운영체제

Assignment 4 과제

수업 명 : 운영체제

과제 이름 : assignment4

담당 교수님 : 최상호 교수님

학 번 : 2019202005

이 름 : 남종식

Introduction

4-1 과제는 PID 를 바탕으로 프로세스의 이름과 pid, 정보가 위치하는 가상 메모리 주소, 프로세스의 데이터 주소, 코드 주소, 힙 주소, 정보의 원본 파일의 전체 경로를 출력하는 모듈을 작성하는 과제입니다. 이번 과제에서 또한 2 차 과제에서 작성한 ftrace 시스템 콜을 file_varea 로 wrapping 하여 사용합니다. 위 정보들을 출력하기 위해서 task_struct 에 대해 잘 살펴보아야 합니다.
4-2 과제는 공유 메모리에 존재하는 코드에 대해서 최적화를 진행하여 최적화 후의 실행결과를 최적화 전의 실행결과와 비교해야 합니다. 불필요하게 반복되는 과정을 없앤 후 dynamic recompilation 을 통해 최적화를 진행합니다.

Result

4-1 과제

먼저 과제에서 요구하는 프로세스의 정보에 대해서 출력하기 위해서 task_struct 에서 찾아보았습니다.

프로세스의 이름과 pid

```
	 * - lock it with task_lock()
	 */
	char				comm[TASK_COMM_LEN];

```

Comm 을 통해 프로세스의 이름을 알 수 있습니다.

정보가 위치하는 가상 메모리 주소

```
#endif

	struct mm_struct		*mm;
	struct mm_struct		*active_mm;

	/* Per-thread vma caching: */
```

```
struct mm_struct {
	struct {
		struct vm_area_struct *mmap;		/* list of VMAs */
		struct rb_root mm_rb;
		u64 vmacache_seqnum;		/* per-thread vmacache */
```

```
/*
 * This struct defines a memory VMM memory area. There is one of these
 * per VM-area/task.  A VM area is any part of the process virtual memory
 * space that has a special rule for the page-fault handlers (ie a shared
 * library, the executable area etc).
 */
struct vm_area_struct {
	/* The first cache line has the info for VMA tree walking. */

	unsigned long vm_start;		/* Our start address within vm_mm. */
	unsigned long vm_end;		/* The first byte after our end address
					   within vm_mm. */

	/* linked list of VM areas per task, sorted by address */
	struct vm_area_struct *vm_next, *vm_prev;

	struct rb_node vm_rb;

	/*
	 * Largest free memory gap in bytes to the left of this VMA.
	 * Either between this VMA and vma->vm_prev, or between one of the
	 * VMAs below us in the VMA rbtree and its ->vm_prev. This helps
	 * get_unmapped_area find a free area of the right size.
	 */
	unsigned long rb_subtree_gap;
```

struct mm_struct 은 리눅스 커널에서 메모리 관리 정보를 담는 구조체입니다. 이 구조체는 각각의 프로세스에 대한 메모리 관리 정보를 저장합니다.

Mmap 은 struct vm_area_struct 연결 리스트를 가리키는 포인터입니다. 각각의 vm_area_struct 구조체는 가상 주소 공간에서 특정한 메모리 영역에 대한 정보를 저장합니다. 이 리스트를 통해 프로세스의 가상 주소 공간에 할당된 메모리 영역들을 순회할 수 있습니다.

프로세스의 데이터 주소, 코드 주소, 힙 주소

```
spinlock_t arg_lock; /* protect the below fields */
unsigned long start_code, end_code, start_data, end_data;
unsigned long start_brk, brk, start_stack;
unsigned long arg_start, arg_end, env_start, env_end;
```

start_code, end_code, start_data, end_data 는 실행 코드와 데이터의 시작과 끝을 가리키는 값들입니다. 이 정보는 프로세스의 메모리 레이아웃을 나타냅니다.

start_brk, brk 는 힙 영역의 시작과 끝을 가리키는 값입니다. 동적으로 할당되는 메모리가 여기에 위치합니다.

