SSE2030: Introduction to Computer Systems (Fall 2010)

Programming Assignment #4:

Implementation of the B+ tree search in IA-32 assembly language

Due: December 20, 11:59PM.

1. Introduction

이 과제는 주어진 키값을 이용하여 B+ 트리에서 데이터를 탐색하는 프로그램을 IA-32 어셈블리 언어를 이용하여 구현하는 것이다. 이 과제의 목적은 트리 자료구조의 일종인 B+ 트리를 이해하고 트리내에서 데이터를 검색하는 알고리즘을 구현하는 과정을 통하여 IA-32 어셈블리 언어를 학습하는데 있다.

2. Problem Specification

2.1. Overview

다음과 같은 원형을 갖는 `bptree_find()` 함수를 IA-32 어셈블리 언어를 이용하여 구현한다. `bptree_find()` 함수는 트리의 헤더 노드를 담은 bptree_node 포인터 1개와 검색할 Key값을 담은 4Byte의 Unsigned Integer형 1개, 그리고 검색된 데이터를 저장할 Unsigned Integer 포인터 1개를 인자로 받고, 검색 결과를 반환한다.

이 때 검색결과는 검색이 되었을 경우 0, 실패했을 경우 그 이외의 숫자를 반환한다.

```
int bptree_find(bptree_node *bptree, unsigned int key, unsigned int* data);
```

첫 번째 인자에 미리 구성된 트리의 정보를 담은 포인터, 찾고자 하는 Key값, 찾은 데이터를 저장할 포인터를 넘기면 그 검색결과를 리턴하고(찾았을 경우 0, 아니면 그 외의 숫자) 그 값을 data에 넣는것이다.

단, B+ 트리 노드들간의 탐색은 리커젼을 이용하여 구현한다. 그리고, 각 노드 내에서 엔트리의 검색은 이진탐색으로 구현하되, 리커젼 구조로 구현해야 한다.

B+ 트리에 관한 자세한 설명은 *[2.2. Background]*에 기술되어 있다.

인자로 넘기는 구조체 포인터 `bptree_node *btree`를 포함한 B+ 트리 구성에 관한 설명은 *[2.3. B+ Tree Specific Design]*에 기술되어 있다.

본 과제 수행에 있어서 별도의 함수를 생성할 필요가 있는 경우, `btree_find.s` 파일 내에 구현할 수 있다.

2.2. Background

2.2.1. B+ tree

B+트리는 키에 의해 식별되는 레코드의 효율적인 삽입, 검색 및 삭제를 통해 정렬된 데이터를 표현하기 위한 트리자료구조의 일종이다.

B+트리는 각 노드가 M개의 서브트리를 가질 수 있는 multiway 트리인데, 여기서 각 노드가 몇 개의 서브트리를 가질 수 있는지 결정하는 M을 트리의 order라 한다. [그림1] 참고.

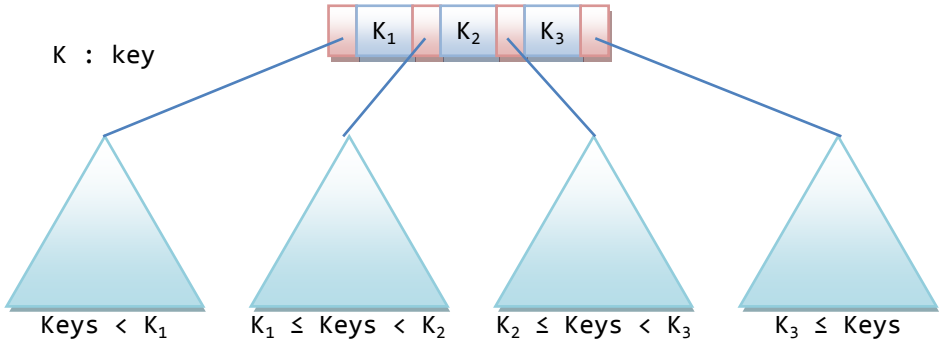


그림 1

B+트리는 크게 두 부분으로 구성되는데, 리프 노드를 찾기 위해 그 경로를 제공하는 인덱스 부분과, 실제 데이터가 들어있는 순차부분으로 구성된다. [그림2] 참고.

