Operating System Project #0-2

20141531

변재승

1. bitmap

bitmap 관련 command들을 처리하기 위해서는 bitmap의 name과 array를 저장할 수 있는 bit\_array라는 구조체를 만들고 배열로 할당하여 create된 bitmap array를 저장한다. bitmap array는 최대 10개까지 만들 수 있다.

(1) bitmap\_create

format -> struct bitmap \*bitmap\_create (size\_t bit\_cnt)

기능 : bit\_cnt만큼의 bitmap을 초기화하고 bitmap의 모든 bit를 false로 설정한다. bitmap이 create되면 true를 메모리 할당을 실패하면 fail을 return한다.

(2) bitmap\_dump

format -> void bitmap\_dump (const struct bitmap \*b)

기능 : bitmap에 저장된 content들을 16진수로 출력한다

(3) bitmap\_mark

format -> void bitmap\_mark ( struct bitmap \*b, size\_t bit\_idx)

기능 : bitmap array의 bit\_idx번째 bit를 true로 바꿔준다.

(4) bitmap\_all

format -> bool bitmap\_all ( const struct bitmap \*b, size\_t start, size\_t cnt)

기능 : bitmap array의 start번째 start + cnt 번째까지의 비트가 모두 true이면 true를 return하고 하나라도 false이면 false를 return한다.

(5) bitmap\_destroy

format -> void bitmap\_destroy ( struct bitmap \*b)

기능 : bitmap array의 storage의 할당을 해제한다. bitmap\_create\_preallocated()로 만들어진 bitmap한테는 사용할 수 없다.

(6) bitmap\_any

format -> bool bitmap\_any ( const struct bitmap \*b, size\_t start, size\_t cnt)

기능 : bitmap array의 start번째부터 start + cnt 번째까지의 비트중 true가 존재하면 return true, 모두 false이면 false를 return한다.

(7) bitmap\_contains

format -> bool bitmap\_contains ( const struct bitmap \*b, size\_t start, size\_t cnt, bool value)

기능 : bitmap array의 start번째부터 start + cnt번째까지의 비트중 value와 일치하는 비트가 존재하면 true를 return하고 존재하지 않으면 false를 return 한다.

(8) bitmap\_count

format -> size\_t bitmap\_count ( const struct bitmap \*b, size\_t start, size\_t cnt, bool value)

기능 : bitmap array의 start번째부터 start + cnt번째까지의 비트중 value와 일치하는 비트의 개수를 return한다.

(9) bitmap\_expand

format -> struct bitmap \*bitmap\_expand ( struct bitmap \*bm, int size)

기능 : 확장할 bitmap array를 받아서 size의 개수만큼 확장한다. 확장에 성공할 경우 확장된 bitmap을 return하고, 실패하면 NULL을 return 한다.

(10) bitmap\_set\_all

format -> void bitmap\_set\_all ( struct bitmap \*b, bool value)

기능 : bitmap array의 모든 비트들을 value와 동일하게 설정한다.

(11) bitmap\_flip

format -> void bitmap\_flip ( struct bitmap \*b, size\_t bit\_idx)

기능 : bitmap array의 idx번째 비트가 true이면 false로 바꾸고, false이면 true로 바꾼다.

(12) bitmap\_none

format -> bool bitmap\_none ( const struct bitmap \*b, size\_t start, size\_t cnt)

기능 : bitmap array의 start번째부터 start+cnt번째까지의 비트가 true로 설정되어 있지 않으면 true를 돌려주고 false로 설정되어 있지 않으면 false를 return한다.

(13) bitmap\_reset

format -> void bitmap\_reset ( struct bitmap \*b, size\_t bit\_idx)

기능 : bitmap array의 idx번째 비트를 false로 설정한다.

(14) bitmap\_scan\_and\_flip

format -> size\_t bitmap\_scan\_and\_flip ( struct bitmap \*b, size\_t start, size\_t cnt, bool value)

기능 : bitmap array의 start이후의 비트들 중에서 value와 일치하는 비트가 cnt만큼 연속적으로 존재하면 그 비트들을 value와 반대되는 value로 설정하고 첫번째 시작되는 bit의 위치를 출력한다.

(15) bitmap\_scan

format -> size\_t bitmap\_scan ( const struct bitmap \*b, size\_t start, size\_t cnt, bool value)

기능 : bitmap array의 start이후의 비트들 중에서 cnt만큼 value와 일치하는 비트가 cnt만큼 연속적으로 존재하면 그 비트들이 시작되는 위치를 출력한다.

