**Pintos Project 0-2: Pintos Data Structure**

|  |  |
| --- | --- |
| 담당 교수 : | 문의현 교수님 |
| 학번 : | 20201116 |
| 이름 : | 이수빈 |
|  |  |

**반드시 아래의 양식과 순서를 따라서 작성하기 바랍니다.**

1. **Additional Implementation**

|  |  |
| --- | --- |
| **Prototype** | void list\_swap(struct list\_elem \*a, struct list\_elem \*b) |
| **Parameter** | 바꾸고자 하는 원소 a, b |
| **Return** | X |
| **Function** | 리스트의 두 원소를 바꾸어 주는 반환 값이 없는 함수이다. 바꾸고자 하는 리스트의 두 원소 a, b를 인자로 받는다. 첫번째 인자 a를 stuct list\_elem 형태의 temp라는 변수를 지정해 복사한다. 그리고 a를 b로 재정의하고 b는 미리 복사해준 a, 즉, temp로 재정의한다 |

|  |  |
| --- | --- |
| **Prototype** | void list\_shuffle(struct list \*list) |
| **Parameter** | 셔플하고자 하는 리스트 |
| **Return** | X |
| **Function** | 이는 리스트를 받아 리스트를 셔플해주는 반환 값이 없는 함수이다. 셔플하고자 하는 리스트를 인자로 받는다. 해당 리스트의 사이즈를 list\_len이라는 값에 저장하고 0부터 list\_len-1까지 for문을 통해 바꾸어 주는 작업을 계속한다. 이때, 바꾸어지는 방법은 랜덤하게 바꾸기 위해 srand(time(NULL))을 이용해 랜덤한 값을 잡아주고, 인덱스가 i인 함수와 바꾸어 줄 랜덤한 인덱스, rand()%(i+1)을 잡아준다. 그리고 이를 현재 원소와 바꾼다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int\_2(int i); |
| **Parameter** | 정수 값 |
| **Return** | 정수를 해시 값으로 바꾼 값을 반환 |
| **Function** | 버킷이 4개 있다 가정하여 가장 심플하게 나머지 함수를 이용해 구현하였다. |

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_expand(struct bitmap \*bitmap, size\_t size) |
| **Parameter** | 확장 하고자 하는 비트맵, 확장 사이즈 |
| **Return** | 확장된 비트맵 |
| **Function** | 비트맵의 사이즈를 size로 확장한다. 이때, 새로운 비트맵을 bitmap\_create를 이용하여 만든다. 그리고 원래 사이즈인 부분까지 새로운 비트맵에 똑같이 복사해준다. 이때, 성공하면 새로운 비트 맵을 반환하고 실패하면 NULL 값을 반환한다. |

1. **List**

더블 링크드 리스트를 구현하면 된다. 단, 이때 리스트를 정의할때 앞뒤 리스트를 가리키는 포인터를 list\_elem으로 정의한 후 이를 list\_item이라는 구조체에서 데이터와 함께 원소로 갖게 한다. 단, 함수를 이용할 때 list\_elem을 사용한다.핀토스에서 리스트를 사용하기 위해 구현한 함수들을 살펴보도록 하자.

|  |  |
| --- | --- |
| **Prototype** | void list\_init(struct list \*list) |
| **Parameter** | 초기화하고자 하는 리스튼 |
| **Return** | X |
| **Function** | 리스트를 빈 리스트로 초기화한다. 이는 리스트에 원소가 삽입되기 이전에 일어 나야 한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_begin(struct list \*list) |
| **Parameter** | 리스트 |
| **Return** | 가장 처음 원소 |
| **Function** | 인자로 받은 리스트의 가장 처음 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_next(struct list\_elem \*elem) |
| **Parameter** | 리스트 |
| **Return** | 다음 원소 |
| **Function** | 인자로 받은 리스트의 다음 원소를 반환한다 |

|  |  |
| --- | --- |
| **Prototype** | struct list\_elem\* list\_end(struct list \*list) |
| **Parameter** | 리스트 |
| **Return** | 마지막 원소 |
| **Function** | 인자로 받은 리스트의 마지막 원소를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | void list\_insert(struct list\_elem \*before, struct list\_elem \*elem) |
| **Parameter** | 추가하려는 리스트의 마지막 원소와 추가하고자 하는 원소 |
| **Return** | X |
| **Function** | 기존 리스트의 마지막 원소의 포인터와 추가 하고자 하는 원소의 포인터를 새로 지정하여 추가한다. |

