성 변경 – Input Queue 개수 만큼 생성하도록 변경
- Input Queue 입력 및 Output Queue 출력 변경 – 콘솔 입/출력에서 HTTP 응답/요청으로 변경

상세설명



※ Controller로부터 Input/Output Queue 정보 수신

- (1, 2번 소문항에서 2개로 고정되었던) <Input Queue 개수>와 <Input Queue URI>, <Output Queue URI>를 Controller로부터 HTTP 응답으로 수신
- URI : GET http://127.0.0.1:8080/queueInfo
- 호출방향 : 구현프로그램(SP_TEST) → 제공프로그램(MOCK.BAT)
- 응답 Body : JSON 문자열 형식

```
{  "inputQueueCount":<Input Queue 개수>,
   "inputQueueURIs":<Input Queue URI JSON 배열>,
   "outputQueueURI":"<Output Queue URI>"  }
```

- 예>

```
{  "inputQueueCount":3,
   "inputQueueURIs":["http://127.0.0.1:8010/input",     → Input Queue(0)의 URI
                     "http://127.0.0.1:8011/input",     → Input Queue(1)의 URI
                     "http://127.0.0.1:8012/input"],    → Input Queue(2)의 URI
   "outputQueueURI":"http://127.0.0.1:9010/output"  }
```

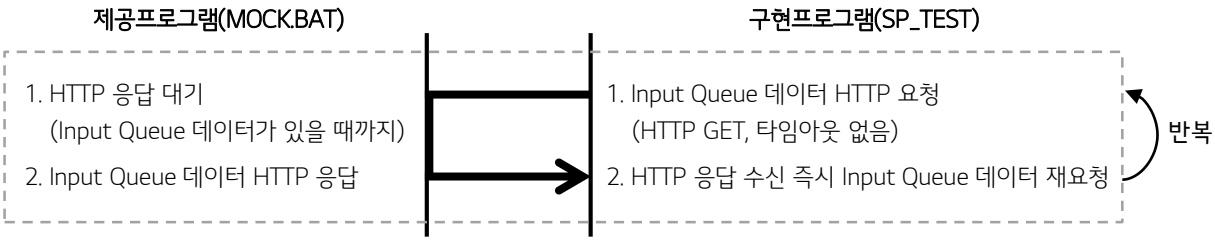
- 동작 : 구현프로그램(SP_TEST) 구동 후 즉시 Controller로부터 정보를 수신해야 하며, 이후 채점 시나리오에 따른 Input Queue 데이터 입력이 시작됨

※ Worker 생성 변경

- Controller도부터 조회한 <Input Queue 개수> 만큼의 Worker 인스턴스를 생성하도록 변경

※ Input Queue 입력 변경

- Input Queue 입력 방법은 콘솔 입력에서 "HTTP 응답"으로 변경
- URI : GET <Input Queue URI> ('※ Controller도부터 Input/Output Queue 정보 수신' 참조)
- 호출방향 : 구현프로그램(SP_TEST) → 제공프로그램(MOCK.BAT)
- 응답 Body : `{"timestamp":<Timestamp>, "value":"<Value>"}`
  예> `{"timestamp":1000, "value":"VIEW_AD1"}`
- 동작 : 구현 프로그램은 멀티 쓰레드를 이용하여 모든 Input Queue들에 대해 병렬로 Input Queue 데이터를 요청/대기하고 응답이 수신되면 즉시 재요청해야 함



※ Output Queue 출력 변경

- Output Queue 출력 방법은 콘솔 출력에서 "HTTP 요청"으로 변경
- URI : POST <Output Queue URI> ('※ Controller도부터 Input/Output Queue 정보 수신' 참조)
- 호출방향 : 구현프로그램(SP_TEST) → 제공프로그램(MOCK.BAT)
- 요청 Body : `{"result":"<Worker 실행 결과>"}`
  예> `{"result":"Worker(0):Matched AD1"}`
- 응답 Body : 없음

