**Assignment #2: Finding Sequence**

소프트웨어학과 201911204 이주형

**1. Design and implementation**

이 과제의 목표부터 설명을 시작하자면 어떤 주어진 문자열에서 일정한 패턴을 갖는 문자열을 검색한다. 그런데 여기서 구간을 나누어서 병렬적으로 처리할 수 있도록 하고 각 병렬 처리를 쓰레드를 이용해서 시간을 절약하도록 유도하고 마지막에 해당 패턴을 갖는 문자열의 시작 인덱스를 출력하면 된다. 문제 해결에 있어서 조금 더 신경써야하는 부분은 쓰레드로 구간을 나누면 필연치 않게 해당 패턴임에도 불구하고 검색하는 시작 인덱스가 꼬여서 overlap 되는 구간이 생긴다. 즉, 쓰레드 사이에서 overlap 될 가능성을 염두하고 문제 해결에 임해야 한다. 그리고 file offSet을 direct하게 접근하는데 이 때 첫 번째 줄과 개행문자도 연산에서 주의해야한다.

그리고 문자열은 input.txt 파일을 만들어 문자열을 만들어 사용하면 되고 이를 통해 나는 ku\_fs.c 파일에 구현을 후 결과를 output.txt에 담으면 된다.

그러면 천천히 문제 해결을 위한 방법들을 크게 나누어 보고 그 후에 세밀하게 분석하도록 하겠다.

**1. 실행시 주어진 값들을 각 변수에 맞게 담는다.**

**2. 일단 병렬 처리를 하기 위한 쓰레드를 주어진 개수만큼 만든다.**

**2. 그렇게 쓰레드가 n개가 생겼다면 n개의 파이프로 문자열의 구간 별로 overlap 되는 부분을 고려해서 패턴을 검색한다. file offSet은 공유되지 않으니 해당 쓰레드 별로 시작해야하는 구간을 lseek()로 연산해서 고려한다.**

**3. 패턴이 없다면 그냥 넘어가고 있다면 해당 file offSet을 연산하여 그 값을 구현한 오름차순 PriorityQueue 자료구조에 넣는다.**

**4. 오름차순 PriorityQueue에는 insert시 자동으로 오름차순 크기로 값을 유지하기에 따로 sort 할 필요는 없다.**

**5. 그렇게 모든 쓰레드가 위의 연산을 모두 수행하면 main 쓰레드에서 Priority에 저장된 데이터들을 받은 데이터 횟수만큼 output.txt에 write한다.**

위의 내용은 이번 과제의 문제해결에 큰 줄기 부분을 설명한 것이고 이제부터 구체적으로 세세하게 코드 단위로 설명하도록 하겠다. 설명 순서는 최대한 프로그램 진행 순서에 맞게 하도록 하겠다.

1. 시작은 이번 과제를 위해서 사용될 변수 및 상수들을 세팅해야한다. 이 과정이 생각보다 길지만 매우 중요한 부분이다. 먼저 Data라는 구조체 하나를 선언한다. 이 구조체는 나중에 쓰레드에서 이용될 매개변수들을 응축한 구조체 변수이다. 구체적으로는 찾을 문자열인 patternString, 스레드의 개수인 threadCount, 입출력 파일명인 inputFileName, outputFileName, 그리고 진행되는 스레드의 순서를 기록하는 count, 마지막으로 priorityQueue 구조체이다.

2. 기호 상수로 LINECHARNUM와 MAX\_SIZE를 만들어 둔다. 전자는 input.txt에서 라인별 필요 문자 개수이다. 물론 개행문자까지 고려해야한다. 후자는 나중에 PriorityQueue에 사용될 받을 데이터의 수이다. 어짜피 input내의 문자들도 최대가 10000줄이라서 5만개이하의 문자가 존재한다. 그렇기에 최대 5만개의 데이터를 저장할 수 있는 크기를 선언한다. 그리고 나중에 문자들의 위치를 표현하는 데이터를 한 번에 output에 오름차순 정렬해서 출력하기 위한 PriorityQueue 자료구조의 기본 구조체를 선언한다. 마지막으로 mutex를 사용하기 위해서 해당 변수도 선언 및 초기화한다.

3. 이제 변수들에 대한 설명이 끝났으므로 main에서 시작 흐름을 천천히 살펴볼 것이다. 일단 Data 구조체 포인터 변수 data를 선언하고 매개변수로 넘어온 인자들을 가공해서 적절하게 데이터를 저장한다. 그리고 인자값들을 초기화하기 전에 조건에 관련되어 예외 처리를 해준다. argc = arguments count로 main 함수에 전달된 인자의 개수를 의미하며 argv = arguments vector로서 가변적인 개수의 문자열이다. 일단 실행 매개변수로서 전해지는 데이터는 모두 문자열로 취급이 되기에 적절하게 가공해서 사용해야한다. 이 과정을 initData(Data\*, char\*[]) 함수를 통해서 수행한다.

