OS 2 -ASSIGNMENT REPORT

PROGRAMMING ASSIGNMENT

-THUMATI UJJIEVE (CS16BTECH11039)

-P.RAMKISHAN(CS16BTECH11029)

DATE : 23-4-2018

UTILITIES USED : github, gdb,g++ compiler ,atom,sublime text .

CONTENTS

INTRODUCTION

Goal 1 : To create a simple slab allocator using our own library(libmymem.hpp)

Goal 2 : To make library Thread safe and Re-entarant.

ALL datatypes and functions of libmymem.hpp are disscussed below

USED : iostream , cstdlib , string.h , sys/mann.h

Functioning of mymalloc.

First mymalloc searches for the best fit bucket using the size passed as argument.After finding the bucket index it calls searchInBucket function which returns void PTR.

SearchInBucket function is thread safe as it uses user defined semaphore . This function searches through the slabs for free objects and if found updates the bitmap of the slab

and returns the memory pointer .If no slab with free objects is found then a new slab is created using createSlab function.

How objects are stored??

64KB chunk is given to the slab using mmap which returns pointer p. The objects are stored from the address (char*) p +sizeof(struct slab) .From this address objects are created in this free space and are linked to each other forming linked list.

Functioning of myfree:

Myfree function is thread safe as it uses user defined semaphores for critical section access.

Void * is passed which is used to get the object* by doing pointer arithmetic once object pointer is obtained slab pointer is also obtained. The slab pointer is used to update the bitmap and the freeObject variable.

Thread Safe :

For thread safety we are using locks using user defined semaphores .

Problems Faced:

There were lot of segmentation faults because of dereferencing of void * . The segmentation faults were recognized using gdb tool .

Another problem faced was the allocation of objects in the 64 KB chunk which was solved using pointer arithmetic.

CONSTANTS :

SLAB_SIZE

Stores the size of slab here it is 64*1024 Bytes .

N

Number of different Bucket sizes i.e Object sizes these can provide

Here its 12 . form 4 bytes to 8192 bytes increasing multiple of 2 .

DATA TYPES :

```
struct  Object{
        void* parentSlab = Null  ; //  stores slab address in which object is present
         void* memory = Null ;//        Stores address of free memory space
        Object * nextPointer = Null ;//Stores next object pointer in same slab
};
struct  Slab{
         Int  totolObj = 0 ; //  stores number of object in slab done at initialise time
       Int  freeObj = 0 ; //  stores  free number of object in slab done at initialise time
        Size_t bucketSize = 0 ; //  stores  bucket size  which is same as bucket it is
                                    //   attached  used to allocate correct object
       string  bitmap //  string of len  = totalObj , at free object its '0' at non free
                              //  position    it sotes '1'
       Object * objPtr = Null ;//Stores  first object pointer of slab
         Slab*   nextSlab = Null ;//  Stores address of  next slab in same bucket
        Object * objPtr = Null ;//Stores  first object pointer of slab
};
struct  Bucket{
        Size_t bucketSize ;      //       Stores size of bucket
        Slab  * slab = Null ;    //       Stores first slab address in bucket
};
```

FUNCTIONS :

FUNCTION NAME	PARAMETERS	RETURN VALUE	DESCRIPTION
firstZeroBitmap	std::String	int	return position of first zero in input string if present else returns -1 For finding position of zero it goes through string It takes O(n) time
make_n_length_string	Positive integer	std::string	It gives string of input length with all zeros in it Used in creating bitmap of slab objects
change _char_at	Integer i, string * a , string b	void	Replaces string b in position of integer i in strng a Eg : change_char_at(1,"001","1"); Changes "001" to "011"
update_parent_slab_in_Object	Slab * s , Object * o	void	Updates parent slab in passed object
get_parent_slab_address	Object * o	Slab *	Returns converted void pointer to Slab pointer in Object o
update _parent_bucket_address	Bucket * b , Slab * s	void	Updates bucket pointer in given slab with given bucket pointer.

allocate_slab_chunk		Void *	Returns slabSize chnck of

			memory Internally uses mmap
deallocate_slab_chuck	Void * p		Frees allocates slab which is allocated using allocate_slab_chunck Internally uses munmap()
initializeSlab	Slab *s , Bucket b	0	update s bucket pointer,bucket size in slabs Joins that bucket Updates bit map Divides remaining space from slab size in To Objects
intitialize_bucket	Int i	int	Updates bucket size Initialize with one slab in bucket

createSlab	Slab * s , Bucket b	Slab *	Allocates space to snew slab using allocate_slab_chunck Then intialise new slab Returns new slab address
initialize_all_Buckets	Int intex	0	Calls intiaze_bucket for that index
searchInBucket	Int bindex	Void *	Search in respective bucket according to given input for Free Object . if free object is not present then another new slab is joined. Returns memory value in that free Object sdfsd

mymalloc	unsigned s	void *	Allocates free memory of given size by using BEST FIT ALGORITHM in present buckets and Returns starting address of ALLOCATED MEMORY
myfree	Void * ptr	void	Frees the allocated memory Using mymalloc ,changes bit map in the slab its present Which helps to reuse memory

REFERENCES

1. www.gooogle.com
2. www.stackoverflow.com