성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

[REPORT]

■과 목 명: 운영체제 (SWE3004-42)
■담 당 교 수: 신동군 교수님
■제 출 일: 2022년 3월 24일
■학 과: 수학과
■학 번: 2016314786
■성 명: 김호진

Chap 14. Debugging Memory Allocation w/ *valgrind*

수학과 김호진 (2016314786)

1. First, write a simple program called null.c **that creates a pointer to an integer, sets it to** NULL**, and then tries to dereference it. Compile this into an executable called** null**. What happens when you run this program?**

문제의 요구사항을 만족하는 프로그램의 소스코드는 다음과 같다.

home > borussen > C null.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int *ptr = NULL;
    printf("%d\n", *ptr);  // deference the NULL pointer
    return 0;
}
```

이 프로그램을 실행하면 다음과 같이 Segmentation fault가 발생한다.

```
borussen@DESKTOP-L834KLC:~$ gcc -g null.c -o null
borussen@DESKTOP-L834KLC:~$ ./null
Segmentation fault
```

2. Next, compile this program with symbol information included (with the -g flag). Doing so let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null **and then, once** gdb **is running, typing** run**. What does gdb show you?**

gdb는 허용되지 않은 메모리 액세스에 대한 오류를 의미하는 SIGSEGV를 사용자에게 보낸다.

```
(gdb) run
Starting program: /home/borussen/null

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555168 in main (argc=1, argv=0x7fffffffdee8) at null.c:7
7           printf("%d\n", *ptr);  // deference the NULL pointer
```

3. Finally, use the valgrind **tool on this program. We'll use the** memcheck **tool that is a part of** valgrind **to analyze what happens. Run this by typing in the following:** valgrind --leak-check=yes null**. What happens when you run this? Can you interpret the output from the tool?**

valgrind를 통해 C/C++ 프로그램에서 발생하는 메모리 누수(Memory Leak) 등을 탐지할 수 있다. 해당 프로그램에서 valgrind를 실행하면, 다음과 같이 Invalid read로 인해 나타나는 오류를 발견하여 알려준다. 해당 출력내용을 해석해보면 null.c의 7번째 줄에서 할당되지 않은 null pointer를 deference하려고 했기 때문에 Invalid read로 인한 Segmentation fault가 발생했음을 알 수 있다.

```
borussen@DESKTOP-L834KLC:~$ valgrind --leak-check=yes ./null
==1840== Memcheck, a memory error detector
==1840== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1840== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright inf
==1840== Command: ./null
==1840==
==1840== Invalid read of size 4
==1840==    at 0x109168: main (null.c:7)
==1840==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==1840==
==1840==
==1840== Process terminating with default action of signal 11 (SIGSEGV)
==1840==  Access not within mapped region at address 0x0
==1840==    at 0x109168: main (null.c:7)
==1840==  If you believe this happened as a result of a stack
==1840==  overflow in your program's main thread (unlikely but
==1840==  possible), you can try to increase the size of the
==1840==  main thread stack using the --main-stacksize= flag.
==1840==  The main thread stack size used in this run was 8388608.
==1840==
==1840== HEAP SUMMARY:
==1840==     in use at exit: 0 bytes in 0 blocks
==1840==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==1840==
==1840== All heap blocks were freed -- no leaks are possible
==1840==
==1840== For lists of detected and suppressed errors, rerun with: -s
==1840== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
```

4. Write a simple program that allocates memory using malloc() **but forgets to free it before exiting. What happens when this program runs? Can you use** gdb **to find any problems with it? How about** valgrind **(again with the** --leak-check=yes **flag)?**

문제의 요구사항을 만족하는 프로그램의 소스코드는 다음과 같다.

home > borussen > C forget_free.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int *ptr = (int *) malloc(sizeof(int));
    *ptr = 1;
    printf("%d\n", *ptr);
    return 0;
}
```