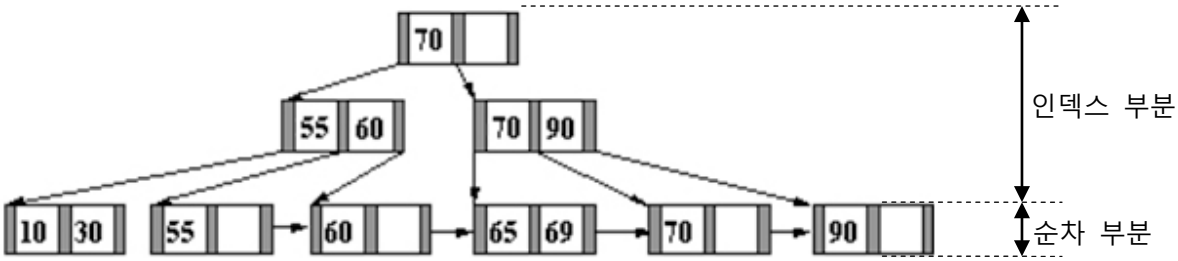


그림 2

트리의 order가 M인 경우 각 노드는 M-1개의 엔트리를 갖게된다.

각 엔트리는 키값(혹은 데이터)과 찾는 키값이 그 값보다 클 경우 찾아갈 서브트리의 포인터를 갖게 되고, 각 노드는 이런 엔트리 M-1개와 첫번째 엔트리보다 작을 경우 찾아갈 서브트리의 포인터를 갖는다. 그리고 검색시 효율을 위해 현재 노드에 몇개의 엔트리가 저장되어있는지 갯수를 저장한다. [그림3] 참고.

```
struct entry {
        int             data;
        struct node*    rightPtr;
}
struct node {
        struct node*    firstPtr;
        int             numEntries;
        struct entry    entries[ORDER];
}
```

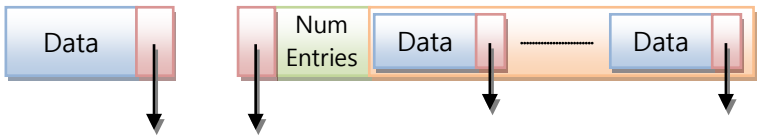


그림 3

다만, 본 과제에서 제공하는 B+ 트리는 구현상의 편의를 위하여 노드의 첫 번째 포인터 firstPtr 대신 키 값 0을 가진 엔트리를 항상 맨 처음에 위치시킨다. (자세한 내용은 *[2.3. B+ Tree Specific Design]* 참고)

B+트리에서 엔트리의 삽입은 아래 [그림4]와 같이 동작한다. Order가 2인 경우의 B+트리 예제이다. (본 과제 수행에 필요한 B+트리는 자동으로 구성되므로 참고할것.)

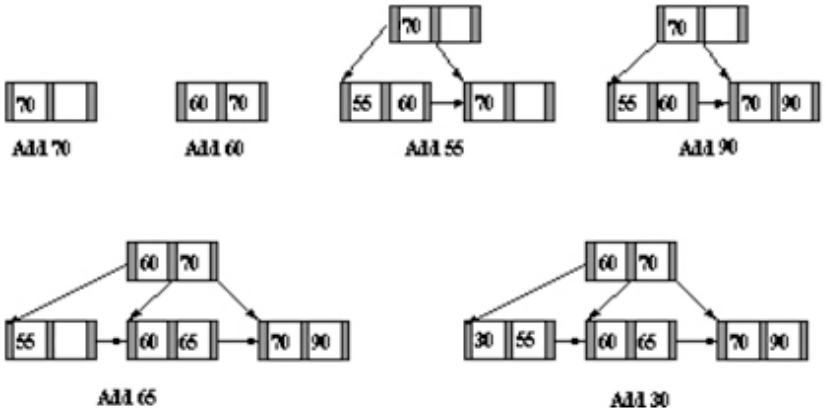


그림 4

최초 70이 노드에 입력되고, 다음 60이 입력된다. 이 때, 노드 내에 엔트리는 항상 정렬된 상태를 유지하여야 한다.

다음으로 55가 입력될 경우, 60보다 앞에 입력되어야 하지만 현재 노드에 55을 입력할 공간이 없다. 이 경우 split연산이 일어나게 된다. 현재 노드를 반으로 나누고, 나뉜 두개의 노드를 구분하는 키값을 상위 인덱스 노드에 추가한다. 인덱스 노드가 존재하지 않을경우 생성하고, 인덱스 노드에 키값을 추가시 노드에 추가 가능한 공간이 없다면, 인덱스 노드 또한 split연산을 수행한다.

