

# Operating Systems

Multi-threaded programming

컴퓨터공학과

12121451

김수환

# < 목 차 >

## 1 . 개요

## 2 . 정의

- (1) Thread
- (2) POSIX Thread API

## 3 . 구현

- (1) Sudoku
- (2) Fibonacci

## 4 . 개발 환경

# 1. 개요

주어진 multi-thread programing 과제를 POSIX Thread API 를 이용하여 구현합니다. 이를 통해 thread 의 개념을 익히고 CPU Scheduling 의 동작 원리를 이해합니다.

## 2. 정의

### (1) Thread

Thread 란 프로세스 내에서 실행되는 흐름의 단위를 말합니다. 일반적으로 하나의 프로세스는 하나 이상의 Thread 를 가집니다. 여기서 환경에 따라 둘 이상의 Thread 를 병렬적으로 수행가능한데, 이를 Multi-thread 라고 합니다.

### (2) POSIX Thread API

POSIX Thread 는 병렬적으로 작동하는 소프트웨어의 작성을 위해서 제공되는 표준 API 입니다. POSIX Thread 를 줄여서 pthread 라고 합니다. 이 pthread API 에서 제공하는 함수는 코드 상단에 #include <pthread.h>를 선언해주면 호출이 가능합니다. 이제 pthread API 에서 제공하는 함수들 중 제가 사용할 자료형과 함수에 관하여 알아보겠습니다.

#### [ 자료형 ]

- pthread\_t  
thread 는 고유의 식별자(tid)를 가지는데 이것으로 각각의 thread 를 구분합니다.  
이 식별자(tid)를 저장할 수 있는 자료형이 pthread\_t 입니다.

#### [ 함수 ]

- pthread\_create( pthread\_t\* tid, pthread\_attr\_t \*attr, (void \*) f, void \*arg)  
새로운 thread 를 생성하기 위한 함수로 pthread\_create()는 다음의 총 4 개의 매개변수를 가집니다.  
첫번째 인자 : 생성하는 thread 의 tid  
두번째 인자 : 생성하는 thread 의 속성값  
(Linux 의 경우 real-time, convention 을 결정합니다.)  
세번째 인자 : 생성하는 thread 가 분기할 함수  
(반환값과 매개변수 모두 (void \*)자료형이므로 (void \*)형으로 선언된 함수만 매개변수로 넘겨줄 수 있습니다.)

네번째 인자 : thread 가 분기할 함수로 넘겨줄 매개변수

(다양한 자료형을 매개변수로 넘겨줄 수 있도록 (void \*)자료형으로 넘겨준 후, 분기하는 함수에서 목적에 맞는 자료형으로 재정의하여 사용합니다.)

pthread\_create() 는 thread 가 정상적으로 생성되었을 경우 0 을 반환합니다.

- pthread\_exit(void \*thread\_return)

현재 실행중인 thread 를 종료하기 위한 함수입니다. 이 pthread\_exit()의 매개변수로 해당 thread 가 수행한 결과값을 pthread\_join()의 매개변수로 넘겨줄 수 있습니다.

- pthread\_join(pthread\_t tid, void \*thread\_return)

부모 thread 에서 pthread\_join()를 호출하면 자신의 state 를 waiting 으로 전환하여, 자식 thread 가 pthread\_exit()를 호출하여 termination 되기를 기다립니다. 또한 자식 thread 가 수행한 결과값을 pthread\_exit()의 매개변수로 넘겨주면 pthread\_join()의 두번째 매개변수로 그 값을 전달 받습니다.

## 3. 구현

### (1) Sudoku

#### ① 요구 조건

- 첫번째 조건 : File 로 Sudoku puzzle number 를 입력받는다.
- 두번째 조건 : 11 개의 자식 thread 를 생성한다. (동작을 확인을 보고서에 작성)
- 세번째 조건 : 자식 thread 는 valid 여부를 main thread 에게 return 하여야 한다.