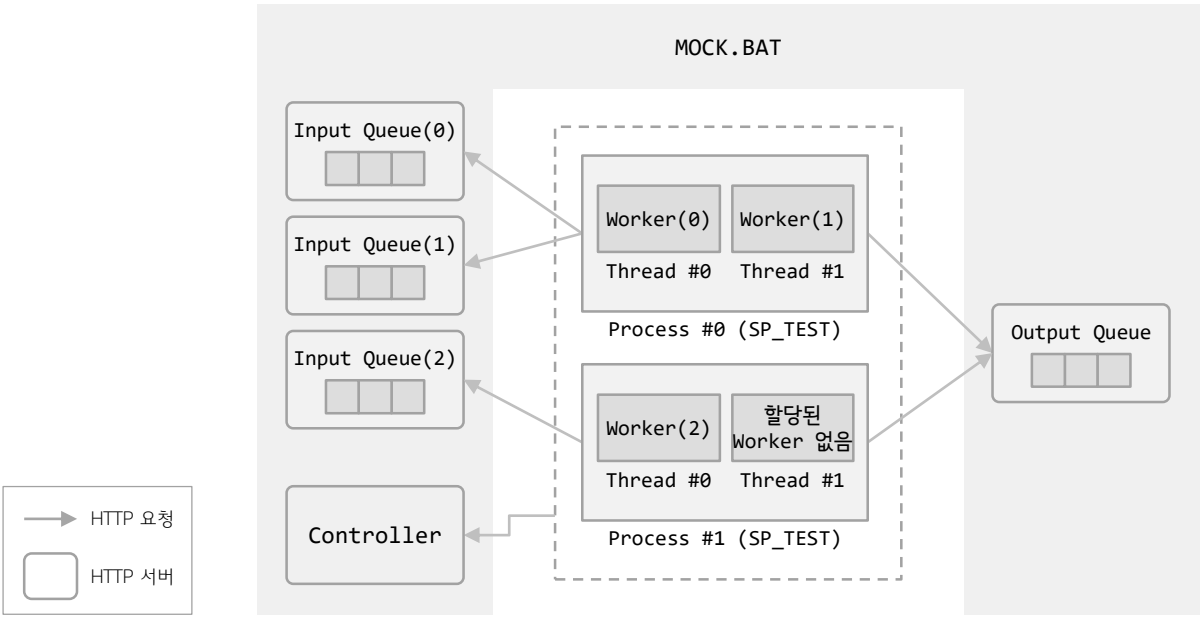
평가대상 프로그램 정상 실행, 제공프로그램(MOCK.BAT) 실행결과 정답/오답 여부

문제

4. 위 3번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 '실시간 스트리밍 시스템'을 구현하시오. (20점)

- Controller로부터 수신한 정보에 따라 Process/Thread/Worker를 생성하고 Worker를 할당하여 실행하도록 변경
- Output Queue 출력 변경 – Worker 처리 결과가 일정 개수만큼 누적되거나 일정 시간이 경과하면 일괄 출력하도록 변경

상세설명



※ Controller로부터 수신하는 정보 추가

- <Process 개수>, <Thread 개수>, <Output Queue 일괄 처리 개수>를 Controller로부터 추가로 수신하도록 변경
- 응답 Body : JSON 문자열 형식

```
{  "processCount":<Process 개수>,
   "threadCount":<Thread 개수>,
   "outputQueueBatchSize":<Output Queue 일괄 처리 개수>,
   "inputQueueCount":<Input Queue 개수>,
   "inputQueueURIs":<Input Queue URI JSON 배열>,
   "outputQueueURI":"<Output Queue URI>"  }
```