이 프로그램은 정상적으로 컴파일 되고 실행된다.

```
borussen@DESKTOP-L834KLC:~$ gcc -g forget_free.c -o forget_free
borussen@DESKTOP-L834KLC:~$ ./forget_free
1
```

gdb 역시 해당 프로그램에서 어떠한 문제도 발생하지 않았다고 알린다.

```
(gdb) run
Starting program: /home/borussen/forget_free
1
[Inferior 1 (process 2286) exited normally]
```

하지만 valgrind는 heap 영역에 할당된 메모리가 free되지 않으면서 발생한 메모리 누수(memory leak)를 탐지하고 이를 사용자에게 보고한다.

```
borussen@DESKTOP-L834KLC:~$ valgrind --leak-check=yes ./forget_free
==2345== Memcheck, a memory error detector
==2345== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2345== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2345== Command: ./forget_free
==2345==
1
==2345==
==2345== HEAP SUMMARY:
==2345==     in use at exit: 4 bytes in 1 blocks
==2345==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==2345==
==2345== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2345==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==2345==    by 0x109185: main (forget_free.c:6)
==2345==
==2345== LEAK SUMMARY:
==2345==    definitely lost: 4 bytes in 1 blocks
==2345==    indirectly lost: 0 bytes in 0 blocks
==2345==      possibly lost: 0 bytes in 0 blocks
==2345==    still reachable: 0 bytes in 0 blocks
==2345==         suppressed: 0 bytes in 0 blocks
==2345==
==2345== For lists of detected and suppressed errors, rerun with: -s
==2345== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

5. Write a program that creates an array of integers called data **of size 100 using malloc; then, set** data[100] **to zero. What happens when you run this program? What happens when you run this program using** valgrind**? Is the program correct?**

문제의 요구사항을 만족하는 프로그램의 소스코드는 다음과 같다.

home > borussen > C data.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int *data = (int *) malloc(100);
    data[100] = 0;
    return 0;
}
```

해당 프로그램은 컴파일 되고 실행되는 데 있어 정상적으로 작동한다.

하지만 valgrind를 이용하여 프로그램을 실행하면, Invalid write와 heap 영역에서 메모리 누수가 발생하였다는 것을 알 수 있다.

```
borussen@DESKTOP-L834KLC:~$ gcc -g data.c -o data
borussen@DESKTOP-L834KLC:~$ ./data
borussen@DESKTOP-L834KLC:~$ valgrind --leak-check=yes ./data
==2539== Memcheck, a memory error detector
==2539== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2539== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2539== Command: ./data
==2539==
==2539== Invalid write of size 4
==2539==    at 0x109174: main (data.c:7)
==2539==  Address 0x4a471d0 is 224 bytes inside an unallocated block of size 4,194,032 in arena "client"
==2539==
==2539==
==2539== HEAP SUMMARY:
==2539==     in use at exit: 100 bytes in 1 blocks
==2539==   total heap usage: 1 allocs, 0 frees, 100 bytes allocated
==2539==
==2539== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2539==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==2539==    by 0x109165: main (data.c:6)
==2539==
==2539== LEAK SUMMARY:
==2539==    definitely lost: 100 bytes in 1 blocks
==2539==    indirectly lost: 0 bytes in 0 blocks
==2539==      possibly lost: 0 bytes in 0 blocks
==2539==    still reachable: 0 bytes in 0 blocks
==2539==         suppressed: 0 bytes in 0 blocks
==2539==
==2539== For lists of detected and suppressed errors, rerun with: -s
==2539== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

sizeof(int) = 4이므로 data는 25개의 elements만을 갖거나, 100 * 4 = 400 size를 할당 받아야 한다. 또한, malloc을 사용해 할당된 메모리는 heap 영역에 위치하기 때문에 프로그램 종료 전 free를 반드시 해줘야 한다. 그러므로 해당 프로그램은 잘못되었다.

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind **on it?**

문제의 요구사항을 만족하는 프로그램의 소스코드는 다음과 같다.

home > borussen > C free_print.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int *data = (int *) malloc(100);
    data[100] = 0;
    free(data);
    printf("%d\n", data[20]);  // print the value of one of the elements of the freed array
    return 0;
}
```