#### ② 사용한 함수 및 변수 설명 ( pthread 함수들은 ③에서 설명합니다 )

- void \*confirmCol()  
스도쿠 규칙에 위반하는 경우가 없는지 열 단위로 확인하는 함수입니다.
- void \*confirmRow()  
스도쿠 규칙에 위반하는 경우가 없는지 행 단위로 확인하는 함수입니다.
- void \*confirmBlock(void \*Point)  
스도쿠 규칙에 위반하는 경우가 없는지 3x3 크기 구역을 확인하는 함수입니다.  
가운데 위치를 담는 구조체(Point)를 매개변수로 전달하여 확인을 시작합니다.

- typedef struct { int x, y; } Point  
스도쿠에서 3x3 크기 구역의 가운데 칸 위치를 저장할 구조체입니다.
- bool confirmNumber[]  
thread 내에서 9 개의 숫자가 모두 등장하는지 확인하는 배열입니다.
- typedef enum {false, true} bool  
C 언어에서는 boolean 자료형이 존재하지 않으므로 직접 정의하였습니다.

### ③ 동작 원리

먼저 **첫번째 요구 조건**에 의해 input.txt로부터 Sudoku puzzle number를 입력받습니다. 그리고 **두번째 요구 조건**에 의해 11 개의 자식 thread를 생성하기 위해 pthread\_t 자료형으로 11 개의 저장공간을 가진 배열을 선언합니다. 이 중 2 개의 자식 thread는 confirmCol()과 confirmRow()를 수행하도록 pthread\_create()를 호출하여 thread를 생성합니다. 여기서 confirmCol()과 confirmRow()는 모두 (0, 0)의 위치에서 열과 행으로 작업을 수행하도록 구현을 하였습니다. 따라서 매개변수로 넘겨줄 값이 없으므로 pthread\_create()의 4 번째 인자 값은 NULL 이 됩니다. 그리고 나머지 9 개의 자식 thread는 confirmBlock()을 수행하도록 합니다. 여기서 confirmBlock()에 3x3 크기의 블록 구역에서 가운데 칸 위치를 매개변수로 넘겨줘야 합니다. 따라서 가운데 칸 위치의 정보를 담을 수 있도록 선언한 구조체 (Point \*) 자료형의 변수를 동적 할당하여 pthread\_create()의 4 번째 매개변수로 넣어줍니다. 그리고 confirmCol(), confirmRow(), confirmBlock() 이렇게 3 개의 함수는 각각의 confirmNumber[] 배열을 이용하여 9 개의 숫자가 모두 등장하는지 확인합니다. 그리고 모든 11 개의 자식 thread는 말은 구역에서 Sudoku 규칙에 어긋나는 경우가 있는지 병렬적으로 동작하며 확인합니다. 마지막으로 **세번째 요구 조건**에 의해 모든 자식 thread는 말은 작업을 모두 마친 후 pthread\_exit()를 호출하여 매개변수로 valid 여부를 main thread로 전달합니다. 그리고 main thread는 pthread\_join()을 호출하여 자식 thread가 termination 되기를 기다림과 동시에 valid 여부를 나타내는 매개변수를 자식 thread로부터 전달받고 is\_valid[] 배열에 저장합니다. 그리고 모든 자식 thread가 termination 되면 main thread에서 is\_valid[] 배열의 값을 확인하여 11 개의 자식 thread가 valid 즉, true 값을 반환하였는지 확인합니다. 그리고 is\_valid[] 배열의 값이 모두 true 라면 Sudoku 조건을 모두 지킨 것이므로 "Valid result !"를 출력합니다. 그리고 하나의 자식 thread라도 false 값을 반환하여 is\_valid[] 배열 값에 false 가 있다면, Sudoku 조건을 만족하지 못한 것이므로 "Invalid result !"를 출력합니다. **만약에 11 개의 자식 thread가 모두 정상적으로 동작하지 않는다면** is\_valid[] 배열에 잘못된 값이 저장될 것입니다. 따라서 올바른 Sudoku puzzle number가 입력되었음에도 불구하고 "Invalid result !"를 출력할 것입니다. 마지막으로 free()를 호출하여 앞서 선언한 변수인 (Point \*) 자료형의 동적 할당을 해제하고 프로그램을 종료합니다.