4. 위의 과정대로 초기화가 잘 되었다면 해당 데이터를 이용해서 쓰레드를 만들어야 한다. 해당 함수 내부에 Data 구조체 변수를 선언하고 기존의 내용을 다 복사를 한다. 원본 훼손을 방지하기 위함이다. 쓰레드를 주어진 개수만큼 반복해서 생성하고 실행 내용은 void\* findSequence(void\*) 함수를 통해서 진행하고 관련 매개 변수는 값을 복사한 구조체 변수를 넘긴다.

5. 연속되는 캐스팅을 방지하기 위해서 Data\* 변수를 선언하고 매개 인자를 받는다. firstLineNumber라는 변수는 input.txt의 첫 줄을 받아 몇 개의 라인이 있는지 파악하는 변수이고 input.txt 파일을 open하고 해당 fd값을 fdRead에 저장한다. fdRead에 해당하는 fd를 read한다. 라인정보를 받아오고 전체 라인수를 totalLine에 저장한다. 그런데 라인정보에 순수 숫자가 아닌 whitespace들이 존재할 수 있기에 ltrim()함수로 다 제거해준다. 여기서 왼쪽만 신경쓰는 이유는 애초에 input.txt의 조건에 개행문자가 아닌 공백이 있다면 왼쪽에만 존재하기 때문이다. 어쨌든 이를 저장하기 위함은 마지막 라인에 몇개의 문자가 있을지 미지수이기 때문이다. 따라서 lseek()로 마지막 라인의 첫 번째 offSet으로 이동하여 마지막 줄의 문자들의 개수를 읽어온다. 이렇게 모든 file offSet의 수를 totalChar에 저장한다. 물론 offSet 연산으로 위치를 알아내기 위해서 개행문자도 포함해야한다. 다시 문자가 시작하는 offSet으로 위치를 이동한 뒤 쓰레드당 담당할 offSet의 수를 perThreadNumber에 저장한다. 여기서는 첫 줄을 제거한다. 어짜피 첫 줄은 문자가 아니기 때문에 상관없다. 그런데 각 쓰레드가 모든 문자를 딱 떨어져서 담당하면 좋으나 보통 그런 이상적인 상황은 없어서 마지막 쓰레드에 남은 문자를 모두 넣는 것으로 결정한다. 그리고 쓰레드당 담당해야할 문자들의 개수가 마지막 쓰레드가 나머지 쓰레드가 서로 다른데 어짜피 시작 위치를 연산하기 위한 문자들의 위치가 배수 단위에서는 같다. 즉, 예를들어 3개의 스레드가 10개의 문자를 담당한다고 가정하면 첫 번째 쓰레드는 0에서 2까지의 offSet을 탐색하고 두 번째 쓰레드는 3에서 5까지의 offSet을 탐색하고 마지막 쓰레드는 6에서 9까지의 4개의 offSet을 탐색한다. 그런데 어쨌거나 각 스레드가 탐색을 시작하는 위치는 3의 배수이다. 그리고 마지막 쓰레드를 표현하기 위해서 counting이라는 추가 변수와 그 counting이 쓰레드의 개수가 작을 때까지 값을 1증가시키면서 진행한다. 이는 마지막 쓰레드에 남은 문자들을 모두 몰아서 넣기 위함이다. perThreadChar char[] 변수는 쓰레드에서 patternString과 비교를 위해 저장할 문자열 변수이다. 여기에 보면 strlen(d->patternString)+strlen(d->patternString)/6 + 2개의 인덱스를 넣을 수 있도록 설정했는데 이는 개행문자를 의식해서 연산된 값으로 검색되는 와중에 개행 문자가 있든 없든 일단 비교 문자열의 개수와 개행 문자가 가능한 최대의 수와 마지막에 null값을 넣기 위한 수이다. 예를들어 patternString의 문자수가 2개라고 가정하자 그러면 중간에 개행문자가 1개 들어갈 수 있다. strlen(d->patternStirng)은 2이고 strlen(d->patternStirng)/6는 0이고 +2중에서 하나는 null을 위한 1과 개행 문자의 가능성을 암시한 1이다. 개행 문자가 1개가 들어갈 수 있는 patternString의 문자의 개수는 2~5개이고 2개가 들어갈 수 있는 patternString의 문자의 개수는 6~11이다 3개가 들어갈 수 있는 patternString의 문자의 개수는 12~17이다. 즉 6으로 나눌 때 몫의 값이 추가로 필요하게 된다. 물론 direct로 붙어 있어도 딱히 상관은 없다. 이후에 일단 문자열들을 취합한 후에 조율이 있다. 이는 뒤에서 설명하겠다. 어쨌든 perThreadChar를 char\* bufp에 대입하고 match\_Index를 선언한다. 참고로 이번 문자열 찾기는 수업 시간의 첫 번째 과제의 설명을 참고했다. 이전에 만든 중간 과제에 비해서 수업시간에 설명한 첫 번째 과제가 더 효율적이라고 생각해서 바꾸었다. 이제 문자열을 찾을 것이다. perThreadNumber의 개수만큼 루프를 돌 것이다. 그리고 비교 하기 위해서 offSet 위치를 매번 1씩 증가해서 조정할 것이다. 일단 개행 문자를 의식해서 read를 하고 첫 번째 문자가 같다면 이후의 문자들을 비교하는 식으로 간다. 그전에 비교할 문자열인 bufp를 아까 말한대로 조율한다. 일단 끝에 NULL을 추가한 뒤 완전한 문자열로 만들고 DeleteEnter(char\*) 함수로 개행 문자가 있다면 이를 없애고 연속된 문자열로 변경하고 patternString의 개수만큼의 인덱스에 NULL을 추가한다. 어짜피 같은 문자열만 찾을 것이기에 다른 문자들은 관심밖이고 어쨌거나 비교도 patternString의 문자 개수만큼만 비교하면 된다. 이제 비교를 시작하는데 만약에 다른 문자가 존재하면 flag 변수인 k를 1로 만들고 반복을 종료하고 match\_Index가 끝까지 비교해서 잘 찾으면 bufp에 null을 추가하고 반복을 종료한다. 반복이 끝나서 잘 찾았으면 flag 변수에 의해서 offSet을 연산하여 해당 문자의 index 기준의 위치를 저장해서 mutex 설정 후 PriorityQueue에 넣는다. 그리고 mutex를 unlock해서 빠져나가고 다음 문자를 비교한다. 못 찾았으면 flag 변수에 의해서 그냥 다음 문자를 비교하게 된다. 해당 쓰레드가 담당 문자의 개수를 다 확인해보았다면 fdRead를 close한다. 이렇게 findSequence 함수가 끝나고 이후에 pthread\_join 시스템 콜로 reaping한다. 물론 예외 처리는 당연히 해둔다.