다음과 같이 해당 프로그램은 정상적으로 컴파일 되고 실행된다.

```
borussen@DESKTOP-L834KLC:~$ gcc -g free_print.c -o free_print
borussen@DESKTOP-L834KLC:~$ ./free_print
0
```

올바른 결과값이 출력되었음에도 불구하고, valgrind를 이용해 해당 프로그램을 실행하면 invalid write와 invalid read가 발생했다는 output을 출력한다.

```
borussen@DESKTOP-L834KLC:~$ valgrind --leak-check=yes ./free_print
==3220== Memcheck, a memory error detector
==3220== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3220== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3220== Command: ./free_print
==3220==
==3220== Invalid write of size 4
==3220==    at 0x1091B4: main (free_print.c:6)
==3220==  Address 0x4a471d0 is 224 bytes inside an unallocated block of size 4,194,032 in arena "client"
==3220==
==3220== Invalid read of size 4
==3220==    at 0x1091CE: main (free_print.c:8)
==3220==  Address 0x4a47090 is 80 bytes inside a block of size 100 free'd
==3220==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3220==    by 0x1091C5: main (free_print.c:7)
==3220==  Block was alloc'd at
==3220==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3220==    by 0x1091A5: main (free_print.c:5)
==3220==
0
==3220==
==3220== HEAP SUMMARY:
==3220==     in use at exit: 0 bytes in 0 blocks
==3220==   total heap usage: 2 allocs, 2 frees, 1,124 bytes allocated
==3220==
==3220== All heap blocks were freed -- no leaks are possible
==3220==
==3220== For lists of detected and suppressed errors, rerun with: -s
==3220== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?

문제의 요구사항을 만족하는 프로그램의 소스코드는 다음과 같다.

home > borussen > C funny.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int *data = (int *) malloc(100);
    *(data+50) = 10;
    free(data+50);
    return 0;
}
```

해당 프로그램은 컴파일 되기는 하지만, 다음과 같이 runtime error가 발생한다.

```
borussen@DESKTOP-L834KLC:~$ gcc -g funny.c -o funny
borussen@DESKTOP-L834KLC:~$ ./funny
free(): invalid pointer
Aborted
```

gdb를 사용할 경우, Abort가 호출되었음을 보고함으로써 발생하는 SIGABRT 신호를 출력한다.

```
(gdb) run
Starting program: /home/borussen/funny
free(): invalid pointer

Program received signal SIGABRT, Aborted.
__GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:50
50      ../sysdeps/unix/sysv/linux/raise.c: No such file or directory.
```

