**Pintos Project 0\_2 : User Program Basic**

**(설계 프로젝트 수행 결과)**

과목 명 : 운영체제

담당 교수 : 소정민

조 / 조원 : 정재훈 20151607

개발 기간 : 2018/9/16 ~ 2018/9/23

1. 프로그램 개요

Pintos Kernel Data Structure 중 list, hash table, bitmap에 특정 기능을 하는 함수를 추가하고 그것의 동작을 확인할 수 있는 프로그램 (Interactive Program을 작성한다)

1. 프로그램 구조

Main Module

사용자의 입력을 받으며 각 입력을 parsing 하여 적절한 함수에 인자를 전달해준다.

symbol table을 관리하며 데이터 이름이 중복되지 않게, 데이터 이름에 따라 접근할 수 있게 해준다.

종료시에 할당되어 있는 데이터 구조들을 모두 해체해준다.

List Command Module

리스트에 대한 명령어들을 관리한다.

Hash table Command Module

해쉬 테이블에 대한 명령어들을 관리한다.

Bitmap Command Module

비트맵에 대한 명령어들을 관리한다.

1. 모듈 함수 설명

<Main Module – 20151607.c 20151607.h>

**EXECUTE COMMAND FUNCTION**

1. int execute\_command(char arg[])

Execute command of user

categorize : create , delete, quit, list, hashtable, bitmap

parameter : argument of stdin

return : true if no error, otherwise false

1. int execute\_create(char arg[])

Execute create command of list, hashtable and bitmap

>> create list list1 : create list whose name is list1

>> create hashtable hash1 : create hashtable whose name is hash1

>> create bitmap bitmap1 bit\_cnt : create bitmap whose name is bitmap1 and bit size is bit\_cnt

parameter : command string except for "create"

return : false if the data name is already exist in program

else true

1. void execute\_quit()

Deallocate all of allocated lists, hashtables, bitmaps and symbol table

and quit the program.

>> quit

1. int execute\_list(struct symbol\* s, char command[], char arg[])

Execute list command which user inputs

>> list\_command list\_name parameter

COMMAND LIST : list\_insert, list\_splice, list\_push\_front, list\_push\_back

list\_pop\_front, list\_pop\_back, list\_front, list\_back, list\_size

list\_empty, list\_sort, list\_reverse, list\_insert\_ordered

list\_unique, list\_max, list\_min, list\_swap, list\_remove, list\_shuffle

parameter : symbol, command\_name, arg

return : true if no error, otherwise false

1. int execute\_hashtable(struct symbol \*s, char command[], char arg[])

Execute hash table command which user inputs

>> hash\_table\_command hash\_table\_name parameter

COMMAND LIST : hash\_insert, hash\_replace, hash\_find, hash\_delete

hash\_clear, hash\_size, hash\_empty, hash\_apply

parameter : symbol, command\_name, arg

return : true if no error, otherwise false

1. int execute\_bitmap(struct symbol\* s, char command[], char arg[])

Execute bitmap command which user inputs

>> bitmap\_command bitmap\_name parameter

COMMAND LIST : bitmap\_size, bitmap\_set, bitmap\_mark, bitmap\_reset

bitmap\_flip, bitmap\_test, bitmap\_set\_all, bitmap\_set\_multiple

bitmap\_count, bitmap\_contains, bitmap\_any, bitmap\_none

bitmap\_all, bitmap\_scan, bitmap\_scan\_and\_flip, bitmap\_dump

bitmap\_expand

parameter : symbol, command\_name, arg

return : true if no error, otherwise false

**PARSING AND DATA TRANSFORM FUNCTION**

1. int str\_to\_int(char\* str)

Transform string to integer(4byte)

It's possible to transform signed string

parameter : string

return : int(string)

1. char\* read\_command(char\*\* parse)

Read and parse the argument by blank ' '

parameter : pointer of argument

return : pointer of argument which point to next part(char \*\*parse)

pointer of argument which point to current part(char \*ret)

**SYMBOL TABLE FUNCTION**

1. unsigned symbol\_hash\_func(const struct hash\_elem \*e, void \*aux)

Hash function of symbol table that hashes string data

parameter : hash element, (aux)

return : hash value

1. bool symbol\_hash\_less(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux)

String compare(less) function of symbol table

parameter : hash element a, hash element b

return : true if data of a is less than data of b, otherwise false

1. void symbol\_destructor(struct hash\_elem \*e, void \*aux)

