RePORT

제 목 : Fibonacci



|  |  |  |
| --- | --- | --- |
| 과 목 명 | : | 오퍼레이팅시스템 |
| 학 과 | : | 컴퓨터정보공학과 |
| 학 번 | : | 12111453 |
| 담당교수 | : | 송민석 교수님 |
| 제 출 일 | : | 17년 04월 23일 |
| 성 명 | : | 김 우 진 |

1. 개요

- 목표

Thread를 이용하여 Fibonacci 수열을 출력하는 프로그램을 작성한다.

- Fibonacci

F1 = 1

F2 = 1

F3 = F1 + F2 = 1 + 1 = 2

F4 = F2 + F3 = 1 + 2 = 3

F5 = F3 + F4 = 2 + 3 = 5

……..

을 만족하는 수열로, 수학 공식으로 나타낼 시에

f(n) = 1 (n＜=2 일 때)

f(n) = f(n-2)+f(n-1) (n＞2 일 때)

이렇게 표현할 수 있다.

2. 요구되는 개념

- Thread

Thread란 process 내에서 실행되는 흐름의 단위를 말한다. process의 경우 웹서버와 같이 유사한 task가 많은 상황에서도 매번 부모 process의 주소 공간에서 heap, stack, code ,data를 전부 복사 해야하기 때문에 오버헤드가 많이 발생한다. 이 때, thread를 사용하면 register와 stack을 제외한 heap, data, code 부분 등은 공유를 하기 때문에 자원이 절약된다. Thread를 사용하면 해당 프로그램이 block되어도 코드의 다른 부분이 실행 되기 때문에 응답성이 좋다. 앞서 말했듯이 자원을 공유하기 때문에 메모리 오버헤드가 적다는 장점도 있다. 또, thread의 생성과 context switch가 프로세스보다 빠르기 때문에 경제성 측면에서도 좋다. 마지막으로 thread를 사용하면 multi processor를 좀 더 효율적으로 사용할 수 있다. (Sudoku에서 요구되는 개념과 동일)

- Parallelism

병행 수행에는 data parallelism과 task parallelism이 존재한다. Data parallelism은 동일한 데이터의 일부분을 여러 processor에 분산하여 동일한 작업을 하는 방식을 말한다. Task parallelism은 thread들을 각 processor에 분산시켜서 고유한 작업을 수행하게 한다. Thread를 이용한 fibonacci의 경우 n-1의 경우에는 thread 생성을 통하여 동작하고, n-2의 경우에는 재귀함수를 호출하여 연산을 하게끔 구현을 하여, 동일한 데이터의 일부분을 분배 시킨, data parallelism 방식으로 구현이 되었음을 알 수 있었다.

- 필요함수

(열혈 TCP/IP 소켓 프로그래밍 참조)

int pthread\_create ( pthread\_t \*restrict thread, const pthread\_attr\_t \*restrict attr,

void \*(\*start\_routine)(void\*), void \*restrict arg );

- 성공 시 0, 실패 시 0 이외의 값

- thread : 생성할 thread의 ID 저장을 위한 변수의 주소 값

- attr : thread에 부여할 특성 정보

- start\_routine : 별도 실행 흐름의 시작되는 함수의 주소

- arg : 세 번째 인자의 함수가 호출될 때 전달할 parameter 정보의 주소 값

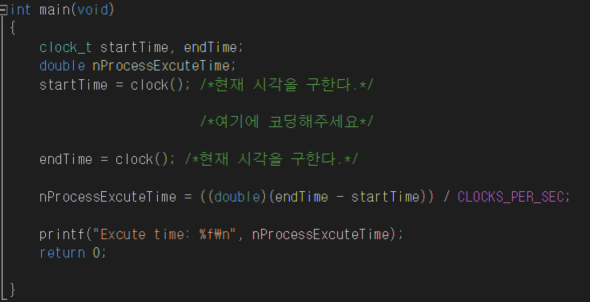
Int pthread\_join ( phtread\_t thread, void \*\*status);

- 성공 시 0, 실패 시 0 이외의 값

- thread : parameter로 전달되는 ID의 thread 종료 시 까지 함수 반환 안됨

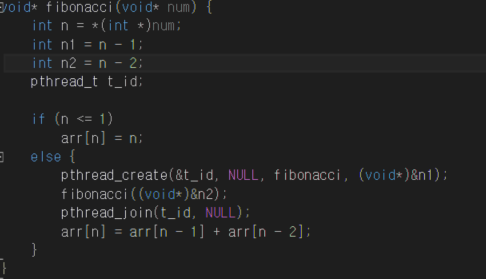
- status : thread가 반환하는 값 저장될 포인터 변수의 주소

