**Assignment #1: 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 파이프를 이용해서 시간을 절약하도록 유도하고 마지막에 해당 패턴을 갖는 문자열의 시작 인덱스를 출력하면 된다. 문제 해결에 있어서 조금 더 신경써야하는 부분은 파이프로 구간을 나누면 필연치 않게 해당 패턴임에도 불구하고 검색하는 시작 인덱스가 꼬여서 overlap 되는 구간이 생긴다. 즉, 파이프 사이에서 overlap 될 가능성을 염두하고 문제 해결에 임해야 한다.

그리고 주어진 문자열은 ku\_fs\_input.h 헤더파일의 문자열을 사용하면 되고 이를 통해 나는 ku\_fs.c 파일에 구현을 하면 된다.

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

**1. 일단 병렬 처리를 하기 위한 자식 프로세스를 주어진 개수만큼 만든다.**

**2. 그렇게 자식 프로세스가 n개가 생겼다면 n개의 파이프로 문자열의 구간 별로 overlap 되는 부분을 고려해서 패턴을 검색한다.**

**3. 패턴이 없다면 그냥 넘어가고 있다면 해당 시작 인덱스를 연산하여 그 인덱스를 부모 프로세스에게 보내도록 파이프에 write를 수행한다. 물론 자식 프로세스에서는 read할 필요가 없으니 해당하는 file description은 close 시켜준다.**

**4. 그렇게 모든 자식 프로세스가 위의 연산을 모두 수행하면 부모 프로세스에서 파이프에 있는 데이터들을 받는다.**

**5. 여기서 또 중요한 것은 해당 패턴의 인덱스의 순서를 보장할 수 없기에 일단 해당 인덱스들을 배열에 저장하고 배열 내에서 sort를 진행한다.**

**6. 여기서 sort는 여러가지 방법이 있겠으나 그나마 성능이 가장 좋은 qsort를 사용하는 것이 빅 오 관점에서 가장 좋다. O(nlogn)안에 해결할 수 있기 때문인데 해당 과제는 일단 그 부분이 중점적이지도 않고 주어진 문자열에 대해서 비교적 데이터 수가 간단하고 적기에 그냥 선택정렬로 해결 하도록 한다.**

**7. 부모에 해당하는 file description을 모두 사용했다면 마찬가지로 close를 시켜준다.**

위의 내용은 이번 과제의 문제해결에 큰 줄기 부분을 설명한 것이고 이제부터 구체적으로 세세하게 코드 단위로 설명하도록 하겠다.

일단 main 내부에는 1개의 함수가 실행되고 그 함수 내부에는 또 2개의 함수가 실행이 되는 구조이다. main 내부에 실행되는 solvingProblem(int argc, char\* argv[])는 사실상 문제해결함수로 볼 수 있다. 매개변수로 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.

그래서 내부에서 str 변수에는 주어진 문자열에서 찾을 패턴을 저장하고 i 변수에는 생성할 자식 프로세스 즉, 생성할 파이프의 개수이므로 적절하게 가공해서 int 타입으로 바꾼다.

그리고 파이프fd를 저장할 변수와 나중에 waitpid로 자식 프로세스의 상태 정보를 담을 child\_status를 생성하고 여기서 이제 파이프를 생성할 것인데 그렇기 위한 pipe() system call을 사용한다. 그리고 그 리턴값으로 성공 여부를 체크하기에 checkingPipe라는 변수에 담아서 그것을 매개변수로 파이프가 문제없이 생성되었는지 여부를 체크하는 함수인 checkPipe에 인자로 넣어 파이프 정상 생성 여부를 체크한다. 만약 생성이 실패했다면 checkingPipe에는 -1이 들어가 오류가 떴다고 알리며 종료할 것이다.

그리고 checkSizePipe() 함수로 처음에 파이프 개수를 실행 매개변수에 담아서 사용한다고 했었는데 파이프 개수가 유효한지 여부를 체크하는 함수이다. 내부에는 파이프 개수인 i가 문자열보다 크다면 최대 개수를 문자열의 길이로 설정하도록 했고 0이하로 주어졌다면 최소 개수인 2개로 설정하기로 한다.

그리고 자식 프로세스의 id를 담기 위한 배열 변수를 선언한다.

이제부터 패턴을 찾는 알고리즘의 핵심인 processExe라는 변수가 있는데 이것은 MAXS/i -1로서 파이프당 처리할 기본 string 개수이다. 즉 MAXS가 99이고 i가 4이면 processExe 24인데 인덱스는 0부터 시작하니 1을 빼주어 23의 값을 갖게 되는 것이다.

strlen은 패턴의 길이이고 나중에 패턴을 검색해서 해당 주소값을 리턴 받도록 유도하는 ptr 포인터 변수, 그리고 인덱스를 저장해서 파이프에 write해서 보낼 index 배열까지 선언을 완료한다.