Destructor which deallocate symbol element in symbol table

parameter : hash element, (aux)

**DELETE DATA AND DUMP FUNCTION**

1. void delete\_data\_struct(struct symbol\* symbol)

Delete data structure in symbol, remove the symbol in symboltable

and deallocate the symbol

parameter : symbol

1. void dumpdata(struct symbol\* symbol)

Show the content of data structure pointed by symbol

parameter : symbol

<List Command Module – list\_command.c list\_command.h>

**LIST COMMAND FUNCTION**

1. int create\_list(char list\_name[])

Create symbol whose name is list\_name in symbol table

parameter : list name

return : true if no error, otherwise false

1. int l\_insert(struct symbol\* symbol, size\_t idx, int data)

Insert integer in list according to index parameter

parameter : symbol of list, index, int data

return : true if no error, otherwise false

1. void l\_push\_front(struct symbol\* symbol, int data)

Insert integer in front of list

parameter : symbol of list, int data

1. void l\_push\_back(struct symbol\* symbol, int data)

Insert integer in rear of list

parameter : symbol of list, int data

1. int l\_pop\_front(struct symbol\* symbol)

Delete integer in front of list and deallocate

parameter : symbol of list, int data

return true if non empty, otherwise false

1. int l\_pop\_back(struct symbol\* symbol)

Delete integer in rear of list and deallocate

parameter : symbol of list, int data

return : true if non empty, otherwise false

1. int l\_front(struct symbol\* symbol)

Print integer in front of list

parameter : symbol of list, int data

return : true if non empty, otherwise false

1. int l\_back(struct symbol\* symbol)

Print integer in rear of list

parameter : symbol of list, int data

return : true if non empty, otherwise false

1. void l\_size(struct symbol\* symbol)

Print size of list

parameter : symbol of list

1. bool l\_empty(struct symbol\* symbol)

Print truth value about EMPTY

parameter : symbol of list

return true if list is empty, otherwise false

1. void l\_sort(struct symbol\* symbol)

Sort the list to accending order

parameter : symbol of list

1. void l\_reverse(struct symbol\* symbol)

Make the list reverse order

parameter : symbol of list

1. void l\_insert\_ordered(struct symbol\* symbol, int data)

Insert integer in list keeping accending order

parameter : symbol of list, int data

1. void l\_unique(struct symbol\* symbol, struct symbol\* duplicate)

duplicate list1 to list2 such that there no exist element which apear more than once

if list2 is NULL, list1 duplicate to itself and list1 will be deleted

parameter : symbol of list1, symbol of list2

1. int l\_max(struct symbol\* symbol)

Print max element in list

parameter : symbol of list

return : true if no error, otherwise false

1. int l\_min(struct symbol\* symbol)

Print min element in list

parameter : symbol of list

return : true if no error, otherwise false

1. int l\_swap(struct symbol\* symbol, size\_t idx1, size\_t idx2)

Swap element1 indexed idx1 and element2 indexed idx2

parameter : symbol of list, index1 ,index2

return : true if no error, otherwise false

1. l\_shuffle(struct symbol\* symbol)

Shuffle the list in some method

parameter : symbol of list

1. int l\_remove(struct symbol\* symbol, size\_t idx)

Remove element indexed idx in list

parameter : symbol of list, index

return : true if no error, otherwise false

1. int l\_splice(struct symbol\* dest, int before\_idx, struct symbol\* src, int start\_idx, int end\_idx)

Paste (start\_idx ~ end\_idx -1) of src list to before\_idx of dest list

and remove this part in src list

parameter : dest list, before index, src list, start index, end index

return : true if no error, otherwise false

**LIST COMPARE FUNCTION**

1. bool l\_less\_func(const struct list\_elem \*a, const struct list\_elem \*b, void \*aux)

Integer compare(less) function of list

parameter : list element a, list element b, (aux)

return true if data of a is less than data of b, otherwise false

<Hash table Command Module

– hash\_table\_command.c hash\_table\_command.h>

**HASH TABLE COMMAND FUNCTION**

1. int create\_hashtable(char hash\_table\_name[])

Create hash table whose name is hash\_table\_name

parameter : hash table name

return : true if no error, otherwise false

1. int h\_insert(struct symbol\* symbol, int data)