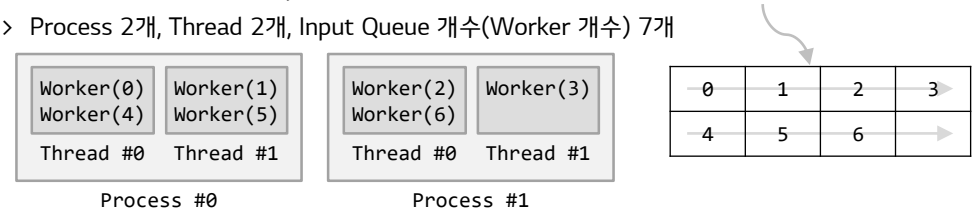
- 예>

```
{  "processCount":2,
   "threadCount":2,
   "outputQueueBatchSize":2,
   "inputQueueCount":3,
   "inputQueueURIs":["http://127.0.0.1:8010/input",
                     "http://127.0.0.1:8011/input",
                     "http://127.0.0.1:8012/input"],
   "outputQueueURI":"http://127.0.0.1:9010/output"  }
```

- 동작 : 어느 Process에서도 여러 번 Controller 호출 가능함

※ Process/Thread에 Worker 할당 및 실행

- Controller로부터 응답 수신이 완료되면 <Process 개수>의 Process를 생성(외부 프로그램 호출 기술 사용)하고 각 Process 마다 <Thread 개수>의 Thread를 생성
- 생성된 Process/Thread에, <Input Queue 개수>의 Worker를 생성하여 순서대로 할당함
- 예> Process 2개, Thread 2개, Input Queue 개수(Worker 개수) 7개



- **Worker는 자신이 할당된 Process/Thread에서 실행되어야 함** (채점 시 Process/Thread Id 체크)
- Worker를 포함하지 않는 보조 Process를 생성해서 활용해도 채점에 영향 없음

※ Output Queue 출력 변경

- 단일 Process에 할당된 모든 Worker들의 실행 결과가 <Output Queue 일괄 처리 개수>만큼 누적되면 Output Queue로 HTTP 요청하도록 변경
- 단, 생성 후 2초를 초과하여 전송되지 않은 Worker 실행 결과가 존재하면 전송 대기중인 모든 Worker 실행 결과를 즉시 Output Queue로 출력
- 요청 Body : {"result":<Worker 실행 결과 JSON 배열>}
- 예>

```
{  "result":["Worker(0):Matched AD3",
             "Worker(1):Matched AD2"]  }
```

※ <Process/Thread 개수>와 <Input Queue 개수>에 따른 Worker 할당 시나리오

- 자가 검수를 위해 4개의 시나리오 제공
- 실행방법 : MOCK.BAT 실행 시 아규먼트로 시나리오 번호 입력 (아규먼트 미입력 시 1번 시나리오 실행)

```
C:\>MOCK.BAT 3<엔터키>                    ← 3번 시나리오로 MOCK.BAT 실행
...
```

시나리오 번호	Controller 수신	Process #0			Process #1		
		Thread #0	Thread #1	Thread #2	Thread #0	Thread #1	Thread #2
1	<Process 개수> 2 <Thread 개수> 2 <Input Queue 개수> 3	Worker(0)	Worker(1)		Worker(2)	할당된 Worker 없음	
2	<Process 개수> 1 <Thread 개수> 3 <Input Queue 개수> 6	Worker(0) Worker(3)	Worker(1) Worker(4)	Worker(2) Worker(5)			
3	<Process 개수> 2 <Thread 개수> 3 <Input Queue 개수> 10	Worker(0) Worker(6)	Worker(1) Worker(7)	Worker(2) Worker(8)	Worker(3) Worker(9)	Worker(4)	Worker(5)

시나리오 번호	Controller 수신	Process #0		Process #1		Process #2	
		Thread #0	Thread #1	Thread #0	Thread #1	Thread #0	Thread #1
4	<Process 개수> 3 <Thread 개수> 2 <Input Queue 개수> 10	Worker(0) Worker(6)	Worker(1) Worker(7)	Worker(2) Worker(8)	Worker(3) Worker(9)	Worker(4)	Worker(5)

평가대상 프로그램 정상 실행, 제공프로그램(MOCK.BAT) 실행결과 정답/오답 여부

문제

5. 위 4번 문항까지 구현된 내용을 기준으로, 아래 사항을 추가로 반영한 '실시간 스트리밍 시스템'을 구현하시오. (10점)

