**Pintos Project 0-2 :**

**Pintos Data Structure Analysis**

**과목 명 :** [CSE4070-01] 운영체제

**담당 교수 :** 김영재 교수님

**개발 기간 :** 18.09.20 ~ 19.09.22

**20131579 윤기영**

**프로젝트 제목 : pintos project 0-2 : Pintos Data Structure Analysis**

**제출일 : 18.09.23**

1. **개발 목표**

* Pintos Kernel Data Structure 중 list, hash table, bitmap에 특정 기능을 하는 함수를 추가하고, 그것의 동작을 확인할 수 있는 프로그램(Interactive Program)을 작성한다.

1. **개발 범위**

* 각 자료구조(list, hash table, bitmap)을 생성/삭제하는 기능과 각 자료구조의 기능들을 함수로 구현한다.
* Command를 입력 받아 각 자료구조의 기능들 확인할 수 있는 Interactive Program을 작성한다.

1. **개발 방법**

* lib\_hw1 폴더에 있는 bitmap.c, bitmap.h, hash.c, hash.h, list.c, list.h 를 모두 읽고 기존의 라이브러리로 뭐가 있는지 확인하고 무슨 기능을 하는지 알아 둔다.
* 기존 라이브러리를 기반으로 개발해야 하는 기능을 구현한다.

1. **연구 결과**

**추가 Structure**

// list elem과 int형 데이터를 저장하는 structure

struct list\_item

{

struct list\_elem elem;

int data;

};

// hash elem과 int형 데이터를 저장하는 structure

struct hash\_item

{

struct hash\_elem elem;

int data;

};

**추가 Util 함수**

int find\_name\_idx(char input[50], int select);

* 자료구조의 이름과 select 를 넘겨받아 select가 0이면 list에서 해당 자료구조를 찾아 인덱스를 반환한다. 1이면 hash, 2면 bitmap에 대해 수행한다.

**list 라이브러리 및 추가 기능 함수**

#define list\_entry(LIST\_ELEM, STRUCT, MEMBER) \

((STRUCT \*) ((uint8\_t \*) &(LIST\_ELEM)->next \

- offsetof (STRUCT, MEMBER.next)))

- list\_elem의 주소를 이용해 list\_elem을 담고 있는 structure의 주소를 반환해준다.

void list\_init (struct list \*);

* 리스트를 넘겨받아 리스트의 연결정보를 초기화해준다.

/\* List traversal. \*/

struct list\_elem \*list\_begin (struct list \*);

* 리스트를 넘겨받아 리스트의 맨 앞 노드를 반환한다.

struct list\_elem \*list\_next (struct list\_elem \*);

* list\_elem을 넘겨받고 그 elem의 다음 노드를 반환한다.

struct list\_elem \*list\_end (struct list \*);

* 리스트를 넘겨받아 리스트의 tail을 반환한다. 리스트를 iterating 할 때 iteration종료 조건으로 사용한다.

struct list\_elem \*list\_rbegin (struct list \*);

* 리스트를 넘겨받아 리스트의 마지막 노드를 반환한다. 리스트를 뒤에서부터 iteration할 때 사용한다.

struct list\_elem \*list\_prev (struct list\_elem \*);

* list\_elem을 넘겨받고 그 elem의 이전 노드를 반환한다.

struct list\_elem \*list\_rend (struct list \*);

* 리스트를 넘겨받아 리스트의 head를 반환한다. 리스트를 역으로 iteration할 때 iteration 종료 조건으로 사용한다.

struct list\_elem \*list\_head (struct list \*);

* 리스트를 넘겨받아 리스트의 head노드를 반환한다.

struct list\_elem \*list\_tail (struct list \*);

* 리스트를 넘겨받아 리스트의 tail노드를 반환한다.

/\* List insertion. \*/

void list\_insert (struct list\_elem \*, struct list\_elem \*);

* list\_elem 2개를 인자로 받고 첫번째 elem 앞에다가 두번째 elem을 삽입한다.

void list\_splice (struct list\_elem \*before,

struct list\_elem \*first, struct list\_elem \*last);

* list\_elem 3개를 인자로 받고 첫번째 elem 앞에다가 first 부터 last 전까지의 노드를 삽입한다.

void list\_push\_front (struct list \*, struct list\_elem \*);

* 리스트와 list\_elem을 넘겨받아 리스트 맨 앞에 노드를 추가한다.

void list\_push\_back (struct list \*, struct list\_elem \*);

* 리스트와 list\_elem을 넘겨받아 리스트 맨 뒤에 노드를 추가한다.

