시스템 프로그래밍

Assignment #2 Finding Sequence

201910162 컴퓨터공학부 문무현

Design and Implementation

프로그램 구현의 아이디어를 1차 때와 다른 방향으로 잡았습니다.

**이번 과제에서 구현한 방식은 1차 때와 비교했을 때 병렬성이 떨어지거나 경우에 따라서 없다고 볼 수도 있습니다. 하지만 제가 1차 때 구현했던 방식(구간을 나누어 할당하고 탐색 문자열의 길이만큼 겹치는 영역을 만듦)을 그대로 사용하기 보다는 수업 시간에 새롭게 배운 내용(파일 오프셋의 공유)을 활용하여 다른 방식으로 구현해보고자 기존에 사용했던 방식을 다시 사용하지 않았습니다.**

실행 시 오류로 처리하고 프로그램을 종료하는 경우는 다음과 같습니다.

1. 인자의 개수가 정확하지 않은 경우
2. 인자로 들어온 쓰레드의 개수가 정수가 아닌 경우

다음의 경우들은 프로그램이 정상적으로 동작하지 않기 때문에 예외로 처리하였습니다. 이외의 실행 중 오류 발생 시 perror() 함수를 통해 오류 메시지를 출력하도록 하였습니다.

pthread\_create() 시스템 콜로 호출되는 함수는 readFile() 함수입니다. readFile() 함수에서는 dup()을 통해 메인 함수에서 오픈한 읽기 위한 파일 디스크립터와 같은 파일 엔트리를 가리키도록 했습니다. 모든 쓰레드가 파일 오프셋을 공유하는 것이 핵심입니다.

또한 쓰레드들은 전역변수인 INDEX와 COUNT를 공유합니다. INDEX는 read() 시스템 콜이 알파벳 대문자를 읽을 때마다 증가합니다. read() 시스템 콜을 통해 파일을 읽을 때는 한 문자씩 읽고, 읽는 즉시 인자로 전달받은 탐색 대상 문자열과 비교합니다. 읽어온 값이 대상 문자열의 첫번째 문자와 일치하면 COUNT가 1 증가하고, 불일치하면 0으로 초기화됩니다. 파일에서 문자를 한 개씩 읽어오는 즉시 대상 문자열과 비교하고, 전역 변수의 값을 갱신하기 때문에 파일을 읽어온 값을 저장할 공간을 매우 작게 만들 수 있습니다. 또한 파일을 읽는 작업과 문자열을 탐색하는 작업을 동시에 하기 때문에, 파일을 전부 읽고 탐색하는 것보다 빠를 것이라고 생각합니다.

쓰레드들이 생성되어 파일 디스크립터를 복사한 후에는 무한반복문으로 진입합니다. 이 무한반복문 안에서 파일의 끝까지 한 글자씩 읽고 비교하는 작업을 하게 됩니다. 반복문 진입 후 read() 시스템 콜을 호출하고 전역변수의 값을 변경하기 때문에 pthread\_mutex\_lock을 사용하여 레이스 컨디션이 발생하지 않도록 했습니다. Pthread\_mutex\_unlock은 한 문자를 읽고 전역변수를 수정하는 작업이 완전히 끝난 이후에 호출됩니다. 한 문자를 읽는 작업을 마칠 때마다 무한반복문 안에 진입한 모든 쓰레드들은 경쟁적으로 다음 문자를 읽는 작업을 하기 위해 대기할 것입니다.

모든 쓰레드가 파일 오프셋과 전역변수를 공유하고 있기 때문에 어떤 쓰레드가 다음 문자를 읽는 작업을 하는지는 상관이 없습니다. 뮤텍스를 사용하지 않고 모든 쓰레드가 동일한 파일 오프셋으로 병렬적으로 읽는 작업을 수행한다면, 속도는 빨라지겠지만 레이스 컨디션으로 인하여 유효한 결과를 얻을 수 없을 것이기 때문에 다음과 같이 구현하였습니다. 뮤텍스로 인하여 쓰레드들이 파일을 읽는 작업을 동시에 할 수는 없지만, 어떠한 이유로 인하여 한 쓰레드가 자원을 뺏긴 경우, 자원의 여유가 되는 쓰레드가 읽는 작업을 이어서 할 수 있을 것입니다. 따라서 작업의 안정성과 약간의 속도 향상이 이루어졌을 것이라고 기대하고 있습니다.