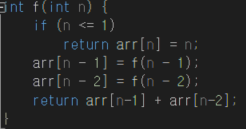
3. 코드설명 및 실행결과

1) 두 가지 구현방식

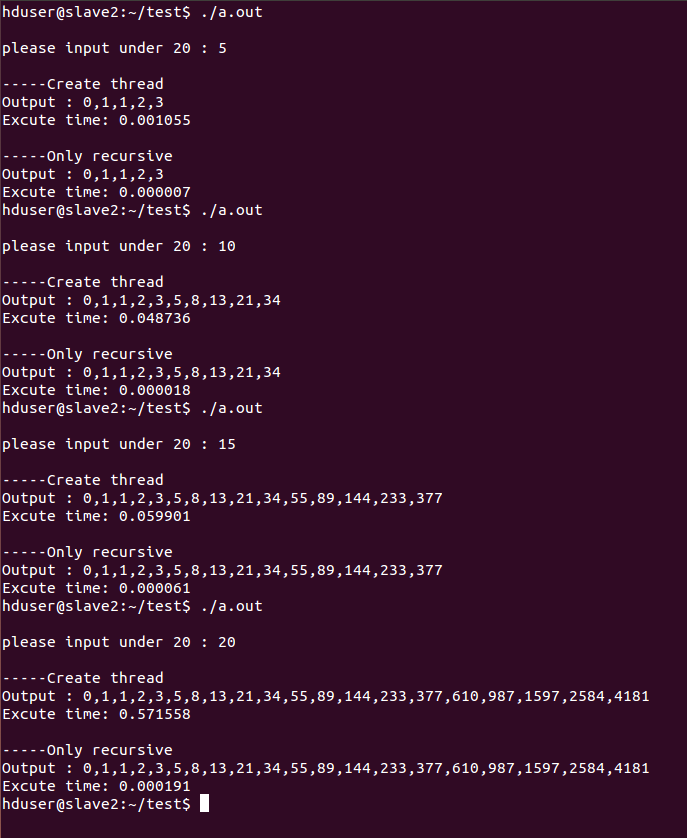
Thread로 구현하였을 시와 일반 재귀함수로 구현하였을 시를 비교하기 위해서 두 가지 방법으로 모두 구현하였다. 그 후 두 방법의 측정시간을 비교하기 위해 <https://www.npteam.net/50>에 나와 있는 방법을 이용하여 시간을 측정하였다. 해당 방법은 그림의 코드와 같다.

2) Thread를 이용한 fibonacci 함수

Thread를 이용하여 생성하는 함수인 fibonacci 함수는 생성 시에 먼저 n-1의 인자를 갖고 수행이 되는 thread를 하나 생성한 후에, 해당 thread의 종료를 기다리는 pthread\_join을 호출하지 않고 재귀적으로 n-2의 인자를 갖는 함수를 호출하는 방식을 취했다. 이 때 재귀함수와 pthread\_join을 둘다 사용하기 때문에 pthread\_exit를 사용할 시에 문제가 발생하였다. 따라서 pthread \_exit을 사용하지 않고 구현을 하였다.

3) 재귀 함수를 이용한 f 함수

최대한 유사한 조건을 갖기 위해서 n-1과 n-2를 인자로 갖는 함수를 바로 호출하지 않고, 값을 배열 arr에 넣고 연산을 진행하는 방식을 채택하였다.

4) 출력내용

첫 번째에는 thread를 통하여 수행하였을 때의 fibonacci 값과 수행시간을 보여주었다. 그 후에 재귀함수로만 진행을 하였을 때의 출력 결과와 수행시간을 보여주었다.

4. 결과

Fibonacci를 thread를 이용하여, 병렬적으로 구현하였을 때, 얼마나 성능이 향상 될지가 궁금했다. 처음 구현을 하였을 때에는 fibonacci 함수 내에서 thread를 생성하고, 그 다음 라인에 바로 pthread\_join을 통하여 종료를 기다리게끔 작성을 하였다. 수행을 하고 보니 이러한 방식은 일반적인 재귀와 차이점이 없다는 것을 직관적으로 알 수 있었다. 그래서 변경한 것이 pthread\_create생성하고 나서 Fibonacci 함수를 호출하고, 그 이 후에 pthread\_join을 호출 하는 방식이었다. 이러한 방식으로 구현을 하였더니 입력 받는 숫자가 20을 넘어갈 경우 segmentation fault가 발생하였다. 이것을 통해서 thread가 생성이 되었으나, fibonacci 함수 구현 시에 너무 많은 thread가 생성이 된다는 것을 알 수 있었다.

분명히 빠르게 실행될 것이라고 예상하였던, thread로 구현된 fibonacci를 실행하였을 때, 입력 값이 조금만 커져도 속도가 느리다는 것이 느껴졌다. 그래서 일반적인 재귀 방식과 비교하였을 때 상대적으로 얼마나 느리게 동작하는 지를 확인해 보았다. 그 결과 많게는 6000배 가량 더 느리다고 확인되었다. 이러한 이유는 pthread\_create함수를 통하여 thread를 생성할 시에 heap, code, data 영역 등이 공유되어 새 process를 생성하는 것보다는 빠르지만, 이것 역시 overhead가 많이 발생한다는 것을 알게 되었다. 이번 구현을 통하여 fibonacci 수열을 조사하는 방법은, 재귀함수를 통하여 연산을 하는 것 보다 thread를 통해서 구현하는 것이 상대적으로 더 느리다는 것을 알게 되었다.