/\* List removal. \*/

struct list\_elem \*list\_remove (struct list\_elem \*);

* list\_elem을 넘겨받아 해당 노드를 리스트에서 삭제하고 다음 노드를 반환한다.

struct list\_elem \*list\_pop\_front (struct list \*);

* 리스트를 넘겨받아 맨 앞 노드를 리스트에서 제거하고 제거한 노드를 반환한다.

struct list\_elem \*list\_pop\_back (struct list \*);

* 리스트를 넘겨받아 맨 뒤 노드를 리스트에서 제거하고 제거한 노드를 반환한다.

/\* List elements. \*/

struct list\_elem \*list\_front (struct list \*);

* 리스트를 넘겨받아 맨 앞 노드를 반환한다. 리스트가 비어 있을 때는 정의되지 않는다.

struct list\_elem \*list\_back (struct list \*);

* 리스트를 넘겨받아 맨 뒤 노르르 반환한다. 리스트가 비어 있을 때는 정의되지 않는다.

/\* List properties. \*/

size\_t list\_size (struct list \*);

* 리스트를 넘겨받아 해당 리스트의 노드 개수를 반환한다.

bool list\_empty (struct list \*);

* 리스트를 넘겨받아 해당 리스트가 비어있는지 여부를 true, false로 반환한다.

/\* Miscellaneous. \*/

void list\_reverse (struct list \*);

* 리스트를 넘겨받아 순서를 역으로 뒤집어 저장한다.

typedef bool list\_less\_func (const struct list\_elem \*a,

const struct list\_elem \*b,

void \*aux);

* list\_elem 2개와 보조 값을 인자로 넘겨받아 a, b의 값을 비교해 true, false를 반환한다. list\_elem 타입의 compare 함수이다.

/\* Operations on lists with ordered elements. \*/

void list\_sort (struct list \*,

list\_less\_func \*, void \*aux);

* 리스트와 compare함수, 보조 값을 넘겨받아 해당 리스트를 compare함수 기준으로 정렬해준다.

void list\_insert\_ordered (struct list \*, struct list\_elem \*,

list\_less\_func \*, void \*aux);

* 리스트와 list\_elem, compare 함수, 보조 값을 넘겨받아 해당 리스트에 elem을 추가할 때 compare함수 기준에 맞는 위치에 삽입해준다. 즉, compare 함수 기준으로 정렬된 형태가 되게끔 노드를 삽입한다.

void list\_unique (struct list \*, struct list \*duplicates,

list\_less\_func \*, void \*aux);

* 리스트 2개와 compare함수, 보조 값을 넘겨받아 중복되는 원소를 compare함수를 이용해 찾아 첫번째 리스트에서 제거하고 두번째 리스트 맨 뒤에 삽입한다.

/\* Max and min. \*/

struct list\_elem \*list\_max (struct list \*, list\_less\_func \*, void \*aux);

* 리스트와 compare 함수, 보조 값을 넘겨받아 리스트에서 가장 큰 값을 가진 노드를 반환한다.

struct list\_elem \*list\_min (struct list \*, list\_less\_func \*, void \*aux);

* 리스트와 compare 함수, 보조 값을 넘겨받아 리스트에서 가장 작은 값을 가진 노드를 반환한다.

bool list\_func(const struct list\_elem \*a, const struct list\_elem \*b, void\* aux);

* list\_elem 2개와 보조 값을 받아 a의 데이터가 더 작으면 true, 아니면 false를 반환한다.

/\* 추가 구현한 함수 \*/

void list\_swap( struct list\_elem \*a, struct list\_elem \*b);

* list\_elem 2개를 넘겨받아 두 노드를 서로 바꿔준다. data값만 바꿔주는 방법으로 구현했다.

void list\_shuffle( struct list \*list);

* 리스트를 넘겨받아 리스트의 노드들을 무작위로 섞어준다.

**hash 라이브러리 및 추가 기능 함수**

#define hash\_entry(HASH\_ELEM, STRUCT, MEMBER) \

((STRUCT \*) ((uint8\_t \*) &(HASH\_ELEM)->list\_elem \

- offsetof (STRUCT, MEMBER.list\_elem)))

- hash\_elem의 주소 값을 이용해 hash\_elem을 담고 있는 structure의 주소 값을 반환해준다.

unsigned my\_hash\_hash(const struct hash\_elem \*e, void \*aux);

* 해싱 함수이다. 해싱은 기존 라이브러리인 hash\_int를 사용했다.

bool my\_hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux);

* hash\_elem 2개와 보조 값을 넘겨받는 hash\_elem compare 함수이다. a의 데이터 값이 더 작으면 true, 아니면 false를 반환한다.

void my\_hash\_delete\_func(struct hash\_elem \*e, void \*aux);

* hash\_elem과 보조 값을 넘겨받아 해당 노드의 메모리를 해제해준다.

void my\_hash\_square\_func(struct hash\_elem \*e, void \*aux);

* hash\_elem과 보조 값을 넘겨받아 해당 노드의 값을 제곱해준다.

void my\_hash\_triple\_func(struct hash\_elem \*e, void \*aux);

* hash\_elem과 보조 값을 넘겨받아 해당 노드의 값을 세 제곱해준다.