6. 쓰레드를 생성하고 해당 쓰레드들이 오류 없이 모든 일을 잘 처리했다면 priorityQueue에 자동 정렬해서 잘 들어갔을 것이다. 그러면 output.txt에 한 번에 write하면 끝이다. 해당 함수는 wirteData(Data \*) 함수에 구현했다. 단순히 output.txt 파일을 연다. 그런데 여기서 반복해서 write를 시도할 것이기에 O\_APPEND를 설정한다. 그리고 다른 것이 있다면 백지상태로 만들기 위해서 O\_TRUNC도 추가 설정을 한다. 그리고 priorityQueue내의 데이터 개수가 0이 될 때까지 데이터를 pop하고 문자열로 변환해서 해당 output.txt에 작성하고 개행 문자 추가하는 방식을 반복하고 마지막은 개행 문자를 추가하지 않도록 제어한다. 다 끝나면 fdWrite를 close()한다

**2. Function description**

제가 사용한 함수는 총 9가지 입니다.

3개는 PriorityQueue를 구현하기 위한 함수 3개이고 남은 함수들은 문자열을 찾기위한 함수입니다.

|  |  |  |
| --- | --- | --- |
| nodeChange | int \*a, int \*b | PriorityQueue에서 데이터 비교 후 두 노드 값을 교환하기 위함 |
|  | return value = void |  |

|  |  |  |
| --- | --- | --- |
| push | priorityQueue\*, int | int의 데이터를 priorityQueue에 삽입 |
|  | return value = void |  |

|  |  |  |
| --- | --- | --- |
| pop | priorityQueue\* | int의 데이터를 priorityQueue에서 값을 제거하면서 반환 |
|  | return value = int | PQ에서 오름차순 정렬로 맨 앞의 값 반환 |

|  |  |  |
| --- | --- | --- |
| initData | Data\* | 초기화할 Data\* 매개변수 |
|  | char \*[] | 시작시 받는 인자로 Data\* 변수에 초기화해줄 변수 |
| return value = void |  |

|  |  |  |
| --- | --- | --- |
| makeThread | Data\* | 쓰레드에 사용할 Data\* 매개변수 |
|  | return value = void |  |

|  |  |  |
| --- | --- | --- |
| writeData | Data\* | output.txt에 write 할 때 해당 값을 전달할 Data\* 매개변수 |
|  | return value = void |  |

|  |  |  |
| --- | --- | --- |
| findSequence | void\* | 쓰레드를 실행할 함수가 사용할 인자 |
|  | return value = void\* | 쓰레드를 실행한 함수가 끝나고 전달할 인자값(해당 함수에서는 사실상 NULL) |

|  |  |  |
| --- | --- | --- |
| ltrim | char\* | 왼쪽에 공백이 있는 문자열을 받는 매개변수 |
|  | return value = char\* | 인자로 넘어온 문자열에서 왼쪽 공백을 모두 제거한 리턴값 |

|  |  |  |
| --- | --- | --- |
| DeleteEnter | char[] | 문자열에 개행문자를 제거할 문자열을 받는 매개변수 |
|  | return value = char\* | 인자로 넘어온 문자열에서 개행문자를 모두 제거한 리턴값 |