- 구현프로그램(SP_TEST)의 Process가 Kill되면 재실행 기능 추가
- Kill된 Process 재실행시 Store 복구를 위해, Store 백업 및 복구 기능 추가

상세설명

※ Process 재실행

- Controller의 응답에 의해 생성된 Process들 중 하나가 제공프로그램(MOCK.BAT)에 의해 Kill 수행됨
- 구현프로그램(SP_TEST)은 Kill된 Process를 즉시 재실행하고 Worker의 Store를 복구함

※ Store 백업 및 복구

- Kill된 Process 재실행시 Store 복구를 위해 구현프로그램(SP_TEST)은 Controller로부터 최초 응답 수신 후 5초 간격으로 Store 백업
  (Store 외 전송 대기중인 Worker 실행 결과를 포함한 다른 데이터는 백업 불필요)
- 백업된 Store의 내용은 Process Kill에도 안전하게 저장되어야 하며, Process 재실행시 복구에 사용되어야 함 (백업 방법은 자율)
- Store 백업 및 복구를 위한 Worker의 메소드/함수 제공 (언어별 상세 사항은 소문항 홈 아래 README.TXT 참조)
  1) 백업을 위해 Worker의 Store 추출

```
(Java) public List<String> getStore()
(C#) public List<string> GetStore()
(C) char** (*get_store)(Worker* this, int* store_count)
```

  2) 복구를 위해 백업된 Store로 Worker 인스턴스 생성

```
(Java) public Worker(int queueNo, List<String> store)
(C#) public Worker(int queueNo, List<string> store)
(C) Worker* New_WorkerWithStore(int queue_no, char* store[1000], int store_count)
```

평가대상

프로그램 정상 실행, 제공프로그램(MOCK.BAT) 실행결과 정답/오답 여부

폴더 정보

※ 프로그램 및 파일 위치 정보 (실행위치 기반 상대경로 사용 필수)

- 구현할 프로그램 위치 및 실행 위치 : 각 소문항 홈 (SUB1/SUB2/SUB3/SUB4/SUB5)
- 자가 검수용 참고 파일명 : COMPARE 폴더 내 CMP_CONSOLE.TXT (SUB1/SUB2)
- 제공되는 MOCK 프로그램 파일명 : 각 소문항 홈 아래 MOCK.BAT (SUB3/SUB4/SUB5)

*** 제공되는 파일들은 문항에 따라 다를 수 있음**

실행 방식

※ 구현할 프로그램 형식

- 프로그램 형태 : 콘솔(Console) 프로그램
- 프로그램 파일명 : SP_TEST
- 실행 방식(문항1~2) : 콘솔 실행→결과처리 (종료 없음)

```
C:\>SP_TEST<엔터키>                    ← 구현한 프로그램 실행 (Argument 없음)
0 VIEW_AD1                            ← 콘솔 입력
1 VIEW_AD2                            ← 콘솔 입력
0 CLICK_AD1                           ← 콘솔 입력
Worker(0):Matched AD1                 ← 콘솔 출력
...
```

- 실행방식(문항3~5) : 콘솔 실행→실시간 HTTP 요청 처리 (종료 없음)

```
C:\>SP_TEST<엔터키>                    ← 구현한 프로그램 실행 (Argument 없음)
...
```

제공 및 제출

✓ 각 언어별 제공파일 압축 해제 후 자동 생성된 폴더 사용 필수

✓ 제공되는 주요 내용

- 샘플 파일
- 제공 프로그램 실행파일 (MOCK.BAT)
- 제출시 사용할 문항별 폴더 구조

✓ 제출 파일 및 폴더 상세 내용 (각 언어별 실기 가이드 참고)

<주의사항>

제출 파일 관련 내용 (폴더위치, 파일명, 프로그램명 등) 이 틀린 경우 및 상대경로를 사용하지 않은 경우에는 평가 시 불이익이 발생할 수 있으므로 반드시 요구되는 내용과 일치시켜 제출해야 함.