/\* Hash table. \*/

struct hash

{

size\_t elem\_cnt;

size\_t bucket\_cnt;

struct list \*buckets;

hash\_hash\_func \*hash;

hash\_less\_func \*less;

void \*aux;

};

* hash 노드의 개수, 버킷의 개수, 해싱 함수 등을 저장하는 해시 구조체이다.

/\* A hash table iterator. \*/

struct hash\_iterator

{

struct hash \*hash;

struct list \*bucket;

struct hash\_elem \*elem;

};

* 해시 자료구조를 순회할 때 사용하는 iterator이다.

/\* Basic life cycle. \*/

bool hash\_init (struct hash \*, hash\_hash\_func \*, hash\_less\_func \*, void \*aux);

* 해시테이블과 해시함수, compare함수, 보조 값을 넘겨받아 해시테이블을 초기화 해준다.

void hash\_clear (struct hash \*, hash\_action\_func \*);

* 해시테이블과 해시 destructor 액션 함수를 넘겨받아 해시테이블의 내용물을 모두 제거한다.

void hash\_destroy (struct hash \*, hash\_action\_func \*);

* 해시테이블과 해시 destructor 액션 함수를 넘겨받아 해시테이블을 삭제한다.

/\* Search, insertion, deletion. \*/

struct hash\_elem \*hash\_insert (struct hash \*, struct hash\_elem \*);

* 해시테이블과 hash\_elem을 넘겨받아 해시테이블에 해당 elem을 삽입한다. 만약 중복된 elem이 있으면 삽입하지 않고 기존 elem을 반환한다.

struct hash\_elem \*hash\_replace (struct hash \*, struct hash\_elem \*);

* 해시테이블과 hash\_elem을 넘겨받아 해시테이블에 해당 elem을 삽입한다. 이 때 중복된 elem이 있다면 기존 elem은 해시테이블에서 삭제 후 반환하고, 삽입이 진행된다.

struct hash\_elem \*hash\_find (struct hash \*, struct hash\_elem \*);

* 해시테이블과 hash\_elem을 넘겨받아 해시테이블에서 해당 elem을 찾아 반환한다. 없으면 NULL을 반환한다.

struct hash\_elem \*hash\_delete (struct hash \*, struct hash\_elem \*);

* 해시테이블과 hash\_elem을 넘겨받아 해시테이블에서 해당 elem을 삭제하고 반환한다. 없으면 NULL을 반환한다.

/\* Iteration. \*/

void hash\_apply (struct hash \*, hash\_action\_func \*);

* 해시테이블과 해시액션 함수를 넘겨받아 만약 액션이 square면 값을 제곱해주고 triple이면 세 제곱해준다.

void hash\_first (struct hash\_iterator \*, struct hash \*);

* 해시 이터레이터와 해시테이블을 받아 이터레이터가 첫번째 노드를 가르키게 한다.

struct hash\_elem \*hash\_next (struct hash\_iterator \*);

* 해시 이터레이터를 넘겨받아 이터레이터의 다음 노드를 반환한다.

struct hash\_elem \*hash\_cur (struct hash\_iterator \*);

* 해시 이터레이터를 넘겨받아 이터레이터가 가르키는 elem을 반환한다.

/\* Information. \*/

size\_t hash\_size (struct hash \*);

* 해시테이블을 넘겨받아 해시테이블의 노드 개수를 반환한다.

bool hash\_empty (struct hash \*);

* 해시테이블을 넘겨받아 해시테이블이 비어있으면 true, 아니면 false를 반환한다.

/\* Sample hash functions. \*/

unsigned hash\_bytes (const void \*, size\_t);

* 해시함수로, Fowler-Noll-vo 32bit 해싱을 사용한다.

unsigned hash\_string (const char \*);

* 해시함수로, 스트링에 대해 문자값을 이용해 해싱한다.

unsigned hash\_int (int);

* 해시함수로, hash\_bytes와 동일하다.

/\* 추가 구현한 함수 \*/

unsigned hash\_int\_2(int i);

* int형 데이터를 해싱해주는 함수로, 해싱 성능이 검증 되어있는 Thomas Wang의 integer해싱을 사용했다.