#### ④ 결과

##### [ 유효한 스도쿠 예시 ]

```

bash - "suhwankim-ㄱ" x Immediate x +
suhwankim:~/workspace $ gcc sudoku.c -o thread -lpthread
suhwankim:~/workspace $ ./thread
Valid result !
suhwankim:~/workspace $

```

1	5	3	4	6	7	8	9	1	2
2	6	7	2	1	9	5	3	4	8
3	1	9	8	3	4	2	5	6	7
4	8	5	9	7	6	1	4	2	3
5	4	2	6	8	5	3	7	9	1
6	7	1	3	9	2	4	8	5	6
7	9	6	1	5	3	7	2	8	4
8	2	8	7	4	1	9	6	3	5
9	3	4	5	2	8	6	1	7	9
10									

< input.txt >

< 실행 화면 >

##### [ 무효한 스도쿠 예시 ]

```

bash - "suhwankim-ㄱ" x Immediate x +
suhwankim:~/workspace $ gcc sudoku.c -o thread -lpthread
suhwankim:~/workspace $ ./thread
Valid result !
Invalid result !
suhwankim:~/workspace $

```

1	5	3	4	6	7	8	9	1	2
2	6	7	2	1	9	5	3	4	8
3	1	9	8	3	4	2	5	6	7
4	8	5	9	7	6	1	4	2	3
5	4	2	6	8	5	3	7	9	1
6	7	1	3	9	2	4	8	5	6
7	9	6	1	5	3	7	2	8	4
8	2	8	7	4	1	9	6	3	5
9	3	4	5	2	8	6	1	7	9

< input.txt >

< 실행 화면 >

11 개의  
thread 가 모두  
정상적으로  
동작함을  
보이기 위해  
임시로 출력  
해보았습니다.

```

bash - "suhwankim-ㄱ" x Immediate x +
suhwankim:~/workspace $ gcc sudoku.c -o thread -lpthread
suhwankim:~/workspace $ ./thread
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
This thread return value is true
Valid result !
suhwankim:~/workspace $

```

## (2) Fibonacci

### ① 요구 조건

- 첫번째 조건 : 사용자가 원하는 피보나치 수의 개수를 입력한다.
- 두번째 조건 : 자식 thread 내에서 피보나치 수 계산하여 공유 가능한 시퀀스에 계산한 값을 저장한다.
- 세번째 조건 : 자식 thread 가 종료되면 main thread 는 시퀀스 값을 출력한다.

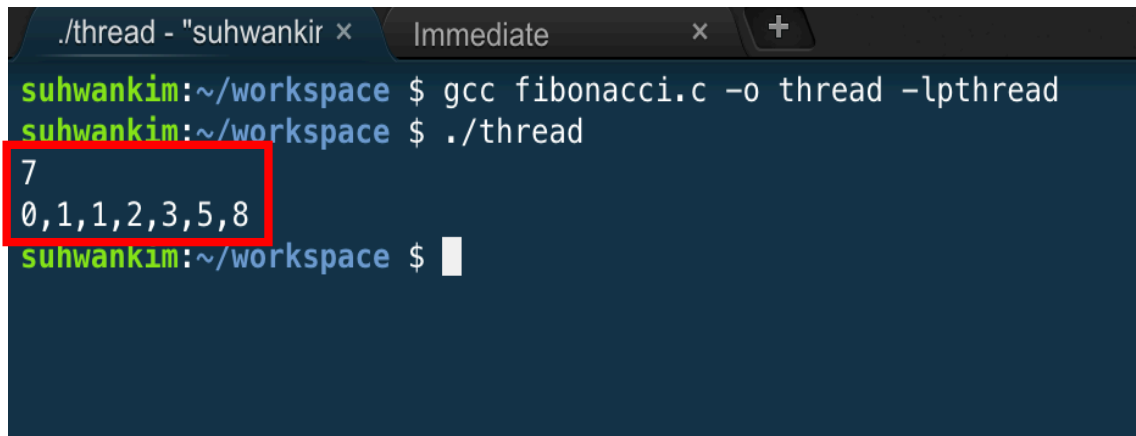
### ② 사용한 함수 및 변수 설명 ( pthread 함수들은 ③에서 설명합니다 )