Insert integer data in hash table

parameter : symbol of hash table, int data

return : true if no error, otherwise false

1. int h\_replace(struct symbol\* symbol, int data)

Replace old integer data with new integer data

if there already exist same integer, delete old and insert new

otherwise only insert new data

parameter : symbol of hash table, int data

return : true if no error, otherwise false

1. int h\_find(struct symbol\* symbol, int data)

Find the integer in hash table and print that

parameter : symbol of hash table, int data

return : if exist, print the integer and return 1

if no exist, return 0

1. int h\_delete(struct symbol\* symbol, int data)

Delete the integer in hash table

parameter : symbol of hash table, int data

return : if exsit, delete in hash table ,deallocate it and return 1

if not, return 0

1. void h\_clear(struct symbol\* symbol)

Clear the hash table and make hash table empty

parameter : symbol of hash table

1. void h\_size(struct symbol\* symbol)

Print the size of hash table,

that is the number of elements in hash table

parameter : symbol of hash table

1. void h\_empty(struct symbol\* symbol)

Print the truth value of wheter the hash table is empty

parameter : symbol of hash table

1. void h\_apply(struct symbol\* symbol, char func[])

Apply function to all of data in hash table

APPLY FUNC LIST : square(x -> x\*x), triple(x -> x\*x\*x)

parameter : symbol of hash table, apply function

**HASH APPLY ACTION FUNCTION**

1. void h\_square(struct hash\_elem \*e, void \*aux)

Apply function which square the data in hash element

parameter : hash element, (aux)

1. void h\_triple(struct hash\_elem \*e, void \*aux)

Apply function which triple the data in hash element

parameter : hash element, (aux)

**HASH ELEMENT DEALLOCATE FUNCTION**

1. void h\_destructor(struct hash\_elem \*e, void \*aux)

Destructor function which deallocate hash element

parameter : hash element, (aux)

**HASH FUNCTION AND COMPARE FUNCTION**

1. unsigned h\_hash\_func(const struct hash\_elem \*e, void \*aux)

Hash function which make hash value of integer in element

parameter : hash element, (aux)

return : hash value of integer

1. bool h\_less\_func(const struct hash\_elem \*a, const struct hash\_elem \*b, void \*aux)

Integer compare(less) function of hash table

parameter : hash element a, hash element b

return true if data of a is less than data of b, otherwise false

<Bitmap Command Module

– bitmap\_command.c bitmap\_command.h>

**BITMAP COMMAND FUNCTION**

1. int create\_bitmap(char bitmap\_name[], size\_t bit\_cnt)

Create bitmap whose name is bitmap\_name and size is bit\_cnt in symbol table

parameter : bitmap name, bit count

return true if no error, otherwise false

1. void b\_size(struct symbol\* symbol)

Print bit count of bitmap

parameter : symbol of bitmap

1. int b\_set(struct symbol\* symbol, size\_t idx, bool value)

Set the truth value in idx position

parameter : symbol of bitmap, index of position, truth value

return : true if no error, otherwise false

1. int b\_mark(struct symbol\* symbol, size\_t idx)

Set the true value in idx position

parameter : symbol of bitmap, index of position

return : true if no error, otherwise false

1. int b\_reset(struct symbol\* symbol, size\_t idx)

Set the false value in idx position

parameter : symbol of bitmap, index of position

return : true if no error, otherwise false

1. int b\_flip(struct symbol\* symbol, size\_t idx)

Flip the truth value in idx position

true -> false, false -> true

parameter : symbol of bitmap, index of position

return : true if no error, otherwise false

1. int b\_test(struct symbol\* symbol, size\_t idx)

Print truth value of idx position in bitmap

parameter : symbol of bitmap, index of position

return : true if no error, otherwise false

1. void b\_set\_all(struct symbol\* symbol, bool value)

Set all bits in bitmap to value

parameter : symbol of bitmap, truth value

1. int b\_set\_multiple(struct symbol\* symbol, size\_t start, size\_t cnt, bool value)

Set cnt bits starting at start position in bitmap to value

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_count(struct symbol\* symbol, size\_t start, size\_t cnt, bool value)

Print the number of bits in bitmap between start and start + cnt - 1

that are set to value

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_contains(struct symbol\* symbol, size\_t start, size\_t cnt, bool value)

Print the truth value whether any bits in bitmap betwwen start and start + cnt - 1

are set to value

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_any(struct symbol\* symbol, size\_t start, size\_t cnt)