테스트 방법

※ 자가 검수를 위해 제공되는 샘플은 검수용 데이터와 다를 수 있음

자가 검수를 위한 샘플 결과 파일은 각 문항 출력 폴더(COMPARE)에 사전 제공됨

[문항1]

- SP_TEST를 실행한 후 콘솔 입/출력 결과를 샘플 결과 파일(CMP_CONSOLE.TXT)와 동일한지 비교

```
C:\>SP_TEST<엔터키>
0 VIEW_AD1<엔터키>
1 VIEW_AD2<엔터키>
0 CLICK_AD1<엔터키>
Worker(0):Matched AD1
1 CLICK_AD3<엔터키>
1 CLICK_AD2<엔터키>
Worker(1):Matched AD2
...
```

[문항2]

- SP_TEST를 실행한 후 콘솔 입/출력 결과를 샘플 결과 파일(CMP_CONSOLE.TXT)와 동일한지 비교

```
C:\>SP_TEST<엔터키>
1000 0 VIEW_AD1<엔터키>
1500 0 VIEW_AD2<엔터키>
2000 1 VIEW_AD3<엔터키>
2500 1 VIEW_AD4<엔터키>
4000 0 CLICK_AD1<엔터키>
Worker(0):Matched AD1
5000 1 CLICK_AD4<엔터키>
Worker(1):Matched AD4
6000 0 CLICK_AD2<엔터키>
7000 1 CLICK_AD3<엔터키>
...
```

테스트 방법 (계속)

[문항3]

- MOCK.BAT를 실행한 후, SP_TEST 실행
- MOCK.BAT의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.BAT의 콘솔에 테스트 Fail의 사유 또는 Stack trace가 출력되므로 자가 검수에 참고 요망)

```
C:\>MOCK.BAT<엔터키>

[Controller] 요청 대기중 (Controller 요청/응답이 수행되면 채점 시나리오에 따른
Input Queue 입력이 시작됨)
[Controller] HTTP 응답
{  "inputQueueCount":3,
   "inputQueueURIs":["http://127.0.0.1:8010/input",
                     "http://127.0.0.1:8011/input",
                     "http://127.0.0.1:8012/input"],
   "outputQueueURI":"http://127.0.0.1:9010/output"  }


[시나리오] 최초 Controller의 응답이 완료되어 2초 후 채점 시나리오 시작
[시나리오] SLEEP 2초

[Input Queue(0)] 입력 ('{"timestamp":1000,"value":"VIEW_AD1"}')
[Input Queue(0)] HTTP 응답 ('{"timestamp":1000,"value":"VIEW_AD1"}')

...

[Output Queue] HTTP 요청 ('{"result":"Worker(0):Matched AD1"}')

...

[Input Queue(2)] 입력 ('{"timestamp":9000,"value":"CLICK_AD6"}')
[Input Queue(2)] HTTP 응답 ('{"timestamp":9000,"value":"CLICK_AD6"}')

[정답]
테스트에 성공했습니다!
```

❖ MOCK.BAT 출력 내용 참고

- [Input Queue(#)] 입력 ('...') : 채점 시나리오에 따라 Input Queue에 Input Queue 데이터가 입력되어 구현프로그램(SP_TEST)이 데이터를 가져갈 수 있는 상태임
- [Input Queue(#)] HTTP 응답 ('...') : 구현프로그램(SP_TEST)이 Input Queue URI로부터 HTTP 응답을 통해 Input Queue 데이터를 가져감
- [Ouput Queue] HTTP 요청 ('...') : 구현프로그램(SP_TEST)이 Output Queue URI로 HTTP 요청을 통해 Worker 실행 결과를 출력함

테스트 방법 (계속)

[문항4]