**bitmap 라이브러리 및 추가 기능 함수**

/\* Creation and destruction. \*/

struct bitmap \*bitmap\_create (size\_t bit\_cnt);

* 비트맵 사이즈를 넘겨받아 해당 크기의 비트맵을 만들어 비트맵 포인터를 반환한다.

struct bitmap \*bitmap\_create\_in\_buf (size\_t bit\_cnt, void \*, size\_t byte\_cnt);

* 이미 메모리가 할당되있는 블록을 이용해 비트맵을 만들고 비트맵 포인터를 반환한다.

size\_t bitmap\_buf\_size (size\_t bit\_cnt);

* 비트맵 구조체의 크기와 bit\_cnt를 합해 바이트 단위로 반환한다.

void bitmap\_destroy (struct bitmap \*);

* 비트맵을 넘겨받아 해당 비트맵의 메모리를 해제한다.

/\* Bitmap size. \*/

size\_t bitmap\_size (const struct bitmap \*);

* 비트맵을 넘겨받아 비트맵의 사이즈를 반환한다.

/\* Setting and testing single bits. \*/

void bitmap\_set (struct bitmap \*, size\_t idx, bool);

* 비트맵의 idx에 해당하는 비트를 bool값으로 설정해준다.

void bitmap\_mark (struct bitmap \*, size\_t idx);

* 비트맵과 idx를 넘겨받아 해당 비트를 true로 만든다.

void bitmap\_reset (struct bitmap \*, size\_t idx);

* 비트맵과 idx를 넘겨받아 해당 비트를 false로 만든다.

void bitmap\_flip (struct bitmap \*, size\_t idx);

* 비트맵과 idx를 넘겨받아 해당 비트를 반전시킨다. 0->1, 1->0.

bool bitmap\_test (const struct bitmap \*, size\_t idx);

* 비트맵과 idx를 넘겨받아 해당 비트 값을 true, false로 반환한다.

/\* Setting and testing multiple bits. \*/

void bitmap\_set\_all (struct bitmap \*, bool);

* 비트맵과 bool값을 넘겨받아 비트맵을 모두 넘겨받은 bool값으로 설정해준다.

void bitmap\_set\_multiple (struct bitmap \*, size\_t start, size\_t cnt, bool);

* 비트맵과 시작위치, 개수, bool값을 넘겨받아 start부터 cnt개수만큼 bool값으로 설정해준다.

size\_t bitmap\_count (const struct bitmap \*, size\_t start, size\_t cnt, bool);

* 비트맵과 시작위치, 개수, bool값을 넘겨받아 bool값에 해당하는 비트가 몇 개인지 세어 반환한다.

bool bitmap\_contains (const struct bitmap \*, size\_t start, size\_t cnt, bool);

* 비트맵과 시작위치, 개수, bool값을 넘겨받아 해당 범위에 bool값에 해당하는 비트가 있으면 true, 없으면 false를 반환한다.

bool bitmap\_any (const struct bitmap \*, size\_t start, size\_t cnt);

* 비트맵과 시작위치, 개수를 넘겨받아 해당 범위에 true인 비트가 있으면 true, 아니면 false를 반환한다.

bool bitmap\_none (const struct bitmap \*, size\_t start, size\_t cnt);

* 비트맵과 시작위치, 개수를 넘겨받아 해당 범위에 true인 비트가 없으면 true, 아니면 false를 반환한다.

bool bitmap\_all (const struct bitmap \*, size\_t start, size\_t cnt);

* 비트맵과 시작위치, 개수를 넘겨받아 해당 범위가 모두 true인 비트면 true, 아니면 false를 반환한다.

/\* Finding set or unset bits. \*/

size\_t bitmap\_scan (const struct bitmap \*, size\_t start, size\_t cnt, bool);

* 비트맵과 시작위치, 개수, bool값을 넘겨받아 넘겨받은 bool값이 cnt개만큼 연속으로 존재하는 곳의 idx를 반환한다.

size\_t bitmap\_scan\_and\_flip (struct bitmap \*, size\_t start, size\_t cnt, bool);

* 비트맵과 시작위치, 개수, bool값을 넘겨받아 넘겨받은 bool값이 cnt개만큼 연속으로 존재하는 곳을 비트를 반전시킨 후, idx를 반환한다

/\* Debugging. \*/

void bitmap\_dump (const struct bitmap \*);

* 비트맵을 넘겨받아 16진수로 출력해주고 아스키문자로 출력해준다.

/\* 추가 구현한 함수 \*/

struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, int size);

* 비트맵과 사이즈를 넘겨받아 비트맵의 크기를 뒤로 사이즈만큼 늘려준다. 늘려준 비트맵을 반환한다.