2016학년도 2학기 운영체제

Operating System Homework #2 Document

2반 20141550 윤은주

<List> list.c list.h

```
struct list_sub
{
        char name[10];
        struct list element;
};
struct list_node
{
        int numinfo;
        struct list_elem pointer;
};
```

1) create_list : list_init
2) delete_list : list_begin, list_end, list_remove
3) dumpdata_list : list_size, list_head, list_next,list_end, list_entry
4) list_insert : list_head, list_next, insert_list()
5) list_splice : list_begin, list_next, list_remove, list_entry
6) list_push_front : insert_list()
7) list_push_back : insert_list()
8) list_remove : list_begin, list_next, list_remove
9) list_pop_front : list_pop_front
10) list_pop_back : list_pop_back
11) list_front : list_front, list_entry
12) list_back : list_back, list_entry
13) list_size : list_size
14) list_empty : list_empty
15) list_reverse : list_reverse
16) list_sort : list_sort
17) list_insert_ordered : list_sort, insert_list()
18) list_unique : list_size, list_begin,list_end, list_entry, list_unique, list_next
19) list_max : list_max, list_entry
20) list_min : list_min, list_entry
21) list_swap : list_begin, list_next, list_entry, list_swap()
22) list_shuffle : list_shuffle()

> list_init : 파라미터로 list를 받아, head와 tail만을 가진 empty list로 초기화한다.
> list_begin, list_head, list_end, list_next : list_elem을 통해 list의 원소들 간의 이동을 할 때, 주로 쓰이는 함수들로, list_begin과 list_head, list_end는 list를 파라미터로 받아, 각각 list의 첫 번째 노드의 포인터, list의 head, list의 tail을 반환하는 함수이다. list_next는 list_elem 포인터를 파라미터로 받아, 해당 포인터의 next 포인터를 반환한다.
> list_entry : 어떤 list_elem 포인터가 가리키고 있는 어떤 structure의 type과 해당 타입에서 list_elem의 변수명을 알고 있다면, 해당 structure의 처음 주소값을 반환한다. 이를 통해, list_elem 변수를 통해 연결만 되어 있다면, 다양한 type의 node에 access 할 수 있다.
> list_remove : list_elem 포인터를 파라미터로 받아, 해당 포인터의 노드를 list에서 제거하는 함수이다.
> list_pop_front, list_pop_back : list를 파라미터로 받아, 각각 list의 첫 번째 노드와 마지막 노드의 포인터를 반환하는 함수이다.

> list_front, list_back : list를 파라미터로 받아, 각각 list의 첫 번째 노드와 마지막 노드의 포인터를 반환하는 함수이다.
> list_size, list_empty : list를 파라미터로 받아, 각각 list의 node 개수와 list가 empty인지 여부를 반환하는 함수이다.
> list_reverse, list_sort : list를 파라미터로 받아, 각각 list의 순서를 바꾸고, list를 오름차순으로 정렬하는 함수이다. 이 때, list_sort는 추가로 list_less_func의 함수 포인터를 받아, 그를 적절히 이용하여 함수가 실행된다.
> list_unique : list를 파라미터로 받아, list 내에 같은 원소가 있을 때는 첫 번째 노드를 제외한 나머지를 모두 삭제하는 함수이다. 이 때도 list_less_func의 함수 포인터를 받아, 두 수의 대소비교에 사용된다.
> list_max, list_min : list를 파라미터로 받아, 각각 list 내의 최댓값과 최솟값의 노드의 포인터를 반환하는 함수이다. 여기서 두 수를 비교할 때도 list_less_func의 함수 포인터가 사용된다.

*insert_list(struct list_sub *list, int num, int order)
: 파라미터로 받은 list에 num을 정보로 갖는 새로운 list_node를 order번째 자리에 삽입하는 함수이다.
*list_swap(struct list_elem *a, struct list_elem *b)
: 파라미터로 받은 struct list_elem 포인터 변수 a와 b가 각각 가리키는 곳에 저장된 숫자를 서로 바꾸는 함수이다.
*list_shuffle(struct list *list)
: 파라미터로 받은 list의 원소 순서를 랜덤으로 바꾸는 함수이다. 위에서 정의한 list_swap을 이용한다.

<Hashtable> hash.c hash.h

```
struct hash_sub
{
        char name[10];
        struct hash element;
};
struct hash_node
{
        int numinfo;
        struct hash_elem pointer;
}
```

1) create_hash : hash_init
2) delete_hash : hash_clear
3) dumpdata_hash : hash_size, hash_first, hash_next, hash_entry
4) hash_insert : hash_insert, newnode_hash()
5) hash_replace : hash_replace, newnode_hash()
6) hash_find : hash_find, newnode_hash()
7) hash_delete : hash_delete, newnode_hash()
8) hash_clear : hash_clear
9) hash_size : hash_size
10) hash_empty : hash_empty

11) hash_apply : hash_apply
12) hash_int_2 :