Print the truth value whether any bits in bitmap betwwen start and start + cnt - 1

are set to TRUE

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_none(struct symbol\* symbol, size\_t start, size\_t cnt)

Print the truth value whether all bits in bitmap betwwen start and start + cnt - 1

are set to FALSE

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_all(struct symbol\* symbol, size\_t start, size\_t cnt)

Print the truth value whether all bits in bitmap betwwen start and start + cnt - 1

are set to TRUE

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_scan(struct symbol\* symbol, size\_t start, size\_t cnt, bool value)

Print the starting index of first group of cnt consecutive bits in bitmap

at or after start that are all set to value

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. int b\_scan\_and\_flip(struct symbol\* symbol, size\_t start, size\_t cnt, bool value)

Print the starting index of first group of cnt consecutive bits in bitmap

at or after start that are all set to value

and flips them all to !value

parameter : symbol of bitmap, start index, count, truth value

return : true if no error, otherwise false

1. void b\_dump(struct symbol\* symbol)

Dump the contest of bitmap to console as hexadecimal

parameter : symbol of bitmap

1. int b\_expand(struct symbol\* symbol, int size)

Expand B to be a bitmap of bit\_cnt + size bits

parameter : symbol of bitmap, size

return : true if no error, otherwise false

1. 추가 함수 & 기능

**-LIST**

1) int l\_swap(struct symbol\* symbol, size\_t idx1, size\_t idx2) // list\_command.c

idx1과 idx2의 인덱스를 가지는 list\_element를 찾고 이를 list\_swap함수를 이용해

swap한다.

2) void list\_swap(struct list\_elem \*a, struct list\_elem \*b) // list.c

list element a와 b를 리스트 내에서 swap한다.

서로 인접한경우와 인접하지 않은 경우를 나누어 swap을 처리한다.

3) void l\_shuffle(struct symbol\* symbol) // list\_command.c

list\_shuffle함수를 이용하여 리스트 내 원소들을 shuffle한다.

4) void list\_shuffle(struct list \*list) // list.c

uniform shuffle 방법으로 리스트를 셔플한다. 이때 시간복잡도는 O(n^2)이다

**-HASH TABLE**

1) unsigned hash\_int\_2(int i) //hash.c

knuth 가 제안한 방법에 의하여 정수 i의 해쉬를 반환한다.

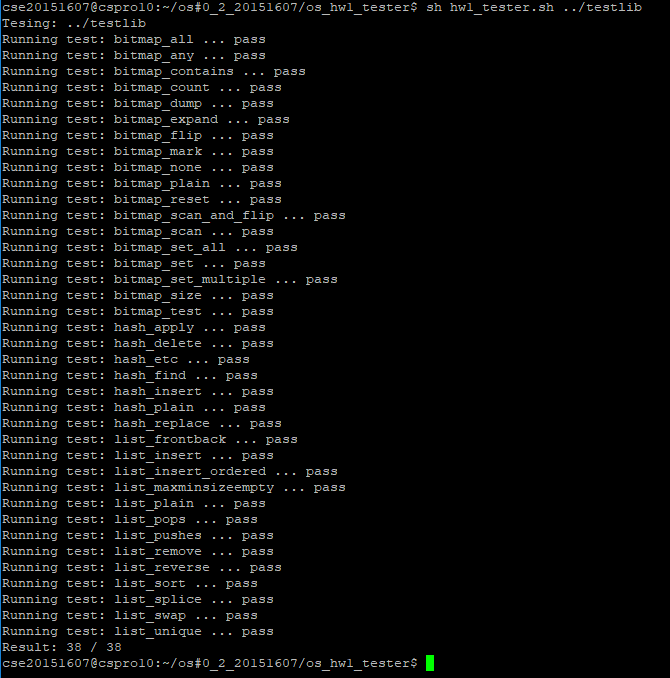
**-BITMAP**

1) int b\_expand(struct symbol\* symbol, int size) //bitmap\_command.c

해당 비트맵을 정보를 유지하면서 bitmap\_expand함수를 이용하여 확장한다.

2) struct bitmap\* bitmap\_expand(struct bitmap \*bitmap, int size) //bitmap.c

추가 사이즈만큼 늘려 새로운 비트배열을 할당하고 여기에 원 비트맵 정보를 복사한다.

1. 평가