M1522.000800 System Programming, Fall 2017

L2 Link Lab: Memtrace

Report

Name	Gao Zhiyuan
Student ID	2017-81842

Introduction

Part1-3 are bonus are implemented in this assignment to manipulate memory allocation details in the given form.

In part1-2, linklab/util/memlist.c and linklab/partn/memtrace.c and linklab/partn/memtrace.h are modified in order to realized items add, removing and block dumping. Details are provided in the following sections.

A modification of test6 is added with a “return 0;” at the end of the main function in order to avoid “makefile error 64”. The decision was made because due to the difference between various C versions, the default return value of main differs. It is abiding the standard to add return statement at the end of main function.

Part1

Illegal Free and Double Free is implemented in this part

As test4.c is suggesting a possibility that there might be ill free and double free function calls, I decided to implement ill free and double free log in this part.

I added a remove_from_list() function. Every time we call alloc(), a new item is added to the linked list. Meanwhile as the memory is freed, we should remove the item from the list. Thus, those on the list are always blocks which are not yet freed.

As for ill free, the input pointer is compared to every item’s pointer in the list. If there is no corresponding pointer, it means the address is not valid.

As for double free, every time we call free, we are expecting to find the handler in the list. If there’s no corresponding item, it means the handler has already been freed.

Calculate total byte and total free

Global variables are altered in my implementation.

```
static unsigned long n_allocb = 0;
static unsigned long n_alloc = 0;
static unsigned long n_freeb = 0;
static unsigned long n_free = 0;
```

The above variables are referring to allocated bytes, allocation called times, freed bytes and free called times. Those are recorded in order to calculate the total allocated and freed bytes and average ones.

I believe the above variables are more useful and save space compared with the variables provided in the skeleton.

remove_from_list() added

remove_from_list() is added in memlist.c to remove an item from the linked list.

As the above section states, removing items from the list is of vital importance to check availability of the memory blocks allocated right now.

Part2

The main task of part2 is to trace unfreed memory block and print them out.

List unfreed memory blocks

The problem is how to check if it has been freed or not.

First I thought about adding a flag field to the item struct so that every time I check the item, I could tell if it has been freed from the flag.

However checking flag takes significant effort in practice and there are many tricky chanllenges about add a field in the struct. So I used flag in the bonus part.

Thus I keep the list to be the collection of all unfreed memory blocks all the time. Everytime we free a block, we remove it from the list, so that finally we could just print all the list.

As a consequence, another function in memtrace called dump_nonfreed() is added and it simply dumps all the items left in the list. It is simple to implement so I will not show it here.

Reallocation is both deallocation and callocating

realloc is first freeing the old item, then adding a new item with the updated size to the list. Thus, the calculation of freed bytes and allocated bytes both increase.

I implemented the function by simply remove the previous item from the list and add a new item to the list.

freed bytes

I made a global variable to store the bytes we free. Every time we call a free function, if it is not illegal or double free, we would add the free size to the global variable.

Part3

In this part, we are backtracing the caller to know their function name and offset using libunwind library.

makefile error 64

I encoutered this error because test6.c has no return statement in main function. In c99 the default return value of main is 0 but it is not in c89 or c90. Thus, to gain a better compatiblilty, I added “return 0;” to the end of main function in test6.c.

backtracing

The first problem is how many time should we trace. The get_callinfo() is called twice each time we malloc, the first time it is called by mlog(), the second time it is called by malloc()/calloc() /realloc().

Fortunately, all of them only requires 3 times of backtracing.

Then we could utilize function unw_get_proc_name() to get function name and offset fromt the destination stack frame.

Bonus

In the bonus part, I enabled illegal free and double free checking up adding a flag field in every item.

flag

A flag is added in the struct. Everytime alloc() is called, the flag is set to 1. While it is successfully freed, the flag is set to 0.

To check illegal free, we look for a corresponding item with the same pointer in free function. If

there is not, then it is an illegal free because we are trying to free a block of memory which has not yet been allocated.

To check double free, we think the item is double freed if it is passed in the free function with flag equals 0.

Setting a flag is obviously powerful as it saves time significant. But it means we cannot remove the items from the list so there are a number of functions we need to change, which might be time consuming for part3.

Experience using Git

Git should be powerful especially when there is an emergency such as a power cut.

On Sunday, I could not log into my system programming acount at the software lab computers, in which there is all of my project progress. I wish I had pushed them onto the git. So that I could check and commit after the power cut.

Everytime I try to rebase or merge, it will be a disaster. Sometimes I just delete the whole repository and clone it from another version.

Writing a useful commit message is of vital importance. It should be simple and clear and most importantly, constaining the necessary information.