이제 i만큼 for 문을 통해서 자식 프로세스를 생성할 것인데 생성하고 자식 프로세스이면 파이프에서 패턴 검색을 하도록 한다. 그런데 여기서 첫 번째로 생성된 자식 프로세스인지 위의 예시로 가면 MAXS가 99이고 i가 4이면 0~23, 24~47, 48~71. 72~99까지 4개의 자식 프로세스가 패턴을 검색하게 된다. 여기서 첫 번째 자식 프로세스와(0~23을 검색하는 프로세스) 마지막 자식 프로세스와 (72~99을 검색하는 프로세스) 그리고 마지막으로 그 사이 프로세스로 나누어서 연산이 진행이 된다.

굳이 이렇게 나누어서 하는 이유는 이제 첫 번째 자식 프로세스에서는 overlap 되는 부분을 고려하지 않고 사이 프로세스와 마지막 프로세스에서만 overlap 되는 부분을 고려하려고 유도하는 부분이고 문자열을 나눌 때 보면 마지막 프로세스와 그 이외의 프로세스가 검색하는 문자열의 길이가 달라질 수도 있기에 마지막 프로세스도 따로 케이스를 나누어서 검색하도록 했다.

사실 내부 연산은 프로세스마다 맥락은 비슷하니 한 번에 설명하도록 하겠다. 일단 첫 번째 자식 프로세스에서 수행할 연산에서는 overlap은 고려하지 않는 것으로 하기로 했으니 strstr 함수를 이용해서 주어진 문자열인 input부터 input+processExe까지 검색을 하고 있으면 그 인덱스를 연산을 한다.(인덱스 연산은 찾은 주소값(ptr) - input 시작 주소값을 빼면 그 인덱스 번호가 나온다.) 만약에 구간내 패턴이 존재하지 않다면 ptr은 NULL을 반환 받아 아무것도 파이프에 write하지 않고 다음 자식 프로세스로 다음 구간에 패턴을 검색하게 된다. 만약 패턴이 구간 내에 존재한다면 그 인덱스를 계산해서 파이프에 그 인덱스를 write 하고 ptr 다음 인덱스부터 다시 패턴을 검색한다. 그렇게 검색하는 부분이 끝나면 write에 해당하는 파이프를 close 하고 마지막에 성공적으로 끝낸다.

이 다음 자식 프로세스와 시작 프로세스 사이에서의 차이는 검색을 시작하는 인덱스가 좀 변경이 된다. 만약에 ABC 패턴을 찾는다고 가정을 했을 때 두 번째 이후 파이프에서 첫 번째 파이프와 똑같은 방식으로 연산을 수행하다보면 overlap 된 부분을 고려할 수 없다. 두 번째 이후 파이프에서부터는 원래 검색하는 부분에서 (패턴의 길이 -1)만큼 더 뒤로 올라간 그곳부터 검색을 한다. 그렇게 되면 overlap된 부분을 모두 고려해서 해당 구간 내의 패턴을 검색 가능하다. 마찬가지로 검색하는 부분이 모두 끝나면 write에 해당하는 파이프를 close 하고 마지막에 성공적으로 끝낸다.

마지막으로 마지막 자식 프로세스에 해당하는 부분은 사이 부분에 생성된 자식 프로세스와 맥락은 같으나 이제 남는 부분의 인덱스까지도 체크를 해야하니 사이 프로세스와 다르게 검색의 끝부분에 대한 제한을 두지 않는다. 그 외의 연산들은 같다. 그리고 마지막에 성공적으로 끝낸다.

이제 자식 프로세스로 패턴을 검색하는 부분이 다 끝나고 부모 프로세스에서 read하는 일만 남았는데 먼저 write에 해당하는 fd를 close 한 후에 자식 프로세스를 생성한 만큼 waitpid로 reaping 해준다. 그리고 성공적으로 reaping 했다면 임시로 만든 k 배열에 파이프에 있는 것들을 모두 read 한다. 그리고 받은 인덱스들을 정렬을 해서 출력하고 해당 file description인 read를 close 하고 프로그램을 성공적으로 끝낸다.

**2. Function description**

제가 사용한 함수는 총 3가지 입니다.

|  |  |  |
| --- | --- | --- |
| checkPipe | int checkingPipe | 생성한 파이프가 정상적으로 생성되었는지 여부 체크하기 위한 pipe() 시스템 콜의 리턴값 |
|  | return value = void |  |

|  |  |  |
| --- | --- | --- |
| checkSizePipe | int i | 처음에 실행 변수로 주어진 자식 프로세스를 생성할 개수 |
|  | return value = void |  |

|  |  |  |
| --- | --- | --- |
| solvingProblem | int argc | argc = arguments count로 main 함수에 전달된 인자의 개수를 의미 |
|  | char \*argv[] | argv = arguments vector로서 가변적인 개수의 문자열이다. 일단 실행 매개변수로서 전해지는 데이터는 모두 문자열로 취급이 되기에 적절하게 가공해서 사용해야함 |
| return value = void |  |