ECEN5013

HOMEWORK2

Mounika Reddy Edula | 09/19/2017

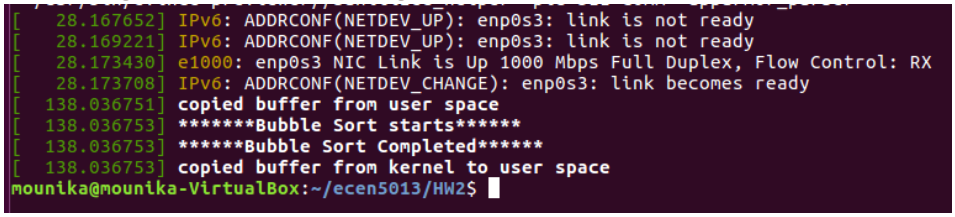
PROBLEM2

How to create system Call?

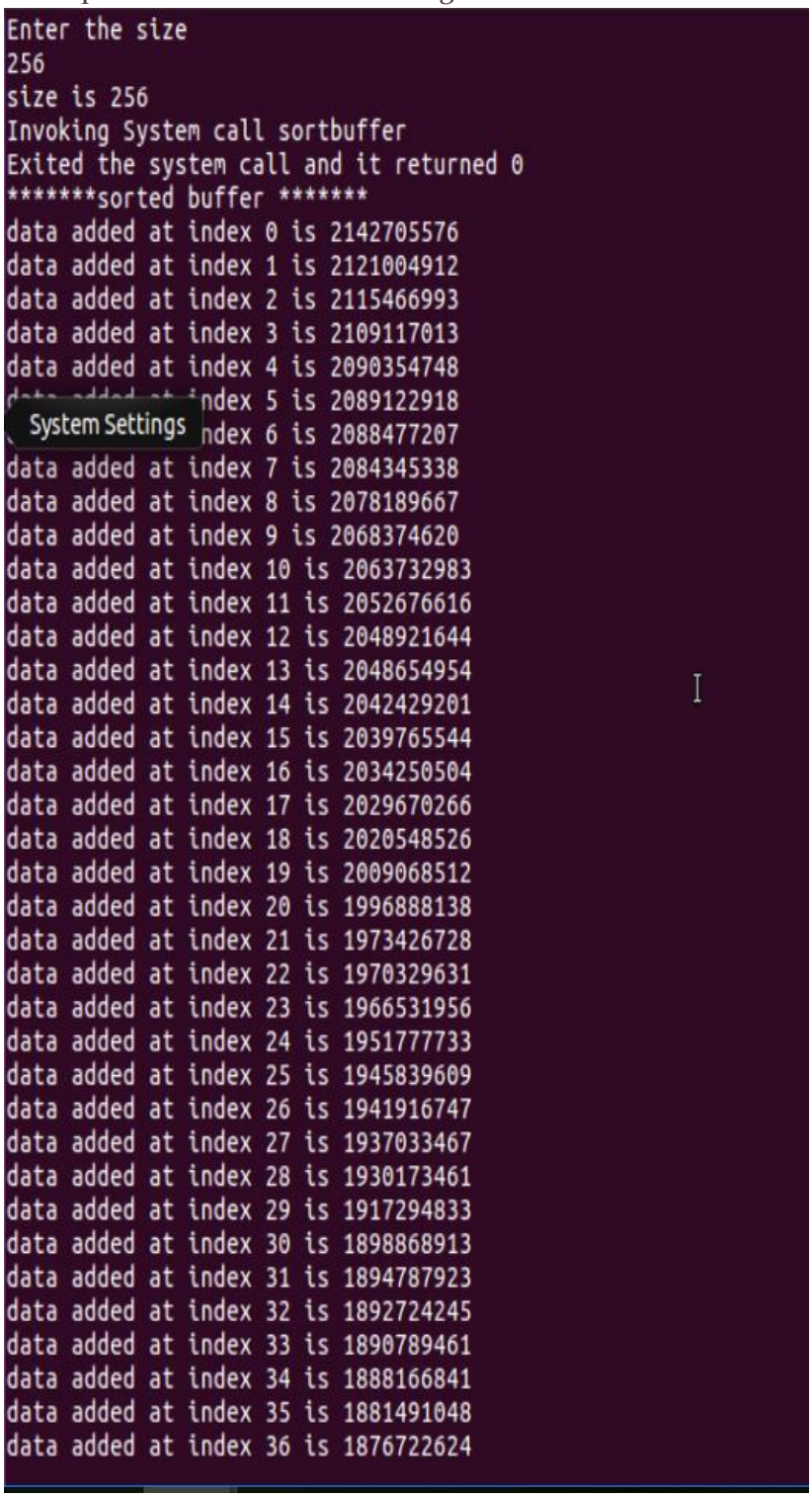
1. Create a new directory in the linux folder
2. Create your syscall
    - SYSCALL_DEFINEn(var args) - n is the number of arguments sys call will take this is just a macro which will be converted to asmlinkage function definition.
    - If you define in by SYSCALL_DEFINEn then we don't need to add the function definition in the syscalls.h
    - Add the syscall to the syscall_64.tbl as the user space searches for the syscall here and maps to the function in the kernel space.
    - Create a makefile to create the object file your syscall
      Core-y: example.o
    - In the makefile of your linux directory add your directory for the build
      Ifeq($(KBUILD_EXTMOD)
      Core-y += directory/:
    - Build your kernel and reboot
3. Now your syscall will be copied to the linux directory and is ready to use
4. Now you can use the syscall for sorting buffer we can invoke syscall with Syscall(333,arguments) 333 – syscall number in the systable. The arguments are separated by , in between datatype and name
   Note: validate the parameters in the syscall as we have access to all memory regions and log any error into the kernel log

