과목 명: 운영체제

담당 교수 명: 이혁준

분반 : 2반

<<Project 0\_2>>

**서강대학교 컴퓨터공학과**

**20141578**

**정민교**

목 차

1. Doubly Linked List 3

1.1 초기화 function 3

1.2 Doubly Linked List iterator 3

1.3 Doubly Linked List Properties 3

1.4 Doubly Linked List Removal 3

1.5 Doubly Linked List Insertion 4

1.6 Doubly Linked List Util 4

2. Hash Table 5

2.1 Life Cycle function 5

2.2 Hash Table Iterator 5

2.3 Hash Table Properties 5

2.4 Hash Table Utils 5

2.5 Hash Function 6

3. Bitmap 6

3.1 Bitmap Creation & Destruction 6

3.2 Bitmap Size 6

3.3 Bitmap Setting Utils 6

3.4 Bitmap Testing Utils 7

3.5 Bitmap Finding Utils 7

3.6 E.T.C 7

# Doubly Linked List

## 초기화 function

1. Void list\_init(struct list \*)

Doubly Linked List를 초기화한다. Head의 next를 tail로, tail의 prev를 head로, head의 prev와 tail의 next를 각각 NULL로 설정해준다.

## Doubly Linked List iterator

1. Struct list\_elem\* list\_begin(struct list\*)

Doubly Linked List의 Head의 next list\_elem\*을 리턴한다. 즉, Head와 Tail을 제외한 첫 번째 element를 리턴한다.

1. Struct list\_elem\* list\_end(struct list\_elem\*)

Doubly Linked List의 Tail을 리턴한다.

1. Struct list\_elem\* list\_next(struct list\_elem\*)

현재 list\_elem\*이 가리키고 있는 다음 list\_elem\*을 리턴한다.

## Doubly Linked List Properties

1. Bool list\_empty (struct list\*)

Doubly Linked List가 비었는지 확인한다. Doubly Linked List가 비었다면 true를, 비어있지 않다면 false를 리턴한다.

1. Size\_t list\_size (struct list\*)

Doubly Linked List의 Head와 Tail을 제외한 실제 데이터를 갖는 Element의 수를 리턴한다.

1. Struct list\_elem\* list\_max(struct list\*, list\_less\_func \*, void\* aux)

Doubly Linked List의 max값을 저장하는 structure의 list\_elem\*를 less function 설정에 따라 리턴한다.

1. Struct list\_elem\* list\_min(struct list\*, list\_less\_func\*, void\* aux)

Doubly Linked List의 min값을 저장하는 structure의 list\_elem\*를 less function 설정에 따라 리턴한다.

1. Struct list\_elem\* list\_front(struct list\*)

Doubly Linked List의 맨 앞 element를 리턴한다.(Head의 next)

1. Struct list\_elem\* list\_back(struct list\*)

Doubly Linked List의 맨 뒤 element를 리턴한다.(Tail의 prev)

## Doubly Linked List Removal

1. Struct list\_elem\* list\_pop\_front(struct list\*)

Doubly Linked List의 맨 앞 element(Head의 next)를 뽑아내고 다른 Element와의 (경우에 따라 Head 혹은 Tail과의) 연결을 끊는다. 그리고 그 element를 반환한다.

1. Struct list\_elem\* list\_pop\_back(struct list\*)

Doubly Linked의 맨 뒤 element(Tail의 prev)를 뽑아내고 다른 Element와의 (경우에 따라 Head 혹은 Tail과의) 연결을 끊는다. 그리고 그 element를 반환한다.

1. Struct list\_elem\* list\_remove(struct list\_elem\*)

인자로 받은 list\_elem을 자신이 속해있는 Linked list로부터 연결을 끊고 그 list\_elem\*의 next list\_elem\*을 리턴한다.

## Doubly Linked List Insertion

1. Void list\_push\_back(struct list\*, struct list\_elem\*)

Doubly Linked List의 맨 뒤에(Tail의 prev) Element를 삽입한다. 이때, 삽입하는 Element 내부의 list\_elem\*는 두 번째 인자이다

.

1. Void list\_push\_front(struct list\*, struct list\_elem\*)

Doubly Linked List의 맨 앞에(Head의 next) Element를 삽입한다. 이때, 삽입하는 Element 내부의 list\_elem\*는 두 번째 인자이다.

1. Void list\_insert(struct list\_elem\*, struct list\_elem\*)

첫 번째 인자의 뒤에 두 번째 인자를 연결한다. 첫 번째 인자의 next에 두 번째 인자를 연결하고, 두 번째 인자의 prev에 첫 번째 인자를 연결한다.

1. Void list\_insert\_ordered(struct list\*, struct list\_elem\*, list\_less\_func\*, void\* aux)

Doubly Linked List가 정렬되어있다는 가정 하에, 해당 List에 인자로 받은 함수포인터(elem간의 비교 연산)를 사용해 함수 포인터가 하는 기능에 따라 Element를 삽입한다. (ex. 함수포인터가 내림차순인 경우 내림차순으로 element가 insert 된다.)

