RePORT

제 목 : Sudoku



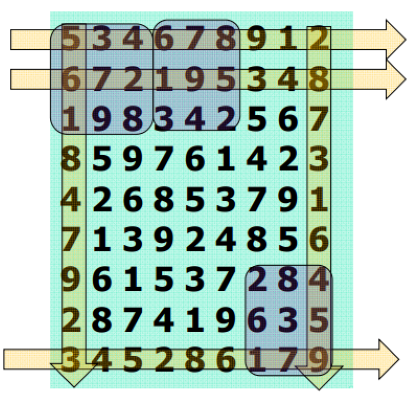
|  |  |  |
| --- | --- | --- |
| 과 목 명 | : | 오퍼레이팅시스템 |
| 학 과 | : | 컴퓨터정보공학과 |
| 학 번 | : | 12111453 |
| 담당교수 | : | 송민석 교수님 |
| 제 출 일 | : | 17년 04월 23일 |
| 성 명 | : | 김 우 진 |

1. 개요

- 목표

Thread와 semaphore를 이용하여 Sudoku 판별 프로그램을 작성한다.

- Sudoku



<homework ppt의 그림>

Sudoku란 가로 세로 9칸인 정사각형 모양의 빈 칸에 1부터 9까지의 숫자를 적당히 배치하여 다음 세 조건을 만족하는 것이다.

1. 각각의 가로 줄에 1~9까지의 숫자가 존재 해야한다.

2. 각각의 세로 줄에 1~9까지의 숫자가 존재 해야한다.

3. 9X9의 정사각형을 3X3의 정사각형으로 나눈 모든 공간에 1~9까지의 숫자가 존재 해야한다.

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. (Fibonacci에서 요구되는 개념과 동일)

- Semaphore

Semaphore란 정수 변수로서, atomic한 두 연산자 wait()과 signal()를 통해 조작된다. Semaphore에는 counting semaphore와 binary semaphore가 존재한다. Counting semaphore는 정수의 값의 제한이 없는 semaphore이다. Binary semaphore는 값이 0과 1만 가능한 semaphore로 공유 영역을 접근하는 critical section에서 발생할 수 있는 race condition과 같은 문제를 해결할 때 사용이 된다. Binary semaphore는 mutex locks라고도 한다.

- 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 이용하여 Sudoku문제를 해결할 경우 각 데이터가 넘겨받은 인자에 따라 동일한 작업을 하는 경우도 있고, 동일하지 않은 작업을 하는 경우도 있기 때문에 data parallelism과 task parallelism을 어느정도 결합하여 동작한다고 판단하였다.

- 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 );

- return : 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);

- return : 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

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

int sem\_destroy(sem\_t \*sem);

- return : 성공 시 0, 실패 시 0 이외의 값

- sem : semaphore 생성시 semaphore 참조 값 저장 위한, 변수의 주소 값 전달

소멸 시에는 소멸하고자 하는 semaphore 참조 값 저장하는, 변수 주소 값 전달

- phared : 0이외 값 전달 시, 둘 이상의 process 접근 가능한 semaphore 생성

- value : 생성되는 semaphore 초기값

int sem\_post(sem\_t \*sem);

int sem\_wait(sem\_t \*sem);

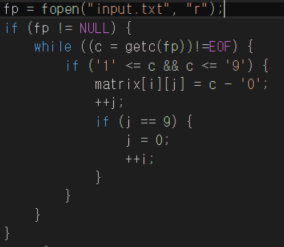
- return : 성공 시 0, 실패 시 0 이외의 값

- sem : semaphore 참조 값 저장하는 변수의 주소 값 전달

- sem\_post : semaphore 값 하나 증가

- sem\_wait : semaphore 값 하나 감소

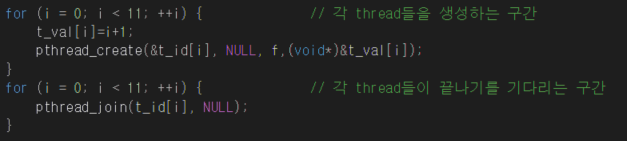
3. 코드설명 및 실행결과

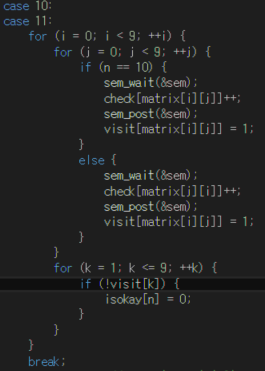


1) 파일에서 input.txt 받기

파일에서 한 글자 씩 읽어서 matrix배열에 글자를 넣는 방식으로 구현하였다. 이 때 한 글자 씩 읽어올 때 개행 문자나 공백의 경우는 필요가 없기 때문에 무시하고 값을 받아오도록 하였다.

2) thread 생성 및 종료

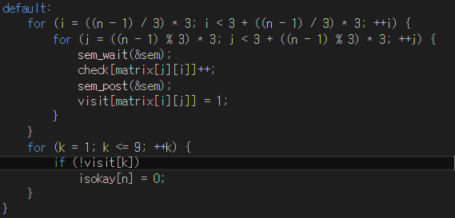
 for문을 통하여 각 thread에 1 ~ 11까지의 숫자를 인자로 넘겨주는 방식으로 구현하였다. Thread를 생성하고 바로 pthread\_join을 할 시에 다음 thread를 생성하지 못하고 thread가 종료되기를 기다려야 하므로, 11개의 thread를 모두 생성한 후에 차례대로 thread가 끝나기를 기다렸다. 이러한 방식이 수업시간에 배운 deferred cancellation이라고 생각하였다. 이 때 thread 종료 시 인자를 넘겨받을 필요가 없기 때문에 pthread\_join과 pthread\_exit시에 인자에 NULL을 두었다.

