컴퓨터구조 PA4 Cache-simulator
2015-11923 최한결

```c
typedef struct
{
    block *blocks;
} set;

typedef struct
{
    int valid;
    unsigned long long tag;
    int recent_count;
} block;
```

우선 다음과 같이 block들을 담고 있는 set의 type과 각 block의 정보를 담고있는 block의 type을 생성했다.

```c
int main(int argc, char *argv[])
{
    FILE *trace = fopen(argv[argc - 1], "r");
    int hit_count = 0, miss_count = 0, eviction_count = 0;
    int i = 0;
    if (strcmp(argv[1], "-v") == 0 || strcmp(argv[1], "-h") == 0)
    {
        i = 1;
    }
    int s = *argv[2 + i] - '0', E = *argv[4 + i] - '0', b = *argv[6 + i] - '0';
    unsigned long long S = 1 << s;
    set cache_sets[S];
    for (int i = 0; i < S; i++)
    {
        cache_sets[i].blocks = (block *)malloc(sizeof(block) * E);
        // printf("cache_sets: %p   cache_sets %d . block :%p\n", cache_sets, i, cache_sets[i].blocks);
        for (int j = 0; j < E; j++)
        {...
        }
    }
    execute(trace, cache_sets, &hit_count, &miss_count, &eviction_count, s, E, b);
    fclose(trace);
    printSummary(hit_count, miss_count, eviction_count);
    return 0;
}
```

메인 함수에서 전달 받은 인자들을 분석해 각 위치에 맞는 값을 s, E, b에 저장한 후 그 s,E,b값을 이용해 cache를 크기에 맞게 초기화한다. cache의 set 개수는 2^s가지이고 각 set의 block개수는 E가지이다. 이후 trace파일의 명령어들을 실행시킬 execute함수를 호출한다.

```
71  void execute(FILE *trace, set *cache_sets, int *hit_count, int *miss_count, int *eviction_count, int s, int E, int b)
72  {
73      char buffer[100];
74      while (fgets(buffer, sizeof(buffer), trace))
75      {
76          if (buffer[0] == ' ')
77          {
78              // printf("%s", buffer);
79              unsigned long long address = 0;
80              for (int i = 3; buffer[i] != ','; i++)
81              {
82                  int digit;
83                  address = address << 4;
84                  if (buffer[i] >= 'a')
85                      digit = buffer[i] - 'a' + 10;
86                  else
87                      digit = buffer[i] - '0';
88                  address += digit;
89              }
90              unsigned long long set_index = ((address >> b) << (64 - s)) >> (64 - s);
91              unsigned long long tag = address >> (b + s);
92              if (buffer[1] == 'M')
93                  access(hit_count, miss_count, eviction_count, set_index, tag, E, cache_sets);
94              access(hit_count, miss_count, eviction_count, set_index, tag, E, cache_sets);
95          }
96      }
97  }
```

trace파일을 한 줄씩 읽어가면서 명령어를 분석해 S, L, M operation일 때 목적 address를 구한다. address에서 하위 b bit는 offset을 나타내고 상위 s+b bit는 tag를 나타내므로 bit이동을 통해 set의 index와 tag를 추출한다. 이후 명령어가 M일 때는 cache 접근을 두 번, S혹은 L일 때는 cache접근(access 함수)을 한 번 실행한다.

```
17  void update_count(block *blocks, int index, int E)
18  {
19      blocks[index].recent_count = 0;
20      for (int i = 0; i < E; i++)
21      {
22          blocks[i].recent_count++;
23      }
24  }
25  void access(int *hit_count, int *miss_count, int *eviction_count, unsigned long long set_inde
26  {
27      for (int i = 0; i < E; i++)
28      {
29          if (cache_sets[set_index].blocks[i].valid == 0) //miss
30          {
31              // printf("miss ");
32              (*miss_count)++;
33              cache_sets[set_index].blocks[i].valid = 1;
34              cache_sets[set_index].blocks[i].tag = tag;
35              update_count(cache_sets[set_index].blocks, i, E);
36              break;
37          }
```

access함수에서 set_index의 set에 접근 후 그 set의 block들이 하나라도 초기화 되지 않을 때(valid ==0) miss가 일어난다. miss가 일어난 block은 valid가 1이 되고 tag는 받은 tag로 초기화된다. LRU replacement policy를 위한 recent_count는 update_count함수에서 현재 miss가 일어난 block에서 1로 초기화되고 나머지 block에서는 1씩 증가한다.

```
        else
        {
            if (cache_sets[set_index].blocks[i].tag == tag) //hit
            {
                // printf("hit ");
                (*hit_count)++;
                update_count(cache_sets[set_index].blocks, i, E);
                break;
            }
            else if (E == i + 1) //evict
            {
                // printf("miss evict ");
                (*miss_count)++;
                (*eviction_count)++;
                int oldest_index = 0;
                int oldest_count = 0;
                for (int j = 0; j < E; j++)
                {
                    int j_count = cache_sets[set_index].blocks[j].recent_count;
                    if (j_count > oldest_count)
                    {
                        oldest_index = j;
                        oldest_count = j_count;
                    }
                }
                cache_sets[set_index].blocks[oldest_index].tag = tag;
                update_count(cache_sets[set_index].blocks, oldest_index, E);
                break;
            }
        }
    }
```

현재 set의 block을 모두 검사하면서 valid가 1이고 태그가 같은 block을 찾으면 hit이 일어나고 LRU count가 업데이트된다. block들이 모두 초기화 되어있을 때 (모든 block의 valid==1) 찾는 block이 없으면 (block.tag != tag) miss와 eviction이 일어난다. 가장 오래된 block(recent_count가 가장 큰 block)이 대체되므로 recent_count를 비교해 oldest_index를 구하고 tag와 count를 초기화, 모든 block의 count를 1 증가시킨다.