## Doubly Linked List Util

1. Void list\_unique(struct list\*, struct list\* duplicates, list\_less\_func \*, void\* aux)

Doubly Linked List가 정렬되어있다는 가정 하에, 해당 list에 인자로 받은 함수포인터(elem간의 비교 연산)를 이용해 element의 data가 같은 경우 두 번째 인자로 받은 list에 저장한다. 만약 두 번째 인자로 받은 struct list\*가 NULL이라면 그 Element는 버린다.

1. Void list\_splice(struct list\_elem\* before, struct list\_elem\* first, struct list\_elem\* last)

First와 last사이의 Element들을 before 뒤에 붙인다. 물론 first와 last 사이의 Element들은 원래 list로부터 분리된다.

1. Void list\_swap(struct list\_elem\*, struct list\_elem\*)

(새롭게 구현한 Library function)첫 번째 인자와 두 번째 인자를 서로 바꾼다. 이때, 각 인자의 이전 list\_elem\*과 다음 list\_elem\*이 다른 인자를 가리키고, 각 인자의 next pointer와 prev pointer를 swap한다.

1. Void list\_shuffle(struct list \*list)

(새롭게 구현한 Library function)실제로 사용하지는 않는 library function으로 Fisher-Yates Shuffle을 사용한다.

1. Void list\_reverse(struct list\*)

Doubly Linked List의 element의 순서를 반대로 바꾼다.

# Hash Table

## Life Cycle function

1. Bool hash\_init(struct hash\*, hash\_hash\_func \*, hash\_less\_func\*, void\* aux)

Hash table을 초기화한다. 각 인자는 Hash table의 구성요소가 된다. 두 번째 요소는 Hash Table의 Hash Function 을, 세 번째 요소는 Hash table의 element 비교 Function이다. 이 함수들은 Hash Table에 저장되어 Hash Table에 element를 insert할 때 이 function을 이용해 bucket과 element 위치를 결정한다.

1. Void hash\_destroy(struct hash\*, hash\_action\_func \*)

Hash table을 삭제하는 함수로 hash\_action\_func의 함수포인터 자리에는 destructor가 들어간다. Hash Table내의 Bucket과 Bucket을 이루고 있는 Doubly Linked List의 모든 element를 free시킨다. 단, element를 free시키는 것은 인자로 받은 destructor 가 한다.

1. Void hash\_clear(struct hash\*, hash\_action\_func \*)

hash\_destroy와는 다르게 bucket은 지우지 않고 bucket내의 element만 hash action function(destructor)로 삭제(memory free)하고 bucket을 초기화 시킨다.

## Hash Table Iterator

1. Void hash\_first(struct hash\_iterator\*, struct hash\*)

Hash\_iterator\* i에 Hash table과 Hash table의 bucket, bucket의 head를 넘긴다.

1. Struct hash\_elem\* hash\_next(struct hash\_iterator\*)

Hash table의 모든 bucket과 bucket의 element를 탐색할 수 있게 하는 함수이다.

## Hash Table Properties

1. Bool hash\_empty(struct hash\*)

Hash table로부터 elem\_cnt 값이 0인지 아닌지 리턴한다. 이는 Hash table의 모든 bucket 중 element가 하나라도 있는지를 알려준다.

1. Size\_t hash\_size(struct hash\*)

Hash table로부터 elem\_cnt값을 리턴한다. 이는 Hash table의 모든 bucket 들의 element 개수들의 합을 의미한다.

## Hash Table Utils

1. Struct hash\_elem\* hash\_insert(struct hash\*, struct hash\_elem\*)

인자로 받은 hash\_elem\*을 Hash table에 추가시키는데, Hash table 생성 시 받은 hash function과 less function을 이용해 element가 어느 bucket에 들어갈지(hash function), 해당 bucket의 몇 번째 element로 들어갈지(less function)를 결정해 넣어준다.

1. Void hash\_apply(struct hash\*, hash\_action\_func\*)

Iterator를 이용해 Hash Table의 모든 element에 대해 인자로 받은 hash action function을 적용시킨다.

1. Struct hash\_elem\* hash\_find(struct hash\*h, struct hash\_elem \*e)

Hash table 내에 두 번째 인자로 받은 것의 data(hash\_elem을 저장하고 있는 element의 data)가 존재하는지 찾는 함수로 없으면 NULL을 반환하고, 있다면 그 element를 반환한다.

1. Struct hash\_elem\* hash\_replace(struct hash\*, struct hash\_elem\*)

두 번째 인자를 Hash table에 insert하는데, 이때 Hash Table내에 같은 값을 가진 element가 있다면 Hash Table로부터 삭제하고 그 element를 리턴한다. 또한, insert 후 rehashing을 한다.

1. Struct hash\_elem\* hash\_delete(struct hash\*, struct hash\_elem\*)

두 번째 인자를 저장하고 있는 element의 data가 Hash Table에 존재한다면 그 element를 Hash table로부터 삭제하고 리턴한다. 그리고 Hash Table을 rehashing 한다.