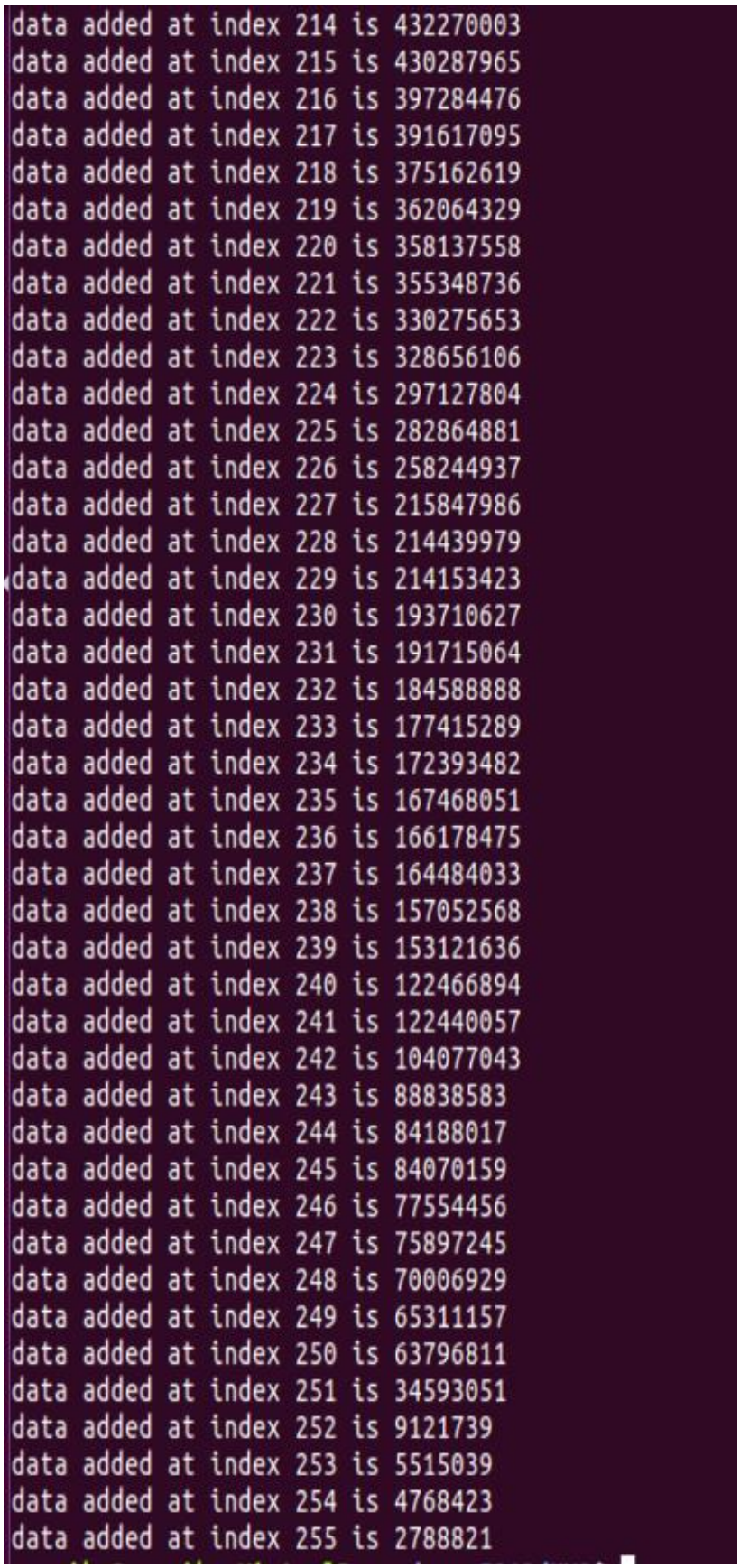
I have checked the validation of the input parameters when they are NULL. The check for the permissions of the memory will be done in the copy_to_user().copy_from_user() implements the validation for that through access_OK() syscall
It uses access_ok() to check the user space 'from' pointer for 'n' Bytes and if it's within the address limit it will continue to the actual copy through __copy_from_user(). If this fails, it will just zero out the kernel memory pointed by 'to' pointer and return.