valgrind를 사용할 경우, 해당 프로그램에서 invalid write, invalid free 그리고 heap 영역의 memory leak이 발생했다는 것을 알려주기 때문에 문제를 보다 자세하게 파악할 수 있다.

```
borussen@DESKTOP-L834KLC:~$ valgrind --leak-check=yes ./funny
==3492== Memcheck, a memory error detector
==3492== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3492== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3492== Command: ./funny
==3492==
==3492== Invalid write of size 4
==3492==    at 0x109194: main (funny.c:7)
==3492==  Address 0x4a47108 is 24 bytes inside an unallocated block of size 4,194,032 in arena "client"
==3492==
==3492== Invalid free() / delete / delete[] / realloc()
==3492==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3492==    by 0x1091AB: main (funny.c:8)
==3492==  Address 0x4a47108 is 24 bytes inside an unallocated block of size 4,194,032 in arena "client"
==3492==
==3492==
==3492== HEAP SUMMARY:
==3492==     in use at exit: 100 bytes in 1 blocks
==3492==   total heap usage: 1 allocs, 1 frees, 100 bytes allocated
==3492==
==3492== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==3492==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3492==    by 0x109185: main (funny.c:6)
==3492==
==3492== LEAK SUMMARY:
==3492==    definitely lost: 100 bytes in 1 blocks
==3492==    indirectly lost: 0 bytes in 0 blocks
==3492==      possibly lost: 0 bytes in 0 blocks
==3492==    still reachable: 0 bytes in 0 blocks
==3492==         suppressed: 0 bytes in 0 blocks
==3492==
==3492== For lists of detected and suppressed errors, rerun with: -s
==3492== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

8. Try out some of the other interfaces to memory allocation. For example, create a simple vector-like data structure and related routines that use realloc() **to manage the vector. Use an array to store the vectors elements; when a user adds an entry to the vector, use** realloc() **to allocate more space for it. How well does such a vector perform? How does it compare to a linked list? Use** valgrind **to help you find bugs.**

문제의 요구사항을 만족하는 프로그램의 소스코드는 다음과 같다. realloc()과 array를 활용하여 vector-like data structure를 만들었고, 추가적으로 vector element를 저장하는 insert, vector element를 제거하는 delete 그리고 vector element의 값을 가져오는 get 기능을 구현했다.

home > borussen > C vector.c

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct vector {
    int *data;
    int capacity;
    int size;
} Vector;

void insert(Vector *v, int value);
void delete(Vector *v);
int get(Vector *v, int index);
void freeVector(Vector *v);

int main(int argc, char *argv[]) {

    Vector v = {.data = (int *)malloc(sizeof(int)), .capacity = 1, .size = 1};
    v.data[0] = 10;

    Vector *vp = &v;

    insert(vp, 20);
    delete(vp);
    insert(vp, 30);
    insert(vp, 40);

    printf("%d %d %d\n", get(vp, 0), get(vp, 1), get(vp, 2));
    printf("vector capacity: %d\n", v.capacity);
    printf("vector size: %d\n", v.size);

    freeVector(vp);
    return 0;
}

void insert(Vector *v, int value) {
    if (v->size >= v->capacity) {
        v->data = (int *) realloc(v->data, (v->capacity + 1) * sizeof(int));
        v->capacity++;
    }
    (v->data)[v->size++] = value;
}
void delete(Vector *v) {
    if (v->size > 0) {
        v->data[--(v->size)] = 0;
    }
}
int get(Vector *v, int index) {
    return v->data[index];
}
void freeVector(Vector *v) {
    free(v->data);
    v->size = 0;
    v->capacity = 0;
}
```

프로그램 상에서 10과 20이 차례대로 insert 된 이후 delete가 한 차례 일어나고 30과 40이 다시 insert 되었으므로 실행결과는 10, 30, 40이 출력되고 capacity와 size는 모두 3이어야 한다. 프로그램을 실행하면 다음과 같이 동일한 결과가 출력되므로 벡터는 정상적으로 작동한다.

```
borussen@DESKTOP-L834KLC:~$ gcc -g vector.c -o vector
borussen@DESKTOP-L834KLC:~$ ./vector
10 30 40
vector capacity: 3
vector size: 3
```

Vector는 미리 일정 크기의 메모리를 할당해 놓고 그 이상의 값이 추가되면 새로운 더 큰 메모리를 할당하기 때문에 값을 넣을 때마다 메모리를 할당하는 Linked list에 비해 pushback 속도가 빠르다. 또한, 구조상 element의 index로 직접 접근이 불가능한 Linked list와는 다르게 Vector는 개별 element의 접근이 가능하여 특정 element에 대한 접근 속도가 빠르다는 장점을 가진다. 하지만 Vector의 경우, 끝이 아닌 곳에서 element를 삽입하거나 제거하게 되면 Linked list에 비해 성능이 현저하게 떨어진다는 단점이 있다.

마지막으로 valgrind를 통해 해당 프로그램에 문제가 없음을 확인하였다.

```
borussen@DESKTOP-L834KLC:~$ valgrind --leak-check=yes ./vector
==7331== Memcheck, a memory error detector
==7331== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7331== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==7331== Command: ./vector
==7331==
10 30 40
vector capacity: 3
vector size: 3
==7331==
==7331== HEAP SUMMARY:
==7331==     in use at exit: 0 bytes in 0 blocks
==7331==   total heap usage: 4 allocs, 4 frees, 1,048 bytes allocated
==7331==
==7331== All heap blocks were freed -- no leaks are possible
==7331==
==7331== For lists of detected and suppressed errors, rerun with: -s
==7331== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```