2.3. B+ Tree Specific Design

첫 번째 인자인 bptree_node *btree는 실제 B+ 트리의 ROOT 노드 주소를 가리키고 있는데 그 구조체는 아래와 같다.

```
typedef struct bptree_node_t {
    bptree_header  header;
    bptree_entry   entries[ORDER];
} bptree_node;
```

bptree_node_t 구조체에는 해당 노드의 정보를 담고 있는 bptree_header 구조체, 노드 내에 엔트리 정보를 담고 있는 bptree_entry 두 개의 멤버로 구성되어있다.

```
typedef struct bptree_header_t {
    unsigned int    isLeaf;
    unsigned int    nentry;
} bptree_header;
```

노드의 정보를 담고 있는 bptree_header 구조체에는 현재 노드가 leaf 노드인지 여부를 저장하는 Unsigned Integer, 현재 노드 내에 엔트리 개수를 저장하는 Unsigned Integer 두 개의 멤버로 구성되어 있다.

```
typedef struct bptree_entry_t {
    unsigned int key;
    union {
        struct bptree_node_t *branch;
        unsigned int data;
    } value;
} bptree_entry;
```

각 엔트리의 정보를 담고 있는 구조체 bptree_entry는 leaf 노드에 속한 엔트리일 경우 data를, 인덱스 노드에 속한 엔트리일 경우 서브 트리의 노드 포인터를 저장할 수 있는 포인터가 공용체로 들어있고, 해당 엔트리의 키를 저장하기 위한 Unsigned Integer형의 key 변수가 멤버로 구성되어있다.

앞서 B+ 트리의 기본적인 내용을 설명하는 *[2.2.1. B+ tree]*에서 언급한것과 같이 본 과제에서 제공되는 B+ 트리는 노드의 키 중 가장 작은 값보다 작을 경우 하위 노드를 가리키는 포인터 대신 키값 0을 가진 엔트리를 항상 맨 앞에 위치시킨다. 이 구조를 그림으로 표현하면 아래 [그림5]와 같다.

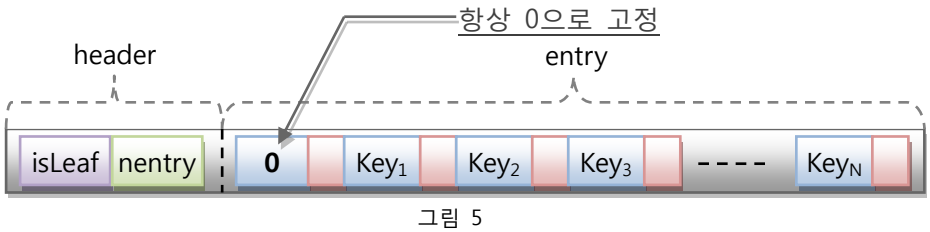


그림 5

2.5. Restrictions

- bptree_find() 함수 내에서는 라이브러리 함수 사용이 불가능하다. 단, 필요에 따라 새로운 함수를 작성하여 사용하는 것은 무방하다.
- bptree_find() 함수 내에서 트리의 노드간 탐색은 리커젼 방식으로 구현한다.
- bptree_find() 함수 내에서 트리의 노드 내 엔트리의 탐색은 이진탐색알고리즘을 이용해 구현하되, 리커젼 방식으로 구현한다.
- bptree_find() 함수의 setup, finish 코드를 제외한 메인 부분 코드 작성시, 사용 가능한 레지스터는 %eax, %ecx, %edx 총 3개로 제한한다. (%ebp, %esp는 가능)
- 과제는 리눅스 환경에서 수행하고, 과제 보고서에 리눅스 환경에서 컴파일하고, 실행한 화면을 캡쳐하여 추가한다.

3. Skeleton Codes

본 과제 수행을 위하여 아래와 같이 3개의 파일이 주어진다.

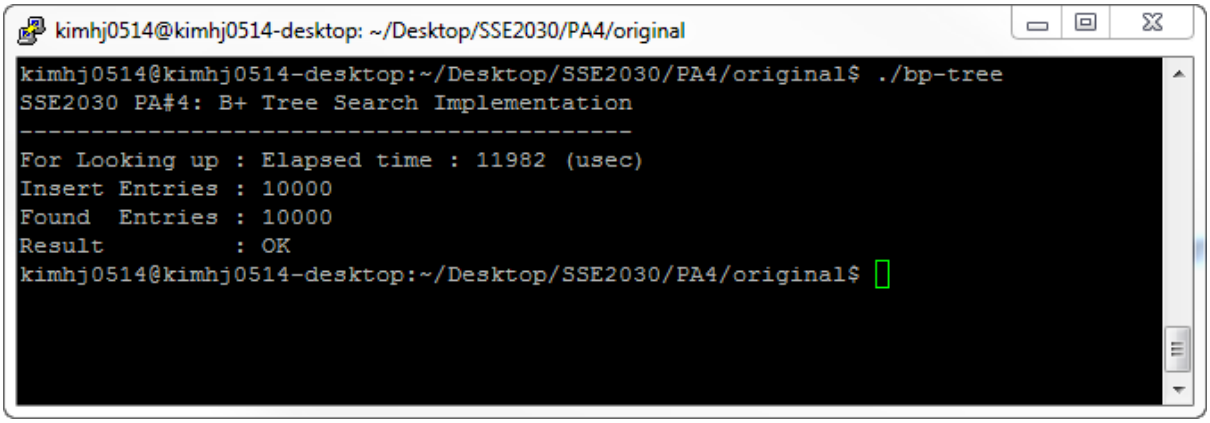
Makefile:	GNU make utility를 위해 사용되는 파일.
main.c:	B+ tree를 초기화하고, btree_find() 함수를 호출하는 main() 함수
bptree.c:	B+ tree 초기화 및 구성을 위한 함수들이 모여있는 파일.
bptree_find.s:	bptree_find() 함수를 구현해 넣을 파일