Output -
Kernel log for successful sorting

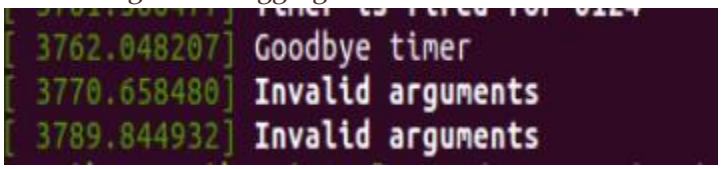


User space after a successful sorting

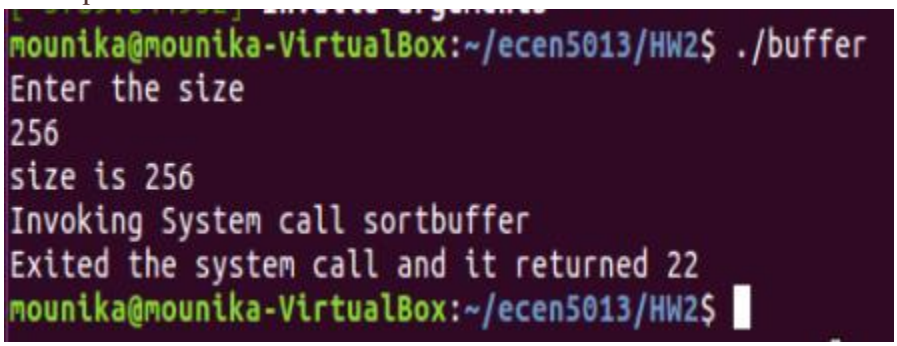



data added at index 214 is 432270003
data added at index 215 is 430287965
data added at index 216 is 397284476
data added at index 217 is 391617095
data added at index 218 is 375162619
data added at index 219 is 362064329
data added at index 220 is 358137558
data added at index 221 is 355348736
data added at index 222 is 330275653
data added at index 223 is 328656106
data added at index 224 is 297127804
data added at index 225 is 282864881
data added at index 226 is 258244937
data added at index 227 is 215847986
data added at index 228 is 214439979
data added at index 229 is 214153423
data added at index 230 is 193710627
data added at index 231 is 191715064
data added at index 232 is 184588888
data added at index 233 is 177415289
data added at index 234 is 172393482
data added at index 235 is 167468051
data added at index 236 is 166178475
data added at index 237 is 164484033
data added at index 238 is 157052568
data added at index 239 is 153121636
data added at index 240 is 122466894
data added at index 241 is 122440057
data added at index 242 is 104077043
data added at index 243 is 88838583
data added at index 244 is 84188017
data added at index 245 is 84070159
data added at index 246 is 77554456
data added at index 247 is 75897245
data added at index 248 is 70006929
data added at index 249 is 65311157
data added at index 250 is 63796811
data added at index 251 is 34593051
data added at index 252 is 9121739
data added at index 253 is 5515039
data added at index 254 is 4768423
data added at index 255 is 2788821

When a null pointer is passed to syscall
Kernel Log for debugging



User space return error number



PROBLEM 3

Create a kernel Module

Kernel Modules can be loaded externally or internally. But the question requires a run time loadable kernel module. So, created an external kernel module

