**2022-2 CSED211 Lab08&09 Report**

학번 : 20210479

이름 : 이주현

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

1. **문제별 코드 설명**
   * **Part A**
     + 우선 필요한 헤더를 적어 주었다. writeup pdf에 쓰여있는 <stdlib.h>, <getopt.h>, <unistd.h>를 #include해주고, 추가로 <stdio.h>와 <string.h>를 포함해주었다.
     + 그리고 필요한 struct와 global variable을 선언해 주었다. 코드 상에서 선언한 순서대로 설명하자면 아래와 같다.
       - typedef struct line: cache를 이루고 있는 단위인 line을 구현하기 위한 구조체이다. 각각의 line이 valid bit, tag bit을 포함하도록 한 후, eviction 시에 LRU 방법으로 구현을 할 것이기 때문에 가장 최근에 사용한 시간을 알 수 있도록 편의를 위해 “lruCnt”라는 변수도 포함해 주었다.
       - lru: 캐시 전체의 최신 사용 시간을 표시할 수 있도록 따로 선언해 준 변수이다. 이는 eviction 시에 각 line을 사용한 최신 시간을 표시하는 데에 사용하였다.
       - command line으로 입력받을 s, E, b와 그들로부터 계산할 수 있는 S, B 역시 변수로 선언해 주었다. 또한 v 옵션이 들어왔을 때는 자세한 내용을 출력해야 하므로, 이 옵션이 선언됐는지 확인하기 위해 ‘vb’라는 변수를 하나 더 선언해 주었다.
       - 프로그램 종료 시 printSummary로 hit / miss / eviction 횟수를 출력해야 했다. 이 출력에 용이하게 사용하기 위해 각각을 카운트하는 hit\_ / miss\_ / eviction\_count 변수를 선언해 주었다.
       - cache는 앞서 만든 구조체 line의 2차원 배열로 생각할 수 있으므로 line\*\* cache라 선언해 주었다.
       - trace 파일을 읽기 위한 FILE\* 변수인 traceFile과, trace 파일 중 어떤 것이 필요한지 그 이름을 저장해줄 char\* 변수인 trace를 선언해 주었다. 특히 trace 변수는 예외 처리에도 용이하게 사용되었다.
       - getopt function을 사용하기 위한 변수인 c를 선언해주었다.
     + 프로그램 구현에 필요한 사용자 정의 함수 5개를 선언해주었다. 각각의 역할 및 각 함수가 유기적으로 작동하는 원리는 아래와 같았다. 각 함수가 사용된 곳에는 함수 이름을 bold 처리하여 나타내어, 각 함수의 역할을 알 수 있도록 했다.
     + 우선 프로그램이 시작되면 main에서 parameter로 argc와 argv를 넘겨 parsingData 함수를 부른다. **parsingData**는 command line으로 입력받은 것을 분리해 입력받은 값으로 적절한 global variable들을 업데이트해주는 함수이다.
       - 우선 parsingData에서는 getopt 함수를 이용해 hvs:E:b:t 형태로 옵션 입력이 들어올 것임을 인지하고 각 옵션을 인식해 값을 나누어 준다.
       - 이 값을 int형 변수인 c (global variable)로 받아 switch-case 문을 통해 적절한 동작이 일어날 수 있도록 해주었다. ‘v’ case, ‘h’ case를 제외하고 다른 case에서는 공통적으로 atoi 함수를 이용해 global variable s, E, b, S, B에 알맞은 값을 대입해주는 동작을 수행했다.
       - 추가로 ‘v’ case에서는 vb 값을 1로 바꾸어 v 옵션이 입력되었음을 알렸고, ‘s’, ‘E’, ‘b’ case에서는 예외 처리를 위해 유효한 값이 입력된 경우 cnt라는 지역 변수에 1을 더하도록 했다.
       - ‘h’ case에서는 printMan 함수를 호출해 csim-ref와 같이 설명이 출력되도록 했다. default case의 경우에도 printMan 함수를 호출하도록 했다. 그리고 이 두 case가 호출될 경우, 1이라는 반환값을 가지고 함수를 반환했다.
       - ‘t’ case의 경우 trace라는 global variable에 trace 파일 이름이 저장되도록 해주었다.
       - 혹시 호출 인자가 모두 유효하지 않았을 경우 예외 처리를 위해, s, E, b 변수가 0보다 작거나 trace가 NULL인지 확인하는 조건문을 작성했다. 만약 예외 처리를 해야 할 조건이 맞다면 csim-ref와 같게 오류 문구를 출력하도록 했다. csim-ref와 같게 3가지의 오류 문구를 출력하도록 만들었다. 예외가 발생한 경우에도 1을 반환하도록 했으며, 앞의 ‘h’ case와 중복되지 않는 선에서 역시 printMan 함수로 설명이 출력되도록 했다. 이상의 예외에 해당하지 않으면 0이라는 반환값을 가지고 함수가 반환되도록 했다.
