System Programming
Lab #1

2023-03-14

sp-tas

Library Interpositioning

- Library Interpositioning
  - 임의의 라이브러리 함수 실행 시에 함수의 호출을 가로채어 원하는 명령을 실행할 수 있게 하는 기술

- 종류
  - **Compile time**: 소스코드가 컴파일 될 때
  - **Link time**: relocatable object file이 executable object file에 정적 링크 될 때
  - **Load/Run time**: executable object file이 메모리에 로드되고, 동적으로 링크되고 실행될 때

Some Interpositioning Applications

- **Security**
  - Confinement (sandboxing)
    - Interpose calls to libc functions.
  - Behind the scenes encryption
    - Automatically encrypt otherwise unencrypted network connections.
- **Monitoring and Profiling**
  - Count number of calls to functions
  - Characterize call sites and arguments to functions
  - Malloc tracing
    - Detecting memory leaks
    - **Generating address traces**

Example program

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

int main()
{
    free(malloc(10));
    printf("hello, world\n");
    exit(0);
}
```

hello.c

- Goal: trace the addresses and sizes of the allocated and freed blocks, without modifying the source code.
- Three solutions: interpose on the `lib malloc` and `free` functions at compile time, link time, and load/run time.

Compile-time Interpositioning

```
#ifdef COMPILETIME
/* Compile-time interposition of malloc and free using C
 * preprocessor. A local malloc.h file defines malloc (free)
 * as wrappers mymalloc (myfree) respectively.
 */

#include <stdio.h>
#include <malloc.h>

/*
 * mymalloc - malloc wrapper function
 */
void *mymalloc(size_t size, char *file, int line)
{
    void *ptr = malloc(size);
    printf("%s:%d: malloc(%d)=%p\n", file, line, (int)size,
ptr);
    return ptr;
}
```

mymalloc.c

Compile-time Interpositioning

```
#define malloc(size) mymalloc(size, __FILE__, __LINE__ )
#define free(ptr) myfree(ptr, __FILE__, __LINE__ )

void *mymalloc(size_t size, char *file, int line);
void myfree(void *ptr, char *file, int line);
```

malloc.h

```
linux> make helloc
gcc -O2 -Wall -DCOMPILETIME -c mymalloc.c
gcc -O2 -Wall -I. -o helloc hello.c mymalloc.o
linux> make runc
./helloc
hello.c:7: malloc(10)=0x501010
hello.c:7: free(0x501010)
hello, world
```

Link-time Interpositioning

```
#ifdef LINKTIME
/* Link-time interposition of malloc and free using the
static linker's (ld) "--wrap symbol" flag. */

#include <stdio.h>

void *__real_malloc(size_t size);
void __real_free(void *ptr);

/*
 * __wrap_malloc - malloc wrapper function
 */
void *__wrap_malloc(size_t size)
{
    void *ptr = __real_malloc(size);
    printf("malloc(%d) = %p\n", (int)size, ptr);
    return ptr;
}
```

mymalloc.c

Link-time Interpositioning

```
linux> make hellol
gcc -O2 -Wall -DLINKTIME -c mymalloc.c
gcc -O2 -Wall -Wl,--wrap,malloc -Wl,--wrap,free \
-o hellol hello.c mymalloc.o
linux> make runl
./hellol
malloc(10) = 0x501010
free(0x501010)
hello, world
```

- The "`-Wl`" flag passes argument to linker
- Telling linker "`--wrap,malloc`" tells it to resolve references in a special way:
  - Refs to `malloc` should be resolved as `__wrap_malloc`
  - Refs to `__real_malloc` should be resolved as `malloc`

Load/Run-time Interpositioning

```
#ifdef RUNTIME
 /* Run-time interposition of malloc and free based on
 * dynamic linker's (ld-linux.so) LD_PRELOAD mechanism */
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

void *malloc(size_t size)
{
    void *(*mallocp)(size_t size);
    char *error;
    void *ptr;

    /* get address of libc malloc */
    if (!mallocp) {
        mallocp = dlsym(RTLD_NEXT, "malloc");
        if ((error = dlerror()) != NULL) {
            fputs(error, stderr);
            exit(1);
        }
    }
    ptr = mallocp(size);
    fprintf(stderr, "malloc(%d) = %p\n", (int)size, ptr);
    return ptr;
}
```

mymalloc.c

Load/Run-time Interpositioning