1. Create a new directory or create inside previously present folders like char/ Block modules
2. I created a new directory and add source code for the module
   - The system enters a module through a module_init() and exits through Module_exit()
   - The information displayed in the modinfo is included through MODULE_LICENSE(),MODULE_VERSION,MODULE_AUTHOR,MODULE_DESCRIPTION.
   - Create a makefile which will build the module
3. Installing module.
   - Compile the module and generate .ko files and copy that to the source of linux.
   - Sudo insmod module.ko
   - Update the modular dependencies sudo dep mode -a

Output
Lsmod |grep mytime

```
mounika@mounika-VirtualBox:~/ecen5013/linux-4.12.10/drivers/mytime$ lsmod |grep mytime
mytime                 16384  0
mounika@mounika-VirtualBox:~/ecen5013/linux-4.12.10/drivers/mytime$
```

Modinfo mytime

```
mounika@mounika-VirtualBox:~/ecen5013/linux-4.12.10/drivers/mytime$ sudo rmmod mytime
mounika@mounika-VirtualBox:~/ecen5013/linux-4.12.10/drivers/mytime$ install_module
filename:       /lib/modules/4.12.10/kernel/drivers/mytime/mytime.ko
version:        0.1
description:    Trigger a timer every 500ms
author:         Mounika Reddy Edula
license:        GPL
srcversion:     BD3592DE4E0C3E86E9C8C0F
depends:
vermagic:       4.12.10 SMP mod_unload modversions
lsmod of mytime
mytime                 16384  0
mounika@mounika-VirtualBox:~/ecen5013/linux-4.12.10/drivers/mytime$
```

Kernel Log dmesg

```
[ 5117.318242] Loading timer module ....
[ 5117.318243] Just a timer
[ 5117.519052] Timer is fired for 1
[ 5118.026971] Timer is fired for 2
[ 5118.538972] Timer is fired for 3
[ 5119.051003] Timer is fired for 4
[ 5119.562892] Timer is fired for 5
[ 5120.074930] Timer is fired for 6
[ 5120.586912] Timer is fired for 7
[ 5121.098785] Timer is fired for 8
[ 5121.610762] Timer is fired for 9
[ 5122.122760] Timer is fired for 10
[ 5122.635522] Timer is fired for 11
```

Remove the module

```
[ 5192.778502] Timer is fired for 148
[ 5193.291057] Timer is fired for 149
[ 5193.333233] Goodbye timer
```

Using the Syscall

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <time.h>
#include <stdint.h>
/*************************************************************************
*@Filename:sort_buffer.c
*
*@Description:Utilising the syscall created for sorting the buffer
*
*@Author:Mounika Reddy Edula
*@Date:09/19/2017
*compiler:gcc
*debugger:gdb
*linux:linux-4.12.10
*************************************************************************/
#define sys_sortbuffer 333



int main()
{
int32_t data =0;
time_t t = 0;
int32_t* buffer = NULL;
int32_t* temp_buffer = NULL;
int32_t* sort_buffer = NULL;
int ret_status = 0;
srand(time(&t));
size_t size = 0;


//Create a random buffer with user specs
printf("Enter the size\n");
scanf("%lu",&size);
printf("size is %lu\n",size);
buffer = malloc(size*sizeof(int32_t));
sort_buffer = malloc(size*sizeof(int32_t));
temp_buffer = buffer;
```

```
if(buffer == NULL)
{
printf("memory not allocated\n");
}


for(int i=0;i<size;i++)
{
 data = rand();
 //printf("data added at index %d is %d\n",i,data);
 *temp_buffer++ = data;
}


//Invoking the system call
printf("Invoking System call sortbuffer\n");
ret_status = syscall(sys_sortbuffer,buffer,size,sort_buffer);
printf("Exited the system call and it returned %d\n",ret_status);