파일이 5글자씩마다 줄 바꿈이 이루어지는 형식이기 때문에, 줄 바꿈으로 인하여 대상 문자열이 나누어진 경우에도 탐색에 실패하지 않도록, 읽은 문자의 아스키 코드 값을 이용해 알파벳 대문자인 경우에만 대상 문자열과 비교하고 COUNT를 변경하도록 하였습니다. 같은 방식으로 파일의 첫 줄에서 제공하는 정보인 파일이 몇 줄로 이루어져 있는가에 대한 정수 정보와 공백 문자가 비교에서 제외됩니다. 파일 오프셋을 쓰레드 별로 직접 나누어 주지 않고, 공유와 경쟁을 통해 구현했기 때문에 해당 정보를 사용하지 않았습니다. 해당 정보를 이용하여 직접 파일 오프셋을 할당한 후 병렬적으로 읽는 것은 1차 때 구현했던 방식과 비슷했기에 제외했습니다. 해당 정보를 통해 모든 쓰레드가 비슷한 작업량을 가지도록(비슷한 수의 문자를 읽을 수 있도록) 구현할까 고민해보았지만, 성능과 효율 면에서 당장 작업이 가능한 쓰레드가 빠르게 다음 작업을 수행하는 것이 낫다고 판단했습니다. 이번 과제를 진행함에 있어서 이 방식대로 구현할 때 해당 정보를 적절히 사용하는 방법을 떠올리지 못했습니다.

COUNT가 대상 문자열의 길이인 TARGETLENGTH와 같아진 경우가 탐색에 성공한 것입니다. 탐색에 성공한 경우 연결 리스트에 시작 인덱스 정보를 삽입합니다. 연결 리스트의 노드 구조체는 평범하게 다음 노드의 주소와, 인덱스를 저장하기 위한 int 변수 하나로 이루어져 있습니다. 모든 쓰레드가 파일 오프셋을 따라서 순서대로 읽고 있기 때문에, 탐색에 성공했을 때의 값은 점점 증가할 것입니다. 따라서 전역변수로 머리 노드를 가리키는 HEAD 노드와 꼬리 노드를 가리키는 TAIL 노드를 만들었습니다. 탐색에 성공하여 새로운 노드를 만든 경우 꼬리 노드 쪽으로 삽입한 후 꼬리 노드의 정보를 갱신합니다. 머리 노드는 처음 만들어진 노드를 가리킨 채로 유지합니다. 따라서 머리 노드로부터 꼬리 노드를 향한 방향으로 보았을 때 오름차순이 될 것입니다.

메인 함수에서는 pthread\_join을 통하여 쓰레드들이 파일을 EOF까지 읽을 때까지 대기합니다. 모든 쓰레드들이 종료된 것을 확인하면 pthread\_mutex\_destory를 통하여 뮤텍스를 소멸시킵니다.

이후 연결 리스트에 저장된 값을 파일에 쓰기 위하여 writeFile 함수를 호출합니다. 인자로 전달받은 파일 이름으로 파일을 열거나 생성합니다. 파일을 오픈할 때 O\_CREAT, O\_TRUC를 사용했기 때문에 전달받은 파일 이름이 없으면 새로 만들고, 이미 존재하는 경우 새롭게 쓸 내용으로 덮어쓸 것입니다. 파일을 쓰는 작업은 연결 리스트의 첫번째 노드를 가리키던 HEAD 포인터를 따라갑니다. 머리 노드의 데이터를 파일에 쓴 이후 동적 할당했던 메모리를 해제하고 다음 노드를 따라가는 과정을 반복합니다. 정수형 데이터를 그대로 파일에 써봤지만 사람이 알아볼 수 없는 형식으로 저장되는 것 같았기에 파일에 쓰는 데이터를 정수형에서 문자형으로 형 변환하였습니다. 형변환은 sprintf를 사용하였으며, 서식문자를 통하여 왼쪽 정렬하여 빈 공간에 의도하지 않은 값이 쓰이지 않도록 하였습니다. int의 최대값 이상을 표현할 수 있도록 넉넉하게 공간을 잡았습니다. 비슷한 방식으로 줄 바꿈 문자도 파일에 넣어서 인덱스를 구분했습니다.