(16) bitmap\_set

format -> void bitmap\_set ( struct bitmap \*b, size\_t idx, bool value)

기능 : bitmap array의 idx번째 비트를 value로 설정한다.

(17) bitmap\_set\_multiple

format -> void bitmap\_set\_multiple ( struct bitmap \*b, size\_t start, size\_t cnt, bool value)

기능 : bitmap array의 start번째부터 start + cnt번째까지의 비트를 value로 설정한다.

(18) bitmap\_size

format -> size\_t bitmap\_size ( const struct bitmap \*b)

기능 : bitmap array의 size를 return 한다.

(19) bitmap\_test

format -> bool bitmap\_test ( const struct bitmap \*b, size\_t idx)

기능 : bitmap array의 idx번째 value를 return한다.

2. list

doubly linked list를 만들때에는 총 4가지 structure가 필요하다

list\_array : 맨 처음에 create된 doubly linked list를 저장하는 구조체 배열, 최대 10개까지 저장가능하면 linked list와 그 이름이 저장되어 있다.

list : list\_array에 연결된 doubly linked list의 head와 tail의 정보를 갖고있는 구조체

list\_item : head와 tail사이에 추가할 list의 data와 list\_elem을 갖고있는 구조체

list\_elem : doubly linked list를 위해 필요한 link들이 저장되어 있는 구조체

(1) create\_list

format -> struct list \*create\_list()

기능 : list structure형 변수에 메모리를 할당한다.

(2) list\_init

format -> void list\_init ( struct list \*list)

기능 : list structure를 비어있는 list로 초기화한다.

(3) list\_remove

format -> struct list\_elem \* list\_remove ( struct list\_elem \*elem)

기능 : list에서 elem에 해당하는 list\_elem을 삭제한다.

(4) list\_begin

format -> struct list\_elem \*list\_begin ( struct list \*list)

기능 : 현재 list의 제일 처음 element를 갖고 있는 ist\_elem을 return한다.

(5) list\_tail

format -> struct list\_elem \*list\_tail ( struct list \*list)

기능 : 현재 list의 tail을 return한다.

(6) list\_next

format -> struct list\_elem \*list\_next ( struct list\_elem \*elem)

기능 : 현재 가리키고 있는 list\_elem의 다음 list\_elem으로 이동해준다. 만약 현재 list\_elem이 list의 마지막 element이면 tail을 return한다. list\_elem이 tail일 경우는 result가 undefine하게 된다.

(7) list\_entry

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

기능 : 입력받은 list\_elem을 이용하여 해당 list\_elem을 MEMBER로 포함하고 있는 STRUCT형 변수를 return한다. list\_item의 bar에 접근할 때 사용된다.

(8) list\_push\_back

format -> void list\_push\_back ( struct list \*list, struct list\_elem \*elem)

기능 : list\_elem을 list의 맨뒷부분에 추가해준다.list\_elem이 list의 맨뒷부분이 된다.

(9) list\_front

format -> struct list\_elem \*list front ( struct list \*list)

기능 : list의 element를 갖고있는 가장 앞 list\_elem을 return한다.

(10) list\_back

format -> struct list\_elem \*list\_back ( struct list \*list)

기능 : list의 맨 뒤 list\_elem을 return한다.

(11) list\_pop\_back

format -> struct list\_elem \*list\_pop\_front ( struct list \*list)

기능 : list의 맨 뒤 list\_elem을 지우고 return한다.

(12) list\_pop\_front

format -> struct list\_elem \*list\_pop\_back ( struct list \*list)

기능 : list의 맨 앞 list\_elem을 지우고 return한다.

(13) list\_insert

format -> void list\_insert ( struct list\_elem \*before, struct list\_elem \*elem)

기능 : list\_elem \*before의 바로 앞 부분에 list\_elem \*elem을 추가해준다.

(14) less

format -> bool less ( const struct list\_elem \*a, const struct list\_elem \*b, void \*aux)

기능 : a와 b를 비교해서 a가 b보다 작으면 true 크면 false를 return한다. 만일 list\_elem안에 여러 변수가 존재하면 그 중 하나를 골라서 비교를 해야 하므로 aux를 이용해야 하지만 이 프로젝트에서 변수는 bar 하나밖에 없으므로 aux에 NULL을 넣는다.