       - **printMan** 함수는 argv를 parameter로 받는 함수이다. 이는 csim-ref에 나온 것과 같은 설명이 출력되도록 하는 함수이다.
     + main에서는 parsingData의 반환값을 받아 해당 값이 0이면 소기의 목적에 맞는 동작을 수행한다. (모든 옵션에 유효한 값이 빠지지 않고 입력되었다는 뜻이므로) 만약 유효하지 않은 입력일 경우, -1을 반환하며 프로그램이 종료되도록 한다.
     + 만약 cache simulation 동작이 수행된다면, 가장 먼저 newCache라는 함수가 호출된다. **newCache**는 앞서 2차원 배열 형태로 선언한 cache를 동적할당하는 함수이다. 먼저 cache를 line\*\* 형으로 동적할당한 후, cache의 각 element를 다시 line\* 형으로 동적할당해 2차원 배열 형태로 동적할당이 이루어지도록 한다. 이후 각 line의 valid, tag, lruCnt 값을 모두 0으로 초기화시킨다.
     + 이후 main에서 trace를 이용해 traceFile로 입력받은 것과 같은 trace 파일을 연다. 만약 이 traceFile이 유효할 경우, 다음 동작을 진행한다.
     + main은 이후 traceFile을 한 줄씩 읽어들인다. 이 동작은 EOF까지 반복적으로 진행된다. 파일이 작성되어 있는 것과 같이 “ %c %llx,%d” 형태로 입력을 받게 되며, 각각을 지역 변수에 저장한다. 이때 ‘I’ 옵션은 읽지 않아도 되므로 파일에서 읽어올 때 앞에 공백까지 읽도록 했다.
     + 그리고 만약 읽어들인 것이 ‘I’ 옵션이 아니고, command line에 v 옵션이 입력되었다면 창에 방금 읽어들인 것을 출력하도록 했다.
     + 각 옵션을 읽은 main은 switch-case 구문을 통해 각 옵션별 동작을 수행하게 된다. ‘L’, ‘S’ case의 경우 findAddr를 한 번, ‘M’ case의 경우 findAddr를 두 번 호출하도록 했다. 그리고 이 구문이 끝난 뒤 역시 만약 방금 읽어들인 것이 ‘I’ 옵션이 아니고 command line에 v 옵션이 입력되었다면 “\n”을 출력해 한 줄을 띄우도록 했다. 이것을 반복해 v 옵션이 입력되었을 경우 csim-ref와 같이 trace 파일 내 각 줄별 상세 내역이 줄을 바꿔 출력되도록 했다.
       - **findAddr**는 실제 cache simulation 동작을 수행하는 함수이다. 이 함수는 trace 파일로부터 읽어들인 addr이라는 unsigned long long int형 변수를 parameter로 받아 동작을 수행한다.
       - 이 함수에서는 addr로 합쳐져 있는 요소들을 targetSet과 targetTag로 나눈다. targetTag의 경우 s, b를 제외한 모든 비트이므로 addr >> (s + b)와 같은 연산을 수행해 구하였다. targetSet의 경우 addr >> b에서 targetTag << s를 빼 구했다. 이때 targetSet은 int로 자료형이 targetTag와 다르므로 int형으로 explicit casting해주는 것을 잊지 않았다.
       - hit, miss, eviction 경우를 구별하기 위해 ifHit, ifMiss, ifEvic이라는 지역 변수를 선언하고 각각의 초깃값을 -1로 주었다.
       - 먼저 hit인 경우를 선별해내기 위해 for문을 돌며 입력받은 것과 같은 index를 갖는 set 내에 valid bit이 1이고 tag가 일치하는 line이 있는지 탐색했다. 만약 그러한 line이 존재할 경우 hit\_count를 1 올리고, ifHit을 1로 바꾸었다. 또한 v 옵션이 입력되었을 경우 “ hit”을 출력하도록 했다. 그리고 해당 line의 lruCnt를 lru로 업데이트 해주었다. 이후 이상의 탐색을 방지하기 위해 break해주었다.
       - 이상의 동작이 끝나면 hit이 되지 않았다는 것을 ifHit을 통해 explicit하게 알아낼 수 있도록 했다. 따라서 이것을 조건으로 하여 ifHit이 음수라면 miss 또는 eviction case에 대한 동작을 수행하도록 만들었다.
       - 우선 hit이 아닌 경우에 대해 공통적으로 입력받은 것과 같은 index를 갖는 set 내에서 for문을 돌며 valid bit이 0인 곳, 즉 비어있는 곳을 찾도록 했다. 비어있는 곳이 있다면 ifMiss를 해당 line의 index로 바꾸고 break했다.
       - 만약 ifMiss가 0 이상이라면 miss situation일 것이고, 0 미만이라면 eviction이 필요한 경우임을 알 수 있다. 따라서 이것을 이용해 miss와 eviction을 구분한 뒤 각각에 맞는 동작을 수행해 주었다.
       - miss인 경우, 우선 miss\_count를 1 올리고, ifMiss를 통해 저장한 line의 index를 사용해 해당 line에 입력받은 tag를 저장해 주었다. 이와 동시에 valid bit을 1로 바꾸고, lruCnt를 업데이트해주었다. 또한 v 옵션이 입력된 경우 “ miss”를 출력하도록 했다.