- MOCK.BAT를 실행한 후, SP_TEST 실행
- MOCK.BAT의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.BAT의 콘솔에 테스트 Fail의 사유 또는 Stack trace가 출력되므로 자가 검수에 참고 요망)

```
C:\>MOCK.BAT 1<엔터키>                    ← 1번 시나리오로 MOCK.BAT 실행

[시나리오] 1번 시나리오

[Controller] 요청 대기중 (Controller 요청/응답이 수행되면 채점 시나리오에 따른
Input Queue 입력이 시작됨)
[Controller] HTTP 응답
{  "processCount":2,
   "threadCount":2,
   "outputQueueBatchSize":2,
   "inputQueueCount":3,
   "inputQueueURIs":["http://127.0.0.1:8010/input",
                     "http://127.0.0.1:8011/input",
                     "http://127.0.0.1:8012/input"],
   "outputQueueURI":"http://127.0.0.1:9010/output"  }

-------------------------------------------
Process #0            Process #1
-------------------------------------------
Thread #0  Thread #1  Thread #0  Thread #1
-------------------------------------------
Worker(0)  Worker(1)  Worker(2)
-------------------------------------------

[시나리오] 최초 Controller의 응답이 완료되어 2초 후 채점 시나리오 시작
[시나리오] SLEEP 2초

[Input Queue(0)] 입력 ('{"timestamp":1000,"value":"VIEW_AD3"}')
[Input Queue(0)] HTTP 응답 ('{"timestamp":1000,"value":"VIEW_AD3"}')
[SP_TEST의 Worker(0) 실행] Process Id(8848), Thread Id(10)

...

[Input Queue(2)] 입력 ('{"timestamp":5200,"value":"CLICK_AD1"}')
[Input Queue(2)] HTTP 응답 ('{"timestamp":5200,"value":"CLICK_AD1"}')
[SP_TEST의 Worker(2) 실행] Process Id(14132), Thread Id(10)

[정답]
테스트에 성공했습니다!
```

❖ MOCK.BAT 출력 내용 참고

- [SP_TEST의 Worker(#) 실행] Process Id(Process Id), Thread Id(Thread Id) : Worker(#)가 실행된 Process와 Thread의 Id가 제공프로그램(MOCK.BAT)에 의해 수집되어 표시됨

테스트 방법 (계속)

[문항5]

- MOCK.BAT를 실행한 후, SP_TEST 실행
- MOCK.BAT의 콘솔에 출력되는 테스트 결과 확인

(SP_TEST가 문항의 요건을 만족하지 못하는 경우, MOCK.BAT의 콘솔에 테스트 Fail의 사유 또는 Stack trace가 출력되므로 자가 검수에 참고 요망)

```
C:\>MOCK.BAT<엔터키>

[Controller] 요청 대기중 (Controller 요청/응답이 수행되면 채점 시나리오에 따른
Input Queue 입력이 시작됨)
[Controller] HTTP 응답
{  "processCount":2,
   "threadCount":2,
   "outputQueueBatchSize":2,
   "inputQueueCount":3,
   "inputQueueURIs":["http://127.0.0.1:8010/input",
                     "http://127.0.0.1:8011/input",
                     "http://127.0.0.1:8012/input"],
   "outputQueueURI":"http://127.0.0.1:9010/output"  }

-------------------------------------------
Process #0            Process #1
-------------------------------------------
Thread #0  Thread #1  Thread #0  Thread #1
-------------------------------------------
Worker(0)  Worker(1)  Worker(2)
-------------------------------------------

[시나리오] 최초 Controller의 응답이 완료되어 2초 후 채점 시나리오 시작
[시나리오] SLEEP 2초

[Input Queue(0)] 입력 ('{"timestamp":1000,"value":"VIEW_AD3"}')
[Input Queue(0)] HTTP 응답 ('{"timestamp":1000,"value":"VIEW_AD3"}')
[SP_TEST의 Worker(0) 실행] Process Id(12908), ThreadId(10)

...

[Input Queue(2)] 입력 ('{"timestamp":5200,"value":"CLICK_AD1"}')
[Input Queue(2)] HTTP 응답 ('{"timestamp":5200,"value":"CLICK_AD1"}')
[SP_TEST의 Worker(2) 실행] Process Id(11420), ThreadId(10)

[정답]
테스트에 성공했습니다!
```