```
linux> make hellor
gcc -O2 -Wall -DRUNTIME -shared -fPIC -o mymalloc.so mymalloc.c
gcc -O2 -Wall -o hellor hello.c
linux> make runr
(LD_PRELOAD="/usr/lib/x86_64-linux-gnu/libdl.so ./mymalloc.so"
./hellor)
malloc(10) = 0x559a34eca260
free(0x559a34eca260)
hello, world
```

- **`The LD_PRELOAD` environment variable tells the dynamic linker to resolve unresolved refs (e.g., to `malloc`) by looking in `libdl.so` and `mymalloc.so` first.**
  - **`libdl.so`** necessary to resolve references to the **`dlopen`** functions.

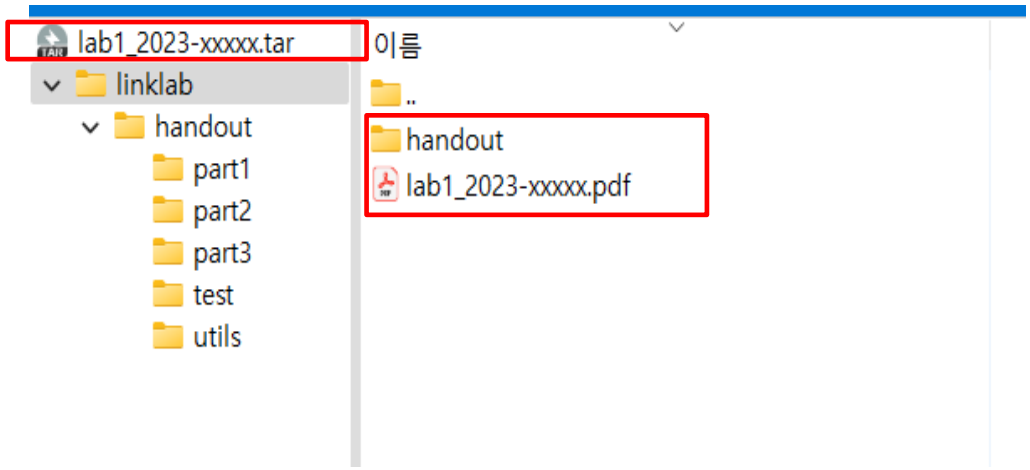
- **-shared: telling linker to make output as a shared objective (.so)**
- **-fPIC: Position-Independent Code**

Interpositioning Recap

- **Compile Time**
  - Apparent calls to malloc/free get macro-expanded into calls to mymalloc/myfree
- **Link Time**
  - Use linker trick to have special name resolutions
    - malloc → __wrap_malloc
    - __real_malloc → malloc
- **Compile Time**
  - Implement custom version of malloc/free that use dynamic linking to load library malloc/free under different names

실습 및 숙제

- Tracing Dynamic memory management
- Using **load/run time** library interpositioning
- Installation
  - part 0
- Assignment
  - part 1, 2, 3(each 30 pts)
  - report (10 pts)
- Due date
  - ~3.27(월) 23:59 (2 weeks)
- Submission
  - Upload your files on eTL
  - Compress (using tar) your implementation and PDF type report
  - File name: lab1_학번.tar/pdf (lab1_2023-xxxxx.tar)
  - 압축파일의 내용과 이름을 반드시 오른쪽 사진과 똑같이



Dynamic Memory Management

`void *malloc(size_t size)`

`malloc` allocates `size` bytes of memory on the process' heap and returns a pointer to it that can subsequently be used by the process to hold up to `size` bytes. The contents of the memory are undefined.

`void *calloc(size_t nmemb, size_t size)`

`calloc` allocates `nmemb*size` bytes of memory on the process' heap and returns a pointer to it that can subsequently be used by the process to hold up to `size` bytes. The contents of the memory are set to zero.

`void *realloc(void *ptr, size_t size)`

`realloc` changes the size of the memory block pointed to by `ptr` to `size` bytes. The contents are copied up to `min(size, old size)`, the rest is undefined.

`void free(void *ptr)`

`free` explicitly frees a previously allocated block of memory.

Dynamic Memory Management

```c
#include <stdlib.h>

void main(void) {
  void *p;
  char *str;
  int  *A;

  // allocated 1024 bytes of memory
  p = malloc(1024);

  // allocated an integer array with 500 integer
  A = (int*)calloc(500, sizeof(int));

  // allocate a string with 16 characters…
  str = (char*)malloc(16*sizeof(char));

  // ...then resize that string to hold 512 characters
  str = (char*)realloc(str, 512*sizeof(char));