(15) list\_insert\_ordered

format -> list\_insert\_ordered ( struct list \*list, struct list\_elem \*elem, list\_less\_func \*less, void \*aux)

기능 : less함수를 이용하여 list안에 list\_elem이 저장될 위치를 찾은 후 넣어준다.

(16) list\_empty

format -> bool list\_empty ( struct list \*list)

기능 : list가 비어있으면 true를 아니면 false를 return한다.

(17) list\_size

format -> size\_t list\_size ( struct list \*list)

기능 : list의 size를 return한다.

(18) list\_max

format -> struct list\_elem \*list\_max ( struct list \*list, list\_less\_func \*less, void \*aux)

기능 : list안의 list\_elem들 중에서 제일 큰 bar를 가리키고 있는 list\_elem을 return한다.

(19) list\_min

format -> struct list\_elem \*list\_min ( struct list \*list, list\_less\_func \*less, void \*aux)

기능 : list안의 list\_elem들 중에서 제일 작은 bar를 가리키고 있는 list\_elem을 return한다.

(20) list\_reverse

format -> void list\_reverse (struct list \*list)

기능 : list의 elem들을 모두 reverse해서 return한다.

(21) list\_sort

format -> void list\_sort (struct list \*list, list\_less\_func \*less, void \*aux)

기능 : less함수를 이용하여 list를 natural iterative merge sort를 이용하여 sort한다.

(22) list\_splice

format -> void list\_splice ( struct list\_elem \*before, struct list\_elem \*first, struct list\_elem \*last)

기능 : list의 first번째부터 last까지의 list\_elem을 잘라서 다른 list의 list\_elem before 앞에 붙여준다.

(23) list\_push\_front

format -> void list\_push\_front ( struct list \*list, struct list\_elem \*elem)

기능 : list의 맨 처음 부분에 list\_elem을 추가해주고 그 list\_elem은 리스트의 front가 된다.

(24) list\_swap

format -> void list\_swap ( struct list\_elem \*a, struct list\_elem \*b)

기능: list\_elem a와 list\_elem b가 가리키고 있는 element를 swap한다.

(25) list\_shuffle

format -> void list\_shuffle ( struct list \*list)

기능 : list를 무작위로 shuffle한다.

구현방식 : random함수를 이용하여 list의 size보다 작은 무작위의 2개의 수를 정한 후 그 위치에 해당하는 list\_elem까지 이동하여 element를 교환한다.

(26) list\_unique

format -> void list\_unique ( struct list \*list, struct list \*duplicates, list\_less\_func \*less, void \*aux)

기능 : list를 처음부터 끝까지 읽으면서 연속한 list\_elem가 같은 element를 가리키고 있으면 한 개만 남기고 제거하여 duplicates에 붙여준다.

(27) list\_head

format -> struct list\_elem \*list\_head ( struct list \*list)

기능 : list의 head를 return 한다.

3. hash

hashtable을 구현하고 command를 진행하기 위해서는 5개의 structure가 필요하다

hash\_array : hashtable의 이름과 hash structure가 저장되어 있다. 최대 10개까지 저장가능하다.

hash : hastable의 정보와 bucket의 개수 그리고 bucket들이 doubly linked list형태로 저장되어 있는 array, hash\_hash\_function, hash\_less\_function이 저장되어 있다

hash\_iterator : hashtable을 돌면서 현재 bucket의 위치와 그 bucket이 포함하고 있는 data를 찾을 때 사용되는 structure이다.

hash\_item : hash\_elem이 가리키고 있는 data를 저장하고 있는 structure이다.

hash\_elem : bucket이 갖고있는 data에 접근할 때 사용되는 structure이다.

(1) create\_hash\_iterator

format -> struct hash\_iterator \*create\_hash\_iterator

기능 : hash\_iterator 타입의 구조체를 만든다.

(2) create\_hash

format -> struct hash \*create\_hash

기능 : hash타입의 구조체를 만든다.

(3) create\_hash\_item

format -> struct hash\_item \*create\_hash\_item

기능 : hash\_item 타입의 구조체를 만든다.

(4) hash\_init

format -> bool hash\_init ( struct hash \*h, hash\_hash\_func \*hash, hash\_less\_func \*less, void \*aux)

기능 : h에 저장된 hashtable을 hash\_hash\_func을 이용하여 hash값을 만들고 hash\_less\_func을 이용하여 hash element를 비교하여 초기화한다.