//print the output


if(ret_status ==0)
{
printf("*******sorted buffer *******\n");
for(int i=0;i<size;i++)
{
 printf("data added at index %d is %d\n",i,*sort_buffer++);
}
}
}
```

Syscall

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <uapi/asm-generic/errno-base.h>
#include <linux/uaccess.h>
#include <linux/gfp.h>


/*********************************************************************
****
*@Filename:sortbuffer.c
*
*@Description:Implementation of syscall for sorting buffer
*
*@param buffer,size of buffer,pointer to sorted buffer
*@return enum Error numbers look into errno.h,errno-base.h
*
*@Author: Mounika Reddy Edula
*@Date:09/18/2017
*@Compiler:gcc
*@Debugger:gdb
*@Kernel version:4.12.10
*********************************************************************
**/


/*Syscall macro will be converted to asmlinkage as in the header
syscalls.h*/
SYSCALL_DEFINE3(sortbuffer, int32_t*, buffer, uint32_t, size, int32_t*,
sort_buffer)
{


 /***intialise local variables validate inputs****/
 int32_t* k_buffer = NULL;
 int32_t swap = 0;
 int status=0;
 int i=0;
 int j=0;
 if(buffer == NULL || sort_buffer == NULL || size == 0)
```

```
{
printk("Invalid arguments\n");
return EINVAL ;
}
k_buffer = (int32_t *)kmalloc( sizeof(int32_t)*size,GFP_KERNEL);
if(k_buffer == NULL)
{
printk("memory allocation failed\n");
return ENOMEM;
}
status = copy_from_user(k_buffer,buffer,sizeof(int32_t)*size);
if(status < 0)
{
printk("can't copy from user space\n");
return EFAULT;
}
printk("copied buffer from user space\n");
/***Bubble sort for sorting***/
printk("*******Bubble Sort starts******\n");
for(i=1;i<size;i++)
{
 for(j=0;j<size-i;j++)
 {
  if(*(k_buffer+j)<*(k_buffer+j+1))
  {
  swap = *(k_buffer+j);
  *(k_buffer+j) = *(k_buffer+j+1);
  *(k_buffer+j+1) = swap;
  }
 }
}
printk("******Bubble Sort Completed******\n");
status = copy_to_user(sort_buffer,k_buffer,sizeof(int32_t)*size);
if(status < 0)
{
printk("can't copy to user\n");
return EFAULT;
}
printk("copied buffer from kernel to user space\n");
return 0;
}
```

Kernel Module

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/timer.h>


/*********************************************************************
*@Filename:mytime.c
*
*@Description:Implemented a module which is triggered every 500ms
*
*@Author:Mounika Reddy Edula
*@Date:09/19/2017
*@compiler:GCC
*@Debugger:GDB
*@kernel:4.12.10
*********************************************************************/
//parameters for modinfo
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Mounika Reddy Edula");
MODULE_DESCRIPTION("Trigger a timer every 500ms");
MODULE_VERSION("0.1");


//static as the count value should be stored
static struct timer_list my_timer;
static int32_t count;


//callback when timer expires
void my_timer_callback(unsigned long data)
{
count++;
printk("Timer is fired for %d\n",count);
my_timer.expires = jiffies+msecs_to_jiffies(500);
add_timer(&my_timer);
}


//The module is started with init function
static int __init timer_start(void)
{
```

```
printk(KERN_INFO "Loading timer module ....\n");
printk(KERN_INFO "Just a timer\n");
setup_timer(&my_timer,my_timer_callback,0);
mod_timer(&my_timer, jiffies+msecs_to_jiffies(200));
return 0;
}


//The module is exited with __exit function
static void __exit timer_end(void)
{
del_timer(&my_timer);
printk(KERN_INFO "Goodbye timer\n");
}


module_init(timer_start);
module_exit(timer_end);
```

Install module

```
#! /usr/bin/python
import os


os.system("sudo insmod /home/mounika/ecen5013/linux-
4.12.10/drivers/mytime/mytime.ko");
os.system("sudo cp -r /home/mounika/ecen5013/linux-4.12.10/drivers/mytime
/lib/modules/4.12.10/kernel/drivers/")
os.system("cd /home/mounika/ecen5013/linux-4.12.10/drivers/mytime/ && sudo
depmod -a")
os.system("modinfo mytime")
print "lsmod of mytime"
os.system("lsmod | grep mytime")
```

Test cases

Test_circbuffer.c

```
#include <stdarg.h>
#include <stdlib.h>
#include <stddef.h>
#include <setjmp.h>
#include <stdint.h>
#include <cmocka.h>
#include <stdbool.h>
#include "circ_buffer.h"


/**************************************************************************
*******
*@Filename:test_circbuffer.c
*
*@Description: Test cases using cmocka to test the boundary conditions for
circular buffer
*pass null pointers, loop back,removing elements,destroy
*
*@Author:Mounika Reddy Edula
*@Date: 09/19/2017
*@Test Framework: CMocka
**************************************************************************
********/