- void \*generateFibonacci(void \*input)  
사용자가 직접 입력했던 수(n)를 매개변수(input)로 전달하고, input 만큼의 피보나치 수열을 계산하여 공유 가능한 시퀀스에 피보나치 값을 저장합니다.
- int \*fibonacciNum  
피보나치 수열을 저장할 시퀀스입니다. 전역 변수로 선언하여 main thread 와 자식 thread 가 공유 가능하도록 설정합니다. 사용자로부터 피보나치 수의 개수(n)를 입력받고, n 개의 배열로 동적 할당 합니다.

### ③ 동작 원리

먼저 **첫번째 요구 조건**에 의해 사용자로부터 직접 원하는 피보나치 수의 개수(n)를 입력받습니다. 그리고 **두번째 요구 조건**에 의해 피보나치 수열을 저장할 전역 변수로 int \*fibonacciNum 를 선언하고 n 개의 배열로 동적 할당합니다. 그리고 pthread\_t 자료형의 변수를 선언하고, pthread\_create()를 호출하여 generateFibonacci()를 수행하는 자식 thread 를 생성합니다. 이때 매개변수로 사용자가 입력한 수(n)를 전달하고, **두번째 요구 조건**에 의해 generateFibonacci() 내에서 for 문을 통해 n 까지의 피보나치 수열을 계산하고 공유 가능한 시퀀스인 fibonacciNum[] 배열(동적 할당된 배열)에 저장합니다. 그리고 자식 thread 는 자신의 작업을 수행한 후 별도로 main thread 로 전달해줄 값이 없으므로 pthread\_exit()을 호출하고, 매개변수로 NULL 을 main thread 에 전달합니다. 그리고 main thread 는 자식 thread 생성 후에, 곧 바로 pthread\_join()를 호출하여 자식 thread 가 termination 되기를 기다립니다. 역시 main thread 도 자식 thread 로부터 전달받을 값이 없으므로 pthread\_join()의 2번째 매개변수 값으로 NULL 을 입력합니다. 자식 thread 가 termination 되고, **세번째 요구 조건**에 의해 main thread 는 자식 thread 가 계산하여 fibonacci[] 배열에 저장한 n 개의 피보나치 수열을 출력합니다. 그리고 마지막으로 free()를 호출하여 fibonacciNum[] 배열의 동적 할당을 해제하고 프로그램을 종료합니다.

④ 결과



```
./thread - "suhwankir x Immediate x +  
suhwankim:~/workspace $ gcc fibonacci.c -o thread -lpthread  
suhwankim:~/workspace $ ./thread  
7  
0,1,1,2,3,5,8  
suhwankim:~/workspace $
```

A terminal window with a dark blue background. The title bar shows two tabs: './thread - "suhwankir x' and 'Immediate x +'. The prompt is 'suhwankim:~/workspace'. The first command is 'gcc fibonacci.c -o thread -lpthread'. The second command is './thread'. The output is '7' followed by '0,1,1,2,3,5,8' on the next line. The prompt is now 'suhwankim:~/workspace \$'.

< 실행 화면 >

## 4. 개발 환경

OS : Ubuntu 14.04.3

Compiler : gcc version 4.8.4

Language : C