## Hash Function

1. Unsigned hash\_int(int)

Integer 형을 위한 library에서 제공하는 hash function으로 함수 내부에서 hash\_bytes를 Call해 그 값을 리턴한다. Hash\_bytes는 Fowler-Noll-Vo Hash function을 사용한다.

1. Unsigned hash\_int\_2(int)

(새롭게 구현한 Library function) gist.github.com/badboy/6267743oid 에서 가져온 integer hash function이다.

# Bitmap

## Bitmap Creation & Destruction

1. Struct bitmap\* bitmap\_create(size\_t bit\_cnt)

Bitmap 구조체를 메모리로부터 할당해 Bitmap 구조체 내의 bits멤버를 bit\_cnt사이즈 만큼 메모리로부터 할당해준다. 그리고 할당받은 Bitmap 구조체를 리턴해준다.

1. Void bitmap\_destroy(struct bitmap\*)

Bitmap 구조체의 bits멤버를 할당해제(memory free)해주고, bitmap 구조체 역시 할당해제 해준다.

## Bitmap Size

1. Size\_t bitmap\_size(const struct bitmap\*)

Bitmap 구조체의 bits멤버의 Size를 리턴한다.

1. Bitmap\_expand(struct bitmap\*, int size)

(새로 구현한 함수)Bitmap 구조체의 bits멤버의 Size를 Resizing 해준다(보통 늘려준다.) Resizing은 bits를 reallocate함으로써 이루어진다.

## Bitmap Setting Utils

1. Void bitmap\_mark(struct bitmap\*, size\_t idx)

Bitmap의 해당 index bit를 True(1)로 만들어준다.

1. Void bitmap\_flip(struct bitmap\*, size\_t idx)

Bitmap의 해당 index bit를 Toggle(true🡪false, false🡪true)해준다.

1. Void bitmap\_reset(struct bitmap\*, size\_t idx)

Bitmap의 해당 index bit를 false로 바꾼다.

1. Void bitmap\_set(struct bitmap\*, size\_t idx, bool)

Bitmap의 해당 index bit를 4번째 인자의 값으로 바꾼다.

1. Void bitmap\_set\_multiple(struct bitmap\*, size\_t start, size\_t cnt, bool)

Bitmap의 start index bit부터 cnt개의 bit만큼을 4번째 인자의 값으로 바꾼다.

1. Void bitmap\_set\_all(struct bitmap\*, bool)

Bitmap의 모든 bit를 두 번째 인자 값으로 바꾼다.

## Bitmap Testing Utils

1. Bool bitmap\_test(const struct bitmap\*, size\_t idx)

해당 index의 bit가 1이면 true, 0이면 false를 리턴한다.

1. Bool bitmap\_contains(const struct bitmap\*, size\_t start, size\_t cnt, bool)

Start index부터 cnt개 만큼의 bit 중 하나가 4번째 인자가 가리키는 값(true/false)를 하나라도 가지고 있으면 true, 그렇지 않으면 false를 리턴한다.

1. Bool bitmap\_any(const struct bitmap\*, size\_t start, size\_t cnt)

Start index부터 cnt개 만큼의 bit중 하나가 true로 set 되어있으면 true를, 그렇지 않으면 false를 리턴한다.

1. Bool bitmap\_all(const struct bitmap\*, size\_t start, size\_t cnt)

Start index부터 cnt개 만큼의 bit가 모두 true로 set 되어있으면 true를, 그렇지 않으면 false를 리턴한다.

1. Bool bitmap\_none(const struct bitmap\*, size\_t start, size\_t cnt)

Start index 부터 cnt개 만큼의 bit가 모두 false로 set 되어있으면 true를, 그렇지 않으면 false를 리턴한다.

## Bitmap Finding Utils

1. Size\_t bitmap\_count(const struct bitmap\*, size\_t start, size\_t cnt, bool)

Start index부터 cnt개 만큼의 bit 중 4번째 인자가 가리키는 값(true/false)를 갖고 있는 bit들의 개수를 리턴한다.

1. Size\_t bitmap\_scan(const struct bitmap\*, size\_t start, size\_t cnt, bool)

Start index부터 시작해서 i번째 index에서 연속으로 cnt개 만큼의 bit 모두에 대해 4번째 인자가 가리키는 값(true/false)로 되어있다면 그 시작 index를 리턴한다.. 만약 cnt가 bitmap의 size보다 크거나 그런 그룹이 없다면 BITMAP\_ERROR (SIZE\_MAX)를 리턴한다.

1. Size\_t bitmap\_scan\_and\_flip(struct bitmap\*, size\_t start, size\_t cnt, bool)

Start index부터 시작해서 i번째 index에서 연속으로 cnt개 만큼의 bit 모두에 대해 4번째 인자가 가리키는 값(true/false)로 되어있다면 그 시작 index부터 cnt개 만큼의 bit를 반대로 뒤집는다(Toggle). 만약 cnt가 bitmap의 size보다 크거나 그런 그룹이 없다면 BITMAP\_ERROR (SIZE\_MAX)를 리턴한다.

## E.T.C

1. Void bitmap\_dump(const struct bitmap\*)

Bitmap의 Hexa dump값을 출력한다.