/*Test the add_item functionality of circular buffer*/
void test_add_item(void **state)
{
    Buffer* test_buffer = NULL;
    int32_t data = 30982;
    int return_value;
    //test if buffer accessed without buffer being allocated
    return_value = add(test_buffer,&data);
    assert_int_equal(return_value,error);
    //test if data is not NULL pointer
    data = NULL;
    return_value = add(test_buffer,&data);
    assert_int_equal(return_value,error);
    //Buffer allocation
    return_value = allocate(&test_buffer);
    assert_int_equal(return_value,Success);
```

```
    //test to add item after buffer allocation
    return_value = add(test_buffer,&data);
    assert_int_equal(return_value,Success);
    //test to see if data is overwritten
    data = 2000;
    add(test_buffer,&data);
    data = -1000;
    add(test_buffer,&data);
    data = 500;
    add(test_buffer,&data);
    data = 60000;
    add(test_buffer,&data);
    data = 50;
    return_value = add(test_buffer,&data);
    assert_int_equal(return_value,Fail);
    //test to see if buffer is looped back
    remove_item(test_buffer);
    data = 100;
    return_value = add(test_buffer,&data);
    assert_int_equal(return_value,Success);
    destroy(test_buffer);
    test_buffer = NULL;
    printf("***1.add item test passed*****\n");
}


/*Test the remove_item functionality of circular Buffer*/
void test_remove_item(void **state)
{
    Buffer* test_buffer = NULL;
    int return_value;
    int32_t data;
    //test when buffer is not allocated and item is removed
    return_value = remove_item(test_buffer);
    assert_int_equal(return_value,error);
    //remove item when buffer is empty
    allocate(&test_buffer);
    return_value = remove_item(test_buffer);
    assert_int_equal(return_value,Fail);
    //add item and then remove
    data = 2000;
    add(test_buffer,&data);
    return_value = remove_item(test_buffer);
```

```
    assert_int_equal(return_value,Success);
    data = -1000;
    add(test_buffer,&data);
    data = 500;
    add(test_buffer,&data);
    data = 60000;
    add(test_buffer,&data);
    data = 50;
    add(test_buffer,&data);
    //remove all items in the buffer
    return_value = remove_item(test_buffer);
    //see if tail is looped back to remove the item
    return_value = remove_item(test_buffer);
    assert_int_equal(return_value,Success);
    return_value = remove_item(test_buffer);
    assert_int_equal(return_value,Success);
    remove_item(test_buffer);
    //when buffer is empty
    return_value = remove_item(test_buffer);
    assert_int_equal(return_value,Fail);
    destroy(test_buffer);
    test_buffer = NULL;
    printf("***2. delete item test passed****\n");
}


/*Test the Is_buffer_full functionality of circular Buffer*/
void test_is_buffer_full(void **state)
{
    Buffer* test_buffer = NULL;
    int32_t data;
    bool value;
    //test if buffer is full when it is empty
    allocate(&test_buffer);
    //test if buffer is full when item is added
    data = 2000;
    add(test_buffer,&data);
    data = -1000;
    add(test_buffer,&data);
    assert_false(Is_buffer_full(test_buffer));
    data = 500;
    add(test_buffer,&data);
    data = 60000;
```

```
    add(test_buffer,&data);
    data = 40;
    add(test_buffer,&data);
    assert_true(Is_buffer_full(test_buffer));
    //test if buffer is full after removing an item
    remove_item(test_buffer);
    printf("size after removing an element is %d\n",size(test_buffer));
    data = 40;
    add(test_buffer,&data);
    data = 40;
    add(test_buffer,&data);
    assert_false(Is_buffer_full(test_buffer));
    destroy(test_buffer);
    test_buffer = NULL;
    printf("***3. Is buffer full test passed*****\n");
}


/*Test Is_Buffer_empty functionality of Circular Buffer*/
void test_is_buffer_empty(void **state)
{
    Buffer* test_buffer = NULL;
    int return_value = 0;
    int32_t data;
    //buffer empty after allocation
    allocate(&test_buffer);
    assert_true(Is_buffer_empty(test_buffer));
    //add item and check for buffer empty
    data = 2000;
    add(test_buffer,&data);
    assert_false(Is_buffer_empty(test_buffer));
    data = -1000;
    add(test_buffer,&data);
    data = 500;
    add(test_buffer,&data);
    data = 60000;
    add(test_buffer,&data);
    data = 50;
    add(test_buffer,&data);
    //remove all items in the buffer
    remove_item(test_buffer);
    remove_item(test_buffer);
    remove_item(test_buffer);
```