       - eviction이 필요한 경우, 역시 우선 eviction\_count를 1 올려주었다. 그런 다음 for문을 돌며 해당 set 내에서 가장 마지막으로 쓰인 line을 탐색했다. 이를 위해 lr이라는 새로운 지역 변수를 선언해 가장 오래된 lruCnt를 저장하도록 했고, ifEvic에 해당 line의 index를 저장하도록 했다.
       - 마지막까지 탐색을 끝낸 후, ifEvic으로 저장해둔 line의 index를 이용해 tag와 lruCnt를 업데이트했다.
       - 이상의 동작을 마친 뒤 함수의 끝에서 lru를 1 올려주었다. 이렇게 하면 findAddr이 호출될 때마다 lru가 1씩 오르게 되므로, eviction을 위해 가장 최근에 사용한 시간을 저장할 수 있게 된다. 즉, lruCnt가 클수록 최근에 사용했다는 것을 표시하게 되는 것이다.
     + 이후 모든 동작을 마친 main은 printSummary를 호출해 hit\_ / miss\_ / eviction\_count가 각각 출력되도록 하며, traceFile을 닫는다. 그런 후 deleteCache를 출력해 동적할당되었던 것을 해제하고, 0을 반환하며 프로그램을 종료한다. **deleteCache**는 동적할당되었던 2차원 배열 형태의 변수 cache를 할당 해제하는 변수로, newCache에서 수행했던 것을 반대 순서로 수행하며 모두 할당이 정상적으로 해제되도록 하는 함수이다. 이를 통해 csim.c의 모든 동작을 마치게 된다.
   * **Part B**
     + transpose\_submit에서는 iteration을 위한 변수 i, j와 for문에서 matrix size를 표현하기 위한 maxRow, maxCol, 임시 저장을 위한 t0 ~ t7을 선언해 주었다. 이상의 변수는 총 12개로 writeup에서 지정한 조건을 넘지 않는다.
     + case 1 (M = 32, N = 32)
       - case 1에서는 cache의 line이 각각 32byte라는 것을 이용해 8개씩 입력을 받았다. t0 ~ t7을 이용해 A를 row-wise로 저장하고, 이것을 B에 다시 저장하는 방법으로 cache friendly한 코드를 작성했다.
       - 이렇게 코드를 작성했을 때, 따로 예외 처리를 해 주지 않고도 시간이 단축되어 만점이 나오는 것을 알 수 있다.
     + case 2 (M = 64, N = 64)
       - case 2의 경우, 전체 배열을 잘게 나누어 각각에 대해 transpose를 수행하는 것이 효율적임을 알 수 있다. 그리고 잘게 나눈 각각의 part에서 B 배열의 첫 줄에 해당하는 원소들을 미리 변수에 대입해 놓으면 cache miss 횟수를 더욱 줄일 수 있었다.
       - 첫 줄의 원소들을 대입할 때, spatial locality를 이용해 추가적으로 두 개의 변수를 더 대입해 두었다. 그리고 그것들이 B에 대입되면 다시 spatial locality가 충족되는 조건 하에 다른 A의 원소를 대입해 두어, 시간 효율을 끌어 올렸다. 추가로 i == j인 상황에 대한 예외 처리도 더해 주어 cache eviction을 추가로 피해주었다.
     + case 3 (M = 61, N = 67)
       - case 3에서는 정방형이 아니므로 for문 4개를 사용해 각 element에 차례로 접근해 그것을 B에 저장하는 방법을 사용했다. 접근 방식을 cache friendly하게 하여 효율적으로 처리될 수 있도록 했다.
       - case 3에서는 i == j인 상황에 예외를 두었다. 이는 i == j인 상황에서 A[i][j] = B[j][i]라는 구문으로 바로 값을 대입할 경우 cache에서 eviction이 일어날 수 있기 때문이다. eviction을 최소화하기 위해서 이 경우를 예외로 빼내어 t0, t1 변수를 이용해 각 변수를 거쳐 B로 값이 대입되도록 해주었다. 그러자 만점을 받을 수 있었다.
2. **Lab08&09에서 배운 점**

* Lab08&09 Cache Lab을 해결하면서 cache의 구조와, cache에서 주소를 이용해 hit, miss, eviction 상황에 맞는 동작을 하는 방법을 알게 되었다. 특히 eviction의 한 policy인 LRU 방법의 원리를 파악할 수 있었으며, 해당 방법을 c언어로 구현하는 방법도 알 수 있었다.
* 그리고 상기한 cache의 동작 원리를 이용해 같은 내용을 수행하면서도 효율적으로 동작하게 만드는 cache friendly한 코드를 작성하는 방법도 알 수 있었다. 대략적으로 어떻게 코드를 작성해야 spatial locality를 이용할 수 있을지 생각하는 방식을 알게 되었으며, 깨달은 내용을 바탕으로 코드를 작성해 실제로 cache에 대한 고려 없이 작성한 코드에 비해 월등히 우수한 성능을 보인다는 것을 확인할 수 있었다.