  // finally, free all allocated memory
  free(p);
  free(A);
  free(str);
}
```

example1.c

Shared library loading interfaces

- `#include <dlfcn.h>`

- `void *dlopen(const char *filename, int flags)`
  - `dlopen` loads the dynamic shared object (shared library) file named by the null-terminated string filename and returns an opaque "handle" for the loaded object.
  - **`RTLD_LAZY`** – perform lazy binding
  - **`RTLD_NOW`** – all undefined symbols in the shared object are resolved before `dlopen` returns

- `void *dlsym(void *handle, const char *symbol)`
  - `dlsym` takes a "handle" of a dynamic loaded shared object returned by `dlopen` and returns the address where that symbol is loaded into memory
  - `RTLD_DEFAULT` – find the first occurrence of the desired symbol
  - **`RTLD_NEXT`** – find the next occurrence of the desired symbol in the search order after the current object

- `int dlclose(void *handle)`
  - `dlclose` decrements the reference count on the dynamically loaded shared object referred to by `handle`

(Part 0) installation

- 압축해제
- $cd test
- $make
- $cd ../part1
- $make compile
- $make run test[n]
  - ex)make run test1

test파일의 make 결과

```
ta@ubuntu:~/linklab/handout/test$ ls
Makefile  test1.c  test2.c  test3.c  test4.c  test5.c  testx  testx.c
ta@ubuntu:~/linklab/handout/test$ make
cc -O2 -fno-dce -fno-dse -fno-tree-dce -fno-tree-dse -o test2 test2.c
cc -O2 -fno-dce -fno-dse -fno-tree-dce -fno-tree-dse -o test4 test4.c
cc -O2 -fno-dce -fno-dse -fno-tree-dce -fno-tree-dse -o test3 test3.c
cc -O2 -fno-dce -fno-dse -fno-tree-dce -fno-tree-dse -o test5 test5.c
cc -O2 -fno-dce -fno-dse -fno-tree-dce -fno-tree-dse -o test1 test1.c
```

installation test

```
ta@ubuntu:~/linklab/handout/part1$ make compile
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/memlist.c -ldl
ta@ubuntu:~/linklab/handout/part1$ ls
libmemtrace.so  Makefile  memtrace.c
ta@ubuntu:~/linklab/handout/part1$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/memlist.c -ldl
[0001] Memory tracer started.
[0002]
[0003] Statistics
[0004]   allocated_total      0
[0005]   allocated_avg        0
[0006]   freed_total          0
[0007]
[0008] Memory tracer stopped.
```

(Part 1) Tracing dynamic memory allocation

- Write code in **part1/memtrace.c**

- Trace
  - allocation size, address
  - deallocation address
  - total, average allocated size

- freed_total은 0인 상태

- Use macros in util/memlog.h to print output
  - printf 사용하면 안됨

- Realloc으로 재할당받는 경우, 원래 할당받은 메모리 사이즈를 빼지 않고 모두 더함

(Part 1) Tracing dynamic memory allocation

test1.c

```
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  a = malloc(32);
  free(malloc(1));
  free(a);
}
```

output

```
ta@ubuntu:~/linklab_sol/handout/part1$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c
../utils/memlist.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x55c8c4aa22d0
[0003]          malloc( 32 ) = 0x55c8c4aa2710
[0004]          malloc( 1 ) = 0x55c8c4aa2770
[0005]          free( 0x55c8c4aa2770 )
[0006]          free( 0x55c8c4aa2710 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          0
[0012]
[0013] Memory tracer stopped.
```

(Part 2) Tracing unfreed memory

- Copy memtrace.c from part1 to part2 directory
- Add code in **part2/memtrace.c**

- Trace
  - part1
  - non-freed block
  - total freed size

- Use linked list functions in util/memlist.c, .h

*Consider reallocation

(Part 2) Tracing unfreed memory

test1.c

```c
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  a = malloc(32);
  free(malloc(1));
  free(a);
}
```

output

```
ta@ubuntu:~/linklab_sol/handout/part2$ make run test1
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/memlist.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x556f950592d0
[0003]          malloc( 32 ) = 0x556f95059710
[0004]          malloc( 1 ) = 0x556f95059770
[0005]          free( 0x556f95059770 )
[0006]          free( 0x556f95059710 )
[0007]
[0008] Statistics
[0009]   allocated_total      1057
[0010]   allocated_avg        352
[0011]   freed_total          33
[0012]
[0013] Non-deallocated memory blocks
[0014]   block              size       ref cnt
[0015]   0x556f950592d0     1024       1
[0016]
[0017] Memory tracer stopped.
```

(Part 2) Tracing unfreed memory

test2 output

```
ta@ubuntu:~/linklab_sol/handout/part2$ make run test2
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/memlist.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x563e230b72d0
[0003]          free( 0x563e230b72d0 )
[0004]
[0005] Statistics
[0006]   allocated_total      1024
[0007]   allocated_avg        1024
[0008]   freed_total          1024
[0009]
[0010] Memory tracer stopped.
```

- 모든 블락이 해제된 경우 non-deallocated memory blocks가 출력되면 안됨

(Part 3) Detect and ignore illegal deallocations

- Copy memtrace.c from part2 to part3 directory
- Add code in **part3/memtrace.c**

- Trace
  - part1, 2
  - illegal free, double free
  - illegal free
    - 할당되지 않은 메모리를 할당 해제
  - Double free
    - 이미 free한 메모리를 다시 free

- Trace, but ignore illegal free (not to invoke error)

(Part 3) Detect and ignore illegal deallocations

- Detect double- free / illegal free

`test4.c – test case for bonus part`