```
    remove_item(test_buffer);
    //when buffer is empty
    remove_item(test_buffer);
    assert_true(Is_buffer_empty(test_buffer));
    destroy(test_buffer);
    test_buffer = NULL;
    printf("******4. Is buffer empty test passed*****\n");
}


/*Test the size() functionality of circular Buffer*/
void test_size(void **state)
{
    Buffer* test_buffer = NULL;
    int return_value;
    int32_t data;
    //validate pointer
    return_value = size(test_buffer);
    assert_int_equal(return_value,error);
    //check size after allocation
    allocate(&test_buffer);
    return_value = size(test_buffer);
    assert_int_equal(return_value,0);
    //add item and check size
    data = 2000;
    add(test_buffer,&data);
    return_value = size(test_buffer);
    assert_int_equal(return_value,1);
    //remove item and check size
    remove_item(test_buffer);
    return_value = size(test_buffer);
    assert_int_equal(return_value,0);
    //check size after buffer loopback
    data = -1000;
    add(test_buffer,&data);
    return_value = size(test_buffer);
    assert_int_equal(return_value,1);
    data = 500;
    add(test_buffer,&data);
    return_value = size(test_buffer);
    assert_int_equal(return_value,2);
    data = 60000;
    add(test_buffer,&data);
```

```
    return_value = size(test_buffer);
    assert_int_equal(return_value,3);
    data = 50;
    add(test_buffer,&data);
    return_value = size(test_buffer);
    assert_int_equal(return_value,4);
    data = 90;
    add(test_buffer,&data);
    return_value = size(test_buffer);
    assert_int_equal(return_value,5);
    data = 100;
    add(test_buffer,&data);
    return_value = size(test_buffer);
    assert_int_equal(return_value,5);
    destroy(test_buffer);
test_buffer = NULL;
printf("***5. size test passed*****\n");
}
```

```
int main(int argc,char **argv)
{
const struct CMUnitTest test[] = {
       cmocka_unit_test(test_add_item),
       cmocka_unit_test(test_remove_item),
       cmocka_unit_test(test_is_buffer_full),
       cmocka_unit_test(test_is_buffer_empty),
       cmocka_unit_test(test_size)
       };


return cmocka_run_group_tests(test,NULL,NULL);
}
```

Test Case

Test_doublell.c

```
#include <stdarg.h>
#include <stdlib.h>
#include <stddef.h>
#include <setjmp.h>
#include <stdint.h>
#include <cmocka.h>
#include <stdbool.h>
#include "double_ll.h"


/************************************************************************
*@Filename:test_doublell.c
*
*@Description: Test the Double linked list with all boundary conditions
*like adding a new node at the start,end,middle.removing them in the start
*end,middle,number of items
*
*@Author:Mounika Reddy Edula
*@Date:09/19/2017
*@Test Framework:Cmocka
************************************************************************/


/*test add node functionality of double linked list*/
void test_add_node(void **state)
{
        struct node* test_head=NULL;
        int return_value;
        uint32_t data;
        //test if you try to remove node on empty list
        uint8_t *removed_data = malloc(sizeof(uint8_t));
        return_value = remove_node(&test_head,removed_data,1);
        assert_int_equal(return_value,Fail);
        //add a node
        data = 20;
        return_value = add_node(&test_head,&data,1);
        assert_int_equal(return_value,Success);
        //add nodes sequentially
        data = 30;
        return_value = add_node(&test_head,&data,2);
        assert_int_equal(return_value,Success);
```

```
        data = 40;
        return_value = add_node(&test_head,&data,3);
        assert_int_equal(return_value,Success);
        data =50;
        //add node in the start
        return_value = add_node(&test_head,&data,1);
        assert_int_equal(return_value,Success);
        data =80;
        //add node in the start
        return_value = add_node(&test_head,&data,1);
        assert_int_equal(return_value,Success);
        data = 60;
        //add node in the middle
        return_value = add_node(&test_head,&data,2);
        assert_int_equal(return_value,Success);
        data = 70;
        return_value = add_node(&test_head,&data,3);
        assert_int_equal(return_value,Success);
        destroy(&test_head);
        test_head = NULL;
        printf("***1.test to add node passed*****\n");
}


