**OS project #0-2**

Documentation for role of library functions used in assignment.

*Author : 20141500권태국*

**Section 1. list.c , list.h**

**STRUCT\* list\_entry(LIST\_ELEM, STRUCT, MEMBER);**

list\_elem를 가리키는 포인터를 이용해 list\_elem 바깥의 (실제 data를 저장하고

있는) 구조체의 포인터를 얻어낸다. 매크로함수임.

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

list를 초기화한다. 내부적으로 head, tail에 대한 초기화가 이루어진다.

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

list에 대해 정 방향 iteration을 시작한다. List의 맨 처음 element를 가리키는 pointer를 반환한다. (맨 처음 element = head.next)

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

정 방향 iteration of list 상에서 현재 element의 다음 element를 얻을 수 있다.

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

정 방향 iteration of list에서 iteration의 끝 지점을 반환한다. (list내의 tail를 가리키는 pointer를 반환한다.)

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

list에 대해 tail부터 역방향 iteration을 시작한다. List의 맨 마지막 element를 가리키는 pointer를 반환한다. (맨 마지막 element = tail.prev)

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

역 방향 iteration of list 상에서 현재 element의 다음 element를 얻을 수 있다.

(현재 element의 prev)

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

역 방향 iteration of list에서 iteration의 끝 지점을 반환한다. (list내의 head를 가리키는 pointer를 반환한다.)

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

list내의 head를 가리키는 pointer를 반환한다.

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

list내의 tail를 가리키는 pointer를 반환한다.

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

첫 번째 인자로 주어진 element 앞에 두 번째 인자로 주어진 element를 삽입한

다. (list\_push\_back()과 동일)

**void list\_splice (struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last);**

[first,last]를 포함하고 있는 list내에서 [first,last]를 제거하고, [first,last]를 before앞에 붙인다.

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

list의 앞에 element를 추가한다.

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

list의 뒤에 element를 추가한다.

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

list내에서 element를 제거하고, 제거된 element의 다음 element를 반환한다.

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

list 맨 앞의 element를 제거하고, 제거된 element를 반환한다.

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

list 맨 뒤의 element를 제거하고, 제거된 element를 반환한다.

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

list내에 맨 앞에 있는 element를 반환한다.

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

list내에 맨 뒤에 있는 element를 반환한다.

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

list의 size(=the number of elements)를 반환한다. 시간복잡도는 O(N).

**bool list\_empty (struct list \*);**

list가 비었으면 true, 아니면 false를 반환한다.

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

list를 거꾸로 뒤집는다.

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

list를 정렬한다.

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

정렬된 순서를 유지하며 list에 element를 삽입한다.

**void list\_unique (struct list \*, struct list \*duplicates, list\_less\_func \*, void \*aux);**

list에서 중복되는 element들을 1개만 남기고 삭제한다. duplicates가 NULL이 아닌 경우 duplicates에 중복되는 element들을 저장한다.

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

list내에서 등호관계상 가장 큰 element를 찾아서 반환한다.

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

list내에서 등호관계상 가장 작은 element를 찾아서 반환한다.

**struct list\_elem \*list\_find\_nth (struct list \*, size\_t);**

list내에서 n-th element를 반환한다.

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

list내에서 2개의 element를 서로 swap한다.

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

list를 무작위로 섞는다.

**Section 2. hash.c , hash.h**

**STRUCT\* hash\_entry(HASH\_ELEM, STRUCT, MEMBER);**

hash\_elem를 가리키는 포인터를 이용해 hash\_elem 바깥의 (실제 data를 저장하고 있는) 구조체의 포인터를 얻어낸다. 매크로함수임.

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

hash table을 초기화한다.

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

hash table내의 모든 element들을 삭제하고 bucket들을 초기화한다. Destructor가 NULL이 아니면 삭제되는 모든 hash element에 대해 destructor를 호출하여준다.

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

hash table을 파괴한다. 내부적으로 hash\_clear()를 호출하고, buckets를 free한다.

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

hash table에 element를 삽입한다. 이미 동일한 element가 존재할 경우, 삽입되지 않는다. 기존에 존재하는 동일한 element를 반환한다. (없으면 NULL)

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

hash table에 element를 삽입한다. 이미 동일한 element가 존재할 경우, 기존의 element를 삭제하고 새로 삽입한다 (hash\_insert와의 차이점). 기존에 존재하는 동일한 element를 반환한다. (없으면 NULL)

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

hash table에서 특정 element을 검색한다.