과제를 위해서는 bptree_find.s 파일만을 수정하면 된다. (bptree_find.s 외에 다른 파일을 추가하지는 말 것)

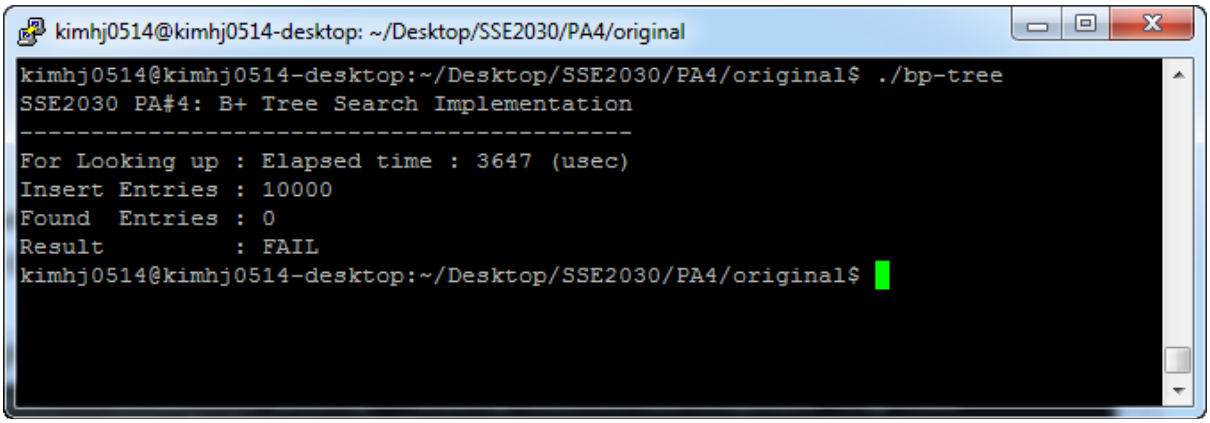
4. Sample Output

프로그램 수행 결과를 입력한 엔트리 갯수, 검색된 엔트리 갯수로 표시한다.

1) 성공시



2) 실패시



5. Hand in instructions

- 작성한 프로그램 코드 상단에 이름과 학번을 주석으로 넣는다.
- 과제 제출시 `bptree_find.s` 파일을 "학번.s" 파일로 이름을 변경하여 제출한다 (예: `2009310123.s`)
- 프로그램의 설계, 구현에 관한 내용을 담은 보고서를 별도로 제출한다. 보고서는 워드, 한글 등의 형식도 상관 없지만 가능하면 PDF format으로 제출하고, 제출시 "학번.pdf" 형태로 파일을 저장하여 제출한다.
- 과제 제출은 프로그램 소스 코드인 "학번.c"와 보고서인 "학번.pdf"를 메일로 첨부하여 sse2030@csl.skku.edu 로 보낸다. 과제 제출시 메일의 제목은 아래와 같은 형식을 따른다.

  [SSE2030] PA#4, 학번, 이름

6. Logistics

- 과제 제출 결과는 본 수업의 홈페이지인 http://csl.skku.edu/SSE2030F10 에서 확인할 수 있다.
- 과제 제출 시간은 메일 도착시간을 기준으로 하며, 기한을 넘겨 제출 할 경우 0점 처리 한다.
- 과제에 대한 의논은 함께 할 수 있으나, 프로그램 소스코드 작성은 스스로 해야 한다.
- 다른 사람의 과제를 copy한 경우, 두 사람 모두 0점 처리한다. 인터넷 등에서 찾은 소스 코드를 그대로 copy한 경우에도 0점 처리한다. 두 번 이상 이와 같은 이유로 0점 처리된 경우 F 학점을 받을 수 있다.

Good luck!

담당조교: 김형준, 031-299-4970, #400629 (hjkim at csl.skku.edu)