stdio 라이브러리를 사용하지 말라고 말씀하신 교수님의 의도를 파일 입출력과 관련된 라이브러리 함수를 사용하지 말고, 시스템 콜을 사용하라는 뜻으로 이해했습니다. 따라서 터미널에 출력되는 표준 출력은 테스트에 사용한 것을 거의 그대로 지우지 않고 두었습니다. 단, output 파일에 써지는 값은 탐색에 성공한 위치의 인덱스 값뿐입니다. 표준 출력으로 출력되는 메시지는 1. 쓰레드가 언제 생성되었는지 2. 몇 번째 쓰레드가 어떤 문자를 읽었는지 3. 연결리스트에 추가된 값은 무엇인지입니다.

해당 프로그램을 수백 번 반복하여 실행한 결과, 연결 리스트에 추가되는 값은 달라지는 경우가 없으며, 쓰레드가 생성되는 시점은 매 번 달라진다는 것을 볼 수 있었습니다. Input.txt의 크기가 작은 경우 한 쓰레드가 모든 작업을 수행하고(심지어 다른 쓰레드가 생성되기 전에) 종료되는 경우도 있었습니다. 하지만 수업 게시판에 올려 주신 샘플 생성기를 이용하여 입력 파일의 크기를 키운 경우를 보면 생성된 쓰레드들이 모두 작업을 수행하는 경우를 쉽게 찾아볼 수 있었으며, 한 쓰레드가 작업을 시작하면 연속적으로 일하는 경우가 많다는 것을 볼 수 있었습니다. 샘플 생성기로 만들 수 있는 최대 크기의 파일을 입력해도 작업이 순식간에 끝나지만, 그 짧은 동안에 작업을 수행하는 쓰레드가 여러 번 바뀌는 것이 보이는데, 운영체제의 의한 관리 때문일 것이라고 생각했습니다. 운영체제 수업이 기대됩니다.

Function Description

메인 함수, 연결 리스트에 노드를 추가하는 함수, 파일을 읽는 함수, 파일을 쓰는 함수가 존재합니다.

|  |  |  |
| --- | --- | --- |
| Function Name | Arguments | Description |
| Main | int argc | 인자의 개수입니다. |
| char\* argv[] | 인자입니다. 탐색할 문자열과 생성할 쓰레드의 개수, 입력 파일 이름, 출력 파일 이름을 의미합니다. |
| Return Value | 정상 종료 시 0, 비정상 종료 시 -1을 반환합니다. |

|  |  |  |
| --- | --- | --- |
| Function Name | Arguments | Description |
| addNode | int data | 생성된 노드에 추가될 데이터입니다. 인덱스를 저장함으로 정수 타입입니다. |
| Return Value | 반환 값은 없습니다. 실행의 결과로 연결 리스트에 노드가 추가됩니다. |

|  |  |  |
| --- | --- | --- |
| Function Name | Arguments | Description |
| readFile | Void \*arg | 의미 없는 값입니다. |
| Return Value | 반환 값은 NULL이지만 의미가 없습니다. 해당 함수의 실행 결과로 메인 함수의 인자로 전달받은 파일을 읽는 작업을 수행합니다. |

|  |  |  |
| --- | --- | --- |
| Function Name | Arguments | Description |
| writeFile | Char\* outputname | 메인 함수에서 인자로 전달받은 출력 파일의 이름입니다. |
| Return Value | 반환 값은 없습니다. 실행의 결과로 연결 리스트에 존재하는 노드들의 데이터를 출력 파일에 쓰는 작업이 이루어집니다. |