```
#include <stdlib.h>

void main(void) {
  void *a;

  a = malloc(1024);
  free(a);
  free(a);
  free((void*)0x1706e90);
}
```

output

```
ta@ubuntu:~/linklab_sol/handout/part3$ make run test4
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/memlist.c -ldl
[0001] Memory tracer started.
[0002]          malloc( 1024 ) = 0x558ac6ef62d0
[0003]          free( 0x558ac6ef62d0 )
[0004]          free( 0x558ac6ef62d0 )
[0005]    *** DOUBLE_FREE  *** (ignoring)
[0006]          free( 0x1706e90 )
[0007]    *** ILLEGAL_FREE *** (ignoring)
[0008]
[0009] Statistics
[0010]   allocated_total      1024
[0011]   allocated_avg        1024
[0012]   freed_total          1024
[0013]
[0014] Memory tracer stopped.
```

(Part 3) Detect and ignore illegal deallocations

- Test5 output

```
cc -I. -I ../utils -o libmemtrace.so -shared -fPIC memtrace.c ../utils/memlog.c ../utils/memlist.c -ldl
[0001] Memory tracer started.
[0002]           malloc( 10 ) = 0xaaaae85d12d0
[0003]           realloc( 0xaaaae85d12d0 , 100 ) = 0xaaaae85d1320
[0004]           realloc( 0xaaaae85d1320 , 1000 ) = 0xaaaae85d13c0
[0005]           realloc( 0xaaaae85d13c0 , 10000 ) = 0xaaaae85d17e0
[0006]           realloc( 0xaaaae85d17e0 , 100000 ) = 0xaaaae85d3f30
[0007]           free( 0xaaaae85d3f30 )
[0008]
[0009] Statistics
[0010]   allocated_total      111110
[0011]   allocated_avg        22222
[0012]   freed_total          111110
[0013]
[0014] Memory tracer stopped.
```

Skeleton code snippet

```c
//
// init - this function is called once when the shared library is loaded
//
__attribute__((constructor))
void init(void)
{
  char *error;

  LOG_START();

  // initialize a new list to keep track of all memory (de-)allocations
  // (not needed for part 1)
  list = new_list();

  // ...
}
```

```c
//
// fini - this function is called once when the shared library is unloaded
//
__attribute__((destructor))
void fini(void)
{
  // ...

  LOG_STATISTICS(0L, 0L, 0L);

  LOG_STOP();

  // free list (not needed for part 1)
  free_list(list);
}
```

Utilities

- Read someone else's code

- `memlog.c`
  - `int mlog(int pc, const char* fmt, …)`

- `memlist.c`
  - `item *new_list(void)`
  - `void free_list(void)`
  - `item *alloc(item *list, void *ptr, size_t size)`
  - `item *dealloc(item *list, void *ptr)`
  - `item *find(item *list, void *ptr)`
  - `void dump_list(item *list)`

Report

- 실행 결과
- 어떻게 구현했는지(파트 별로)
- 어려웠던 점
- 새롭게 배운 점

Q&A

- Specification
  - 질문하기 전 과제 설명 먼저 읽기
- Printf로 log를 출력하면 안됨
  - fprintf or mlog 사용
- part1, 2는 test4를 실행할 경우 에러메시지가 나오는 것이 정상
- 각 part에 대해 다음 Test에 대해서만 만족하도록 구현
  - Part1 – test1,2,3
  - Part2 – test1,2,3
  - Part3 – test4,5
- Freed_total에 realloc 내부에서 발생하는 free반영 여부
  - Realloc()은 기존 메모리 Free와 새 메모리 alloc을 모두 수행한다고 가정하고 Memtrace를 작성. 즉 기존의 포인터 주소를 그대로 return 하더라도 기존 메모리를 Free하고 Re-alloc한 것으로 간주(Log는 realloc만 출력)
- Allocated_total에 realloc 이후 크기 반영 여부
  - Realloc 요청된 크기를 단순히 모두 더하는 방향으로 진행
- Non-deallocated메모리 트레이싱
  - 최소 1개의 block이상 존재할 때만 출력
- Use eTL Q&A for other question

Tips

- 시작 전 과제 스펙 숙지
- 다른 사람의 코드를 읽는 것에 익숙해지기

Next time

- Lab2
  - 3/28 : shell lab