1. **Hash Table**

Key를 해시 함수를 적용해서 value로 변환해 이를 매핑하는 형태이다. 이는 가상 메모리를 이용할 때 page를 frame으로 매핑할 때 사용된다. Hash 구조체는 구성 원소의 개수를 나타내는 elem\_cnt, bucket의 개수를 나타내는 bucket\_cnt, 그리고 그 버킷의 배열을 가리키는 포인터 struct list \*buckets, 해시 함수, 비교 함수로 구성된다. 이제 이 해시 테이블을 구현하는 함수들을 하나하나 살펴보자.

|  |  |
| --- | --- |
| **Prototype** | bool hash\_init ( struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux) |
| **Parameter** | 초기화 하고자 하는 해시 테이블, 해시 함수 |
| **Return** | 성공하면 true 실패시 false |
| **Function** | 해시 테이블을 초기화 해주는 함수이다. 이때, 여기서는 버킷의 개수를 4로 지정한다. 그리고 각 버킷을 가리키는 포인터를 malloc을 이용해 (포인터의 크기)\*(버킷의 개수) 만큼 할당해 준다. |

|  |  |
| --- | --- |
| **Prototype** | void hash\_clear ( struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | 해시 테이블, destructor 함수 |
| **Return** | X |
| **Function** | 해시 테이블의 모든 원소를 제거해준다. 해시 함수의 크기만큼 for문을 돌면서, 각 버킷을 가리키는 포인터로 각 버킷을 접근해, 연결되어 있는 모든 원소들을 제거한다 |

|  |  |
| --- | --- |
| **Prototype** | void hash\_destroy ( struct hash \*h, hash\_action\_func \*destructor) |
| **Parameter** | 해시 테이블, destructor 함수 |
| **Return** | X |
| **Function** | 위 함수와 다르게 해시 함수 자체를 파괴한다. 이 함수 안에서 hash\_clear함수를 호출해 모든 원소를 제거하고 마지막에 추가로 bucket 포인터를 할당 해제 해준다. |

|  |  |
| --- | --- |
| **Prototype** | struct hash\_elem \*hash\_insert( struct hash \*h, struct hash\_elem \*new) |
| **Parameter** | 해시 테이블, 삽입 하고자 하는 원소 |
| **Return** | 삽입 성공하면 NULL, 해당 값이 이미 있으면 해당 원소 반환 |
| **Function** | 해시 함수에 new 새로운 원소를 삽입해준다. find\_elem 함수를 이용해서 해당 원소가 이미 있는지 여부를 확인하고 없다면 find\_bucket을 이용해서 찾은 해당 버킷에 새로운 원소를 넣어준다. 마지막에 rehash 함수를 이용에 필요에 따라 버킷의 개수를 조절해준다 |

|  |  |
| --- | --- |
| **Prototype** | static struct list \*find\_bucket( struct hash \*h, hash\_elem \*e) |
| **Parameter** | 해시 테이블, 찾고자 하는 원소 |
| **Return** | 찾은 버킷 |
| **Function** | hash 함수를 이용해서 해당 원소가 속하는 버킷을 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | static struct hash\_elem \*hash\_find\_elem( struct hash \*h, strcut list \*bucket, struct hash\_elem \*e) |
| **Parameter** | 해시 테이블, 버킷, 찾고자 하는 원소 |
| **Return** | 찾은 원소 |
| **Function** | hash\_find함수에서 호출되는 찾고자 하는 원소를 찾아 반환해주는 함수이다. |

|  |  |
| --- | --- |
| **Prototype** | static void rehash (stuct hash \*h) |
| **Parameter** | rehash 할 해시 테이블 |
| **Return** | X |
| **Function** | BEST\_ELEMS\_PER\_BUCKET이라는 변수에 한 버킷에 들어가면 좋을 원소의 개수가 저장되어 있다. 만약, 총 원소의 수/BEST\_ELEMS\_PER\_BUCKET가 현재 버킷의 수 (4)보다 크다면 새로운 버킷을 만들어 해시 테이블을 조정해준다. 이때, 총 버킷의 개수는 2의 제곱이어야 한다. 새로 버킷을 만들고, 기존 원소들을 find\_bucket함수를 이용해 다시 버킷에 배정해준다. |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_bytes (const void \*buf, size\_t size) |
| **Parameter** |  |
| **Return** |  |
| **Function** |  |

|  |  |
| --- | --- |
| **Prototype** | unsigned hash\_int (int i) |
| **Parameter** |  |
| **Return** |  |
| **Function** | hash\_bytes(&i, size\_of i)를 호출한다. |