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

hash table에서 특정 element를 삭제한다. 삭제된 element를 반환한다.

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

hash table의 모든 element에 대해서 action function을 호출해준다.

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

hash table의 iterator를 초기화한다. (주의 : hash\_first후 hash\_next를 최소 1번 호출해줘야 유의미한 hash element에 접근할 수 있다.)

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

현재 iterate하고 있는 hash table내의 다음 element를 얻는다.

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

현재 iterate하고 있는 hash table내의 현재 element를 얻는다.

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

hash table의 size(=the number of elements)를 얻는다.

**bool hash\_empty (struct hash \*);**

hash table이 비었으면 true, 그렇지 않으면 false를 반환한다.

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

buffer내의 특정 size bytes만큼의 hash value를 반환한다.

**unsigned hash\_string (const char \*);**

문자열의 hash value를 반환한다.

**unsigned hash\_int (int);**

integer의 hash value를 반환한다. 내부적으로 hash\_bytes(&i, sizeof i); 을 호출한다.

**unsigned hash\_int\_2 (int);**

integer의 hash value를 반환한다. Robert Jenkins’ 32 bit integer hash function을 사용한다.

**Section 3. bitmap.c , bitmap.h**

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

bit\_cnt만큼의 bit들을 가지는 bitmap를 생성한다. 초기에 bitmap내부의 bit들은 전부 0값을 가지고 있다.

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

bitmap을 생성하되 필요한 메모리를 동적 할당하지 않고, 주어진 buffer block을 이용하여 생성한다.

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

bit\_cnt만큼의 bit들을 가지는 bitmap을 저장하기 위한 buffer의 size를 반환한다.

**void bitmap\_destroy (struct bitmap \*);**

bitmap을 파괴한다. 동적 할당 해제도 함수내부에서 이루어진다.

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

bitmap의 size(=the number of bits)를 반환한다.

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

bitmap에서 idx번째 bit의 값을 특정 value로 세팅한다.

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

bitmap에서 idx번째 bit의 값을 1로 세팅한다.

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

bitmap에서 idx번째 bit의 값을 0으로 세팅한다.

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

bitmap에서 idx번째 bit의 값을 toggle한다. (0->1 , 1->0)

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

bitmap에서 idx번째 bit의 값을 반환한다.

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

bitmap내부의 모든 bit들을 특정 value로 세팅한다.

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

bitmap에서 start번째부터 cnt만큼의 bit들을 특정 value로 세팅한다.

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

bitmap에서 start번째부터 cnt만큼의 bit들중에 특정 value를 가진 bit들의 개수를 반환한다.

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

bitmap의 [start,start+cnt) 범위에서 특정 value를 가진 bit를 1개라도 포함하면 true, 그렇지 않으면 false를 반환한다.

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

bitmap의 [start,start+cnt) 범위에서 1의 값을 가진 bit를 1개라도 포함하면 true, 그렇지 않으면 false를 반환한다.

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

bitmap의 [start,start+cnt) 범위의 모든 bit들의 값이 0이면 true, 그렇지 않으면 false를 반환한다.

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

bitmap의 [start,start+cnt) 범위의 모든 bit들의 값이 1이면 true, 그렇지 않으면 false를 반환한다.

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

bitmap의 [start,start+cnt) 그룹부터 [last,last+cnt) 그룹까지의 그룹에 대해 그룹내의 모든 bit가 특정 value인 그룹이 1개라도 있으면 그 그룹의 시작 bit index를 반환하고, 그렇지 않으면 BITMAP\_ERROR를 반환한다. (last = bitmap’s bit cnt – cnt)

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

그룹들에 대해 scan을 하여, 모든 bit가 특정 value인 그룹이 존재하면 그 그룹내의 bit들을 toggle하고, 그룹의 시작 bit index를 반환한다. 그런 그룹이 존재하지 않으면 BITMAP\_ERROR를 반환한다.

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

bitmap을 덤프한다. 디버깅용 함수.

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

bitmap을 주어진 크기만큼 추가로 확장한다. 기존 bitmap내부의 bit들의 값은 유지한 채로 뒤에 주어진 크기만큼 확장하여 확장된 bit들에 대해 0으로 값을 세팅한다.