3) 함수내에서의 동작 방식

- Switch~case문을 이용하여 해당 thread가 넘겨받은 값에 따라서 다른 동작을 하도록 구현을 하였다.

(1) 10 ~ 11

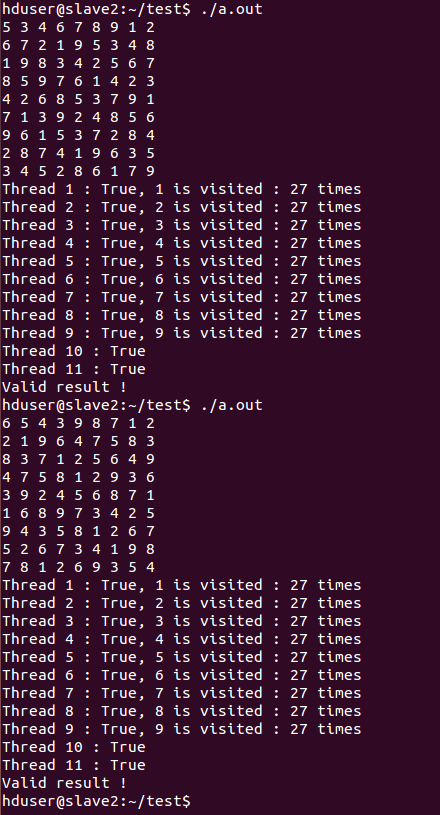
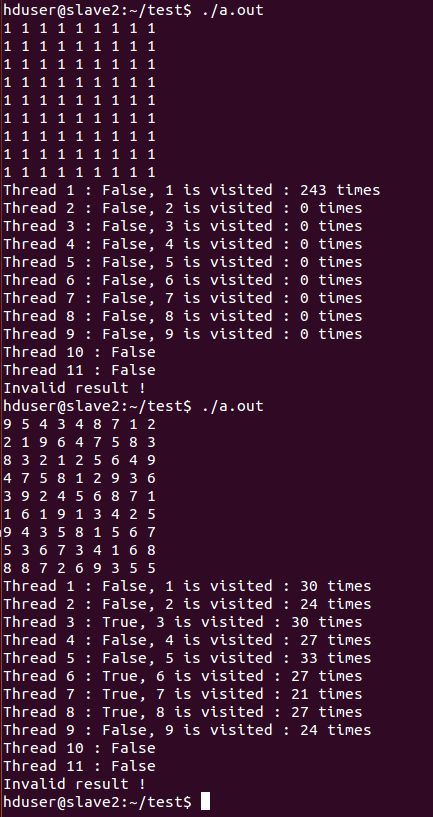
thread가 10 혹은 11을 넘겨 받았을 때, 10의 경우는 각 가로를 검사하고, 11의 경우에는 세로를 검사하도록 하였다. 이 때1~9까지를 조사하여 만약 하나라도 방문 하지 않은 값이 있을 시에 isokay[n]값을 0으로 해주었다. (n은 thread 번호)

(2) 1 ~ 9

1~9까지의 thread의 경우에는 행 시작을 ((n - 1) / 3) \* 3, 열 시작을 ((n - 1) % 3) \* 3로 공식을 만들어서 전부 default로 향하게끔 구현을 하였다. 이 때 각 조사 시 visit배열을 통하여 1~9 까지 전부 존재하는지를 판별하도록 하였다.

4) Semaphore 사용

C:\Users\wooji\AppData\Local\Microsoft\Windows\INetCache\Content.Word\6.png 현재의 구현 상태에서 semaphore가 필요한 상황이 없었기 때문에, 임의로 1~9까지의 값을 몇 번 방문하는지를 검사하는 check 변수를 생성하였다. Valid한 상황에서 값들은 각각 방문 횟수가 27번이어야 한다.

5) 출력내용

처음에 검사할 matrix를 출력하도록 하였다. 그 후에 1번 thread부터 11번 쓰레드까지의 검사 결과와, 1~9까지의 숫자를 각각 몇 번 방문하는 지를 출력한 후에 최종적으로 해당 matrix가 valid한지 invalid한지를 출력하도록 하였다.

4. 결과

처음 구현을 하였을 때는 1~11번 까지의 각 thread를 호출하고, 각각의 thread에서 검사한 구역이 1~9를 전부 포함하였을 시에 1을 return하도록 하였다. 그 후 return 값들의 합을 ans 변수에 저장하도록 하여, ans 값이 11일 때 vaild를 출력하게끔 작성을 하였다. 이에 만족할 수 있지만 구현에 semaphore를 이용해보고 싶은 이유로 인하여, 새로 구현을 하게 되었다. 앞선 구현과 마찬가지로 ans에 값을 저장하는데, 이 때 ans를 전역변수로 선언하여, 각 thread에서 접근하는 방식을 채택하였다. 그러나 이 방식은 ‘반드시 보고서에 11개의 쓰레드가 제대로 동작함 을 기술할 것’이라는 과제의 제약조건에 있어서, 각 thread가 조건을 만족시키는 지에 대한 것을 깔끔하게 보여주지 못하는 코드였기 때문에 semaphore를 다른 방식으로 이용하도록 하였다. 그 방식은 각 thread들이 각 구역을 검사 시에 1~9의 숫자들을 방문 할 때마다 기록을 남기는 것이다. 기록이 남겨질 변수인 check는 공유 변수이기 때문에 semaphore를 이용해서 race condition과 같은 문제를 해결할 수 있었다. 코드의 경우 최대한 가독성이 좋고 깔끔하게 짜보려고 노력한 결과 1~9까지에 해당하는 번호를 토대로 행과 열의 시작과 끝 위치를 나타낼 수 있는 공식을 유도할 수 있었다.