(5) hash\_first

format -> void hash\_first ( struct hash\_iterator \*I, struct hash \*h)

기능 : hashtable을 hash\_iterator에 저장한 후 첫번째 bucket을 가리키게 한다.

(6) hash\_next

format -> struct hash\_elem \*hash\_next ( struct hash\_iterator \*i)

기능 : hash\_iterator가 가리키고 있는 bucket의 다음 bucket으로 이동해주는 함수

(7) hash\_entry

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

기능 : STRUCT형 타입의 변수 중에서 hash\_elem을 MEMBER로 포함하고 있는지 확인이 되었으면 그 STRUCT형 변수를 return한다.

(8) hash\_insert

format -> struct hash\_elem \*hash\_insert ( struct hash \*h, struct hash\_elem \*new)

기능 : hashtable h에 hsah\_elem을 추가한다.

(9) hash\_apply

format -> void hash\_apply ( struct hash \*h, hash\_action\_func \*action)

기능 : hashtable에 저장된 모든 element에 hash\_action\_func을 적용한다.

(10) square

format -> void square ( struct hash\_elem \*a, void \*aux)

기능 : hash\_elem이 가리키는 element를 제곱한다.

(11) triple

format -> void triple ( struct hash\_elem \*a, void \*aux)

기능 : hash\_elem이 가리키는 element를 세제곱한다.

(12) hash\_hah\_func

format -> unsigned hf ( const struct hash\_elem \*e, void \*aux)

기능 : hash\_elem가 가리키고 있는 element를 hash\_function을 적용해 return한다.

(13) hash\_less\_func

format -> bool hl ( const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux)

기능 : hash\_elem a가 가리키는 element와 hash\_elem b가 가리키는 element를 비교해서 a가 b보다 작으면 true, 크면 false를 return한다.

(14) hash\_int

format -> unsigned hash\_int ( int i)

기능 : int i를 Fowler-Noll-Vo 32bit hash를 이용하여 hash한 값을 return한다.

(15) hash\_find

format -> struct hash\_elem \*hash\_find (struct hash \*h, struct hash\_elem \*e)

기능 : hashtable h안에 존재하는 hash\_elem의 위치를 return한다. 존재하지 않을 경우 NULL을 return한다.

(16) hash\_delete

format -> struct hash\_elem \*hash\_delete ( struct hash \*h, struct hash\_elem \*e)

기능 : hash\_elem이 가리키는 bucket을 hashtable에서 지운다.

(17) hash\_clear

format -> void hash\_clear ( struct hash \*h, hash\_action\_fun \*destructor)

기능 : hashtable의 모든 element를 지운다.

(18) hash\_destructor

format -> void hash\_destructor ( struct hash\_elem \*a, void \*aux)

기능 : hash\_elem을 free한다.

(19) hash\_destroy

format -> void hash\_destroy ( struct hash \*h, hash\_action\_fun \*destructor)

기능 : hashtable h의 element를 모두 지우고 메모리 할당을 해제한다.

(20) hash\_replace

format -> struct hash\_elem \*hasah\_replace ( struct hash \*h, struct hash\_elem \*new)

기능 : 새로운 hash\_elem을 추가할 때 hash\_elem이 가리키는 data가 hashtable안에 존재하면 예전의 hash\_elem을 return하고 새로운 hash\_elem을 넣는다.

(21) hash\_size

format -> size\_t hash\_size ( struct hash \*h)

기능 : hashtable h의 size를 return한다.

(22) hash\_empty

format -> bool hash\_empty ( struct hash \*h)

기능 : hashtable h의 element가 하나도 존재하지 않으면 true를 return한다. 존재할 경우 false를 return한다.

(23) hash\_int\_2 ( int i)

format -> unsigned hash\_int\_2 ( int i)

기능 : integer i를 hash해주는 함수로써, key와 key에 대응하는 hash value를 알아도 계산법을 알아내기 힘들게 key에 123456을 곱한 후 3245를 나누고 다시 112432를 곱한 후 43212354로 나눈 나머지 값이 hash value에 저장되도록 구현하였다. return 값이 unsigned int값이므로 0 ~ 4,294,967,295의 범위안에 들어오도록 하였다.