1. **Bitmap**

비트맵은 메모리의 할당 여부를 판단하기 쉽게 해준다. 할당이 된 메모리에는 true 값, 비어 있는 메모리에는 false (0) 값을 갖는다. 예를 들어 디스크의 메모리를 관리한다고 생각해보자. 이때, 한 섹터는 512 (2^9) bytes 그리고 한 페이지는 4KB (2^9 \* 8) 메모리를 차지하므로 8개의 섹터가 한 페이지에 저장될 수 있다. 비트맵을 이용하여 해당 페이지를 저장하기 위해서는 연속적으로 false 값을 갖고 있는 (비어 있는) 부분을 비트맵에서 찾아야한다. 한 페이지를 저장하는데 8개의 연속적으로 0을 갖는 부분이 필요하므로 allocation unit은 8이 된다. 이제 이를 구현하기 위한 함수들을 살펴 보자.

|  |  |
| --- | --- |
| **Prototype** | struct bitmap \*bitmap\_create(size\_t bit\_cnt) ­ |
| **Parameter** | 비트맵의 크기 |
| **Return** | 새로 만든 비트맵 |
| **Function** | 비트맵의 사이즈를 인자로 받아 비트맵을 만든다 |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_destroy(struct bitmap \*b) |
| **Parameter** | 파괴할 비트맵 |
| **Return** | X |
| **Function** | 비트맵은 따로 동적으로 할당된 메모리가 없어서 그냥 destroy 함수를 불러 파괴하면 된다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set (struct bitmap \*b, size\_t idx, bool value) |
| **Parameter** | 비트맵, 바꾸고자 하는 부분의 인덱스, 바꾸고자 하는 값 |
| **Return** | X |
| **Function** | 비트맵 b에서 인덱스 idx에 해당하는 부분의 값을 value (true 또는 false)로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_mark(struct bitmap \*b, size\_t idx) |
| **Parameter** | 비트맵, 바꾸고자 하는 부분의 인덱스 |
| **Return** | X |
| **Function** | 비트맵 b에서 인덱스 idx에 해당하는 부분의 값을 true로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_reset(struct bitmap \*b, size\_t idx) |
| **Parameter** | 비트맵, 바꾸고자 하는 부분의 인덱스 |
| **Return** | X |
| **Function** | 비트맵 b에서 인덱스 idx에 해당하는 부분의 값을 false로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_flip(struct bitmap \*b, size\_t idx) |
| **Parameter** | 비트맵, 바꾸고자 하는 부분의 인덱스 |
| **Return** | X |
| **Function** | 비트맵 b에서 인덱스 idx에 해당하는 부분의 값을 반대로 (true -> false, false -> true) 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_test (struct bitmap \*b, size\_t idx) |
| **Parameter** | 비트맵, 구하고자 하는 부분의 인덱스 |
| **Return** | 해당 인덱스의 값 |
| **Function** | 비트맵 b에서 인덱스 idx에 해당하는 부분의 값을 반환해준다. (b->bits[elem\_idx (idx)]) |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_all(struct bitmap \*b, bool value) |
| **Parameter** | 비트맵, 바꾸고자 하는 값 |
| **Return** | X |
| **Function** | 비트멥에서 모든 비트를 value (true또는 false)로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | void bitmap\_set\_multiple (struct bitmap \*b, size\_t start, size\_t size, bool value) |
| **Parameter** | 비트맵, 바꾸고자 하는 부분의 인덱스, 바꾸고자 하는 부분 길이, 바꾸고자 하는 값 |
| **Return** | X |
| **Function** | 비트멥에서 인덱스가 start인 부분부터 size만큼의 크기의 비트를 value (true 또는 false)로 바꾸어 준다. |

|  |  |
| --- | --- |
| **Prototype** | size\_t bitmap\_count (struct bitmap \*b, size\_t start, size\_t size, bool value) |
| **Parameter** | 비트맵, 구하고자 하는 부분의 첫 인덱스, 구하고자 하는 부분의 첫 인덱스로부터의 길이, 바꾸고자 하는 값 |
| **Return** | value (true 또는 false)인 비트의 개수 |
| **Function** | 비트멥에서 인덱스가 start인 부분부터 start+size까지의 비트 중 값이 value (true 또는 false)인 비트의 개수를 반환한다. |

|  |  |
| --- | --- |
| **Prototype** | bool bitmap\_scan (const struct bitmap \*b, size\_t start, size\_t cnt, bool value) |
| **Parameter** | 비트맵, 구하고자 하는 부분의 첫 인덱스, 구하고자 하는 부분의 첫 인덱스로부터의 길이, 확인하고자 하는 값 |
| **Return** | 비트가 모두 value이면 true 아니면 false를 반환 |
| **Function** | 인덱스가 start부터 start+cnt인 부분의 비트가 모두 value이면 true 아니면 false를 반환단다. |