/*test remove node functionality of double linked list*/
void test_remove_node(void **state)
{
        struct node* test_head=NULL;
        struct node* remove_head=NULL;
        int return_value;
        int32_t data;
        //test when node is not allocated and item is removed
        data = 20;
        add_node(&test_head,&data,1);
        //add nodes sequentially
        data = 30;
        add_node(&test_head,&data,2);
        data = 40;
        add_node(&test_head,&data,3);
        //add item and then remove and validate data removed
        data = 2000;
        add_node(&test_head,&data,4);
        uint16_t *removed_data = malloc(sizeof(uint8_t));
```

```
        return_value = remove_node(&test_head,removed_data,1);
        assert_int_equal(*removed_data,20);
        assert_int_equal(return_value,Success);
        //remove data out of index
        return_value = remove_node(&test_head,removed_data,5);
        assert_int_equal(return_value,Fail);

        //validate the parameters
        return_value = remove_node(&remove_head,removed_data,2);
        assert_int_equal(return_value,Fail);
        return_value = remove_node(&test_head,NULL,3);
        assert_int_equal(return_value,Fail);

        //remove data in the middle
        return_value = remove_node(&test_head,removed_data,3);
        assert_int_equal(*removed_data,2000);
        assert_int_equal(return_value,Success);
        //after remove try to access out of range
        return_value = remove_node(&test_head,removed_data,4);
        assert_int_equal(return_value,Fail);
        //remove first node
        return_value = remove_node(&test_head,removed_data,1);
        assert_int_equal(*removed_data,30);
        destroy(&test_head);
        printf("***2. delete item test passed****\n");
}



/*Test the size functionality of double linked list*/
void test_size(void **state)
{
        struct node* test_head=NULL;
        int return_value;
        int32_t data;
        //validate the parameters
        return_value = size(&test_head);
        assert_int_equal(return_value,0);
        //add node and check size
        data = 20;
        add_node(&test_head,&data,1);
        data = 30;
```

```
        add_node(&test_head,&data,2);
        data = 40;
        add_node(&test_head,&data,3);
        //remove and check size
        data = 2000;
        add_node(&test_head,&data,4);
        return_value = size(&test_head);
        assert_int_equal(return_value,4);
        remove_node(&test_head,&data,3);
        return_value = size(&test_head);
        assert_int_equal(return_value,3);
        destroy(&test_head);
        test_head = NULL;
        printf("***5. size test passed*****\n");
}


/*Test the search functionality of double linked list*/
void test_search(void **state)
{
        struct node* test_head=NULL;
        int32_t data;
        uint8_t *index=NULL;
        uint8_t *error=malloc(sizeof(uint8_t));;
        *error =0;
        //validate the parameters
        index = search(&test_head,&data);
        assert_int_equal(*index,*error);
        index = search(&test_head,NULL);
        assert_int_equal(*index,*error);
        //add nodes
        data = 20;
        add_node(&test_head,&data,1);
        data = 30;
        add_node(&test_head,&data,2);
        data = 40;
        add_node(&test_head,&data,3);
        data = 2000;
        add_node(&test_head,&data,4);
        data = 20;
        //search for data and validate the index
        index = search(&test_head,&data);
        assert_int_equal(*index,1);
```

```
        data = 2000;
        index = search(&test_head,&data);
        assert_int_equal(*index,4);
        data = 40;
        index = search(&test_head,&data);
        assert_int_equal(*index,3);


        //remove a node and access the data
        remove_node(&test_head,&data,3);
        data = 40;
        index = search(&test_head,&data);
        assert_int_equal(*index,0);
        destroy(&test_head);
        test_head = NULL;
        printf("***5. search test passed*****\n");
}


/*test destroy functionality of double linked list*/
void test_destroy(void **state)
{
        struct node* test_head=NULL;
        int return_value;
        uint8_t data;
        //validate the parametrs
        return_value = destroy(&test_head);
        assert_int_equal(return_value,Fail);
        data = 20;
        //add a node and destroy the linked list
        add_node(&test_head,&data,1);
        return_value = destroy(&test_head);
        assert_int_equal(return_value,Success);
}


int main(int argc,char **argv)
{
const struct CMUnitTest test[] = {
       cmocka_unit_test(test_add_node),
       cmocka_unit_test(test_remove_node),
       cmocka_unit_test(test_size),
       cmocka_unit_test(test_search),
```

```
        cmocka_unit_test(test_destroy)
        };


return cmocka_run_group_tests(test,NULL,NULL);
}
```