> hash_init : hash table을 파라미터로 받아, hash table을 초기화하고 hash값을 계산하여 지정해주는 함수이다.
> hash_clear : hash table을 파라미터로 받아, hash의 element에 쓰인 메모리 할당을 해제하는 함수이다.
> hash_first, hash_next : hash table 내의 element 간에 이동할 때 주로 쓰이는 함수들로, hash table과 hash_iterator를 파라미터로 받아 각각 hash table의 첫 번째 원소를 hash_iterator에 반환하고, 현재 hash_iterator의 다음 위치를 반환하는 함수이다.
> hash_size, hash_empty : hash table을 파라미터로 받아, 각각 hash table의 element 개수 혹은 hash table의 empty 여부를 반환하는 함수이다.
> hash_entry : list_entry와 동일하게, 어떤 hash_elem 포인터가 가리키고 있는 어떤 structure의 type과 해당 타입에서 hash_elem의 변수명을 알고 있다면, 해당 structure의 처음 주소값을 반환한다.
> hash_insert, hash_replace : hash table과 새로운 노드의 hash_elem 포인터를 파라미터로 받아, hash table에 삽입하는 함수이다. 단, hash table 내에 새로운 노드와 같은 원소가 있다면, hash_insert는 새로운 노드의 포인터를, hash_replace는 이전의 노드의 포인터를 반환한다는 차이점이 있다.
> hash_find : hash table과 어떤 노드의 hash_elem 포인터를 파라미터로 받아, hash table 안에 동일한 원소의 노드를 반환하는 함수이다.
> hash_delete : hash table과 어떤 노드의 hash_elem 포인터를 파라미터로 받아, hash table 내의 동일한 원소를 제거하여 반환하는 함수이다.
> hash_apply : hash table과 hash_action_func 함수 포인터를 파라미터로 받아, hash table의 각각의 element에 대하여 해당 함수를 실행하는 함수이다.

*newnode_hash() : hash table에 새롭게 저장할 정보를 파라미터로 받아, 새로운 노드를 형성해주는 함수로써, hash_insert를 손쉽게 하기 위한 기초 작업 함수이다.
*hash_int_2(int i) : 파라미터로 받은 정수 i의 hash 값을 임의로 설정하는 함수이다.

<Bitmap> bitmap.c bitmap.h

```
struct bitmap_sub
{
        char name[10];
        struct bitmap *element;
}
```

1) create_bitmap : bitmap_create
2) delete_bitmap : bitmap_destroy
3) dumpdata_bitmap : bitmap_size, bitmap_test
4) bitmap_size : bitmap_size
5) bitmap_set : bitmap_set
6) bitmap_mark : bitmap_mark
7) bitmap_reset : bitmap_reset
8) bitmap_flip : bitmap_flip
9) bitmap_test : bitmap_test

10) bitmap_set_all : bitmap_set_all
11) bitmap_set_multiple : bitmap_set_multiple
12) bitmap_count : bitmap_count
13) bitmap_contains : bitmap_contains
14) bitmap_any : bitmap_any
15) bitmap_none : bitmap_none
16) bitmap_all : bitmap_all
17) bitmap_scan : bitmap_scan
18) bitmap_scan_and_flip : bitmap_scan_and_flip
19) bitmap_dump : bitmap_dump
20) bitmap_expand : bitmap_create, bitmap_test, bitmap_set

> bitmap_create : bitmap의 size를 파라미터로 받아, 해당 크기만큼 bitmap을 false로 초기화하는 함수이다.
> bitmap_destroy : bitmap을 파라미터로 받아, 해당 bitmap의 저장 공간을 free시킴으로써 bitmap을 제거하는 함수이다.
> bitmap_size : bitmap을 파라미터로 받아, bitmap의 size를 반환하는 함수이다.
> bitmap_test : bitmap과 bitmap의 index를 파라미터로 받아, 해당 index의 value를 반환하는 함수이다.
> bitmap_set, bitmap_set_all, bitmap_set_multiple : bitmap_set은 bitmap과 index, value를 파라미터로 받아, bitmap 내의 해당 index의 값을 value로 설정해주는 함수이다. bitmap_set_all은 bitmap 내의 모든 값을 파라미터인 value로 설정해주는 함수이고, bitmap_set_multiple은 시작 index와 count를 파라미터로 받아, bitmap 내의 해당 index부터 count만큼의 값을 value로 설정해주는 함수이다.
> bitmap_mark, bitmap_reset, bitmap_flip : bitmap과 size를 파라미터로 받아, bitmap_mark는 해당 bitmap 내의 모든 값을 true로, bitmap_reset은 해당 bitmap 내의 모든 값을 false라고 설정해주는 함수이다. bitmap_flip은 해당 bitmap 내의 모든 값을 바꾸어 설정해주는 함수이다.
> bitmap_count : bitmap과 시작 index, count, value를 파라미터로 받아, 해당 bitmap의 시작 index부터 count만큼의 값 중에서 value와 같은 값을 갖는 bit의 수를 반환하는 함수이다.
> bitmap_contains, bitmap_any : bitmap과 시작 index, count, value를 파라미터로 받아, 해당 bitmap의 시작 index부터 count만큼의 값 중에서, bitmap_contains는 같은 value를 갖는 bit가 있는지 여부를, bitmap_any는 true 값을 갖는 bit가 있는지 여부를 체크하는 함수이다.
> bitmap_none, bitmap_all : bitmap과 시작 index, count를 파라미터로 받아, 해당 bitmap의 시작 index부터 count만큼의 값 중에서, bitmap_none은 모두 true가 아닌지 여부를, bitmap_all은 모두 true인지 여부를 체크하는 함수이다.
> bitmap_scan, bitmap_scan_and_flip : bitmap과 시작 index, count, value를 파라미터로 받아, 해당 bitmap의 index부터 시작하여 bitmap_scan은 count만큼의 bit가 value값을 갖는, bitmap_scan_and_flip은 count만큼의 bit가 value와 다른 값을 갖는 첫 번째 인덱스를 반환하는 함수이다.
> bitmap_dump : bitmap을 파라미터로 받아, 해당 bitmap에 저장된 내용을 16진수로 나타내는 함수이다.

*struct bitmap *bitmap_expand(struct bitmap *bitmap, int size)

: 파라미터로 받은 bitmap을 size만큼 확장하여 주는 함수이다. 이 때, 기존에 저장된 정보 bitmap_test를 통해 알 수 있고, bitmap_set을 통해 새로운 bitmap에 저장할 수 있고, 새로 확장된 부분은 모두 false로 초기화한다.