정보의 원본 파일의 전체 경로

```
/* Information about our backing store: */
unsigned long vm_pgoff;		/* Offset (within vm_file) in PAGE_SIZE
					   units */
struct file * vm_file;		/* File we map to (can be NULL). */
void * vm_private_data;		/* was vm_pte (shared mem) */
```

```
struct file {
        union {
                struct llist_node       fu_llist;
                struct rcu_head         fu_rcuhead;
        } f_u;
        struct path             f_path;
        struct inode            *f_inode;       /* cached value */
        const struct file_operations    *f_op;
```

```
struct dentry;
struct vfsmount;

struct path {
        struct vfsmount *mnt;
        struct dentry *dentry;
} __randomize_layout;
```

mnt 는 파일 시스템을 나타내는 구조체로, 해당 파일의 마운트 정보를 가리킵니다. dentry 는 파일이나 디렉토리를 나타내는 구조체로, 디렉토리 엔트리 정보를 가리킵니다

```
/**
 * d_path - return the path of a dentry
 * @path: path to report
 * @buf: buffer to return value in
 * @buflen: buffer length
 *
 * Convert a dentry into an ASCII path name. If the entry has been deleted
 * the string " (deleted)" is appended. Note that this is ambiguous.
 *
 * Returns a pointer into the buffer or an error code if the path was
 * too long. Note: Callers should use the returned pointer, not the passed
 * in buffer, to use the name! The implementation often starts at an offset
 * into the buffer, and may leave 0 bytes at the start.
 *
 * "buflen" should be positive.
 */
char *d_path(const struct path *path, char *buf, int buflen)
{
        char *res = buf + buflen;
        struct path root;
        int error;

        /*
         * We have various synthetic filesystems that never get mounted.  On
         * these filesystems dentries are never used for lookup purposes, and
         * thus don't need to be hashed.  They also don't need a name until a
         * user wants to identify the object in /proc/pid/fd/.  The little hack
         * below allows us to generate a name for these objects on demand:
         *
         * Some pseudo inodes are mountable.  When they are mounted
         * path->dentry == path->mnt->mnt_root.  In that case don't call d_dname
         * and instead have d_path return the mounted path.
         */
```

d_path 함수는 struct path 를 받아 파일의 경로를 문자열로 변환하여 반환합니다. 이 함수를 사용하여 파일의 전체 경로를 알아낼 수 있습니다. 예를 들어, 파일이 어떤 디렉터리에 위치하고 있는지를 확인하거나, 파일 시스템의 경로를 문자열로 얻고 싶을 때 사용됩니다.

```makefile
obj-m := file_varea.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
	gcc -o test test.c

clean:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean

```

테스트용 파일도 같이 컴파일 되도록 Makefile 을 작성하였습니다.

```
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-1$ sudo insmod file_varea.ko
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-1$ ./test
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-1$ sudo rmmod file_varea
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-1$ dmesg
```

모듈 적재 후 실행한 다음 삭제 후 출력을 진행했습니다.

```
######### Loaded files of a process 'test(9627)' in VM ########
mem[400000~401000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /home/os2019202005/Downloads/linux-4.19.67/os4/4-1/test
mem[600000~601000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /home/os2019202005/Downloads/linux-4.19.67/os4/4-1/test
mem[601000~602000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /home/os2019202005/Downloads/linux-4.19.67/os4/4-1/test
mem[7f22143a2000~7f2214562000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/libc-2.23.so
mem[7f2214562000~7f2214762000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/libc-2.23.so
mem[7f2214762000~7f2214766000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/libc-2.23.so
mem[7f2214766000~7f2214768000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/libc-2.23.so
mem[7f221476c000~7f2214792000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/ld-2.23.so
mem[7f2214991000~7f2214992000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/ld-2.23.so
mem[7f2214992000~7f2214993000] code[400000~40074c] data[600e10~601040] heap[118f000~118f000] /lib/x86_64-linux-gnu/ld-2.23.so
############################################################
```

강의 자료 형식에 맞게 과제에서 요구하는 프로세스의 정보를 잘 출력하는 모습을 확인할 수 있습니다.

4-2 과제

```
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ gcc -c D_recompile_test.c
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ objdump -d D_recompile_test.o > test.txt
```

다음은 objdump 가 뜨는 과정입니다. 먼저 gcc –c D_recompile_test.c 을 통해 Object file 을 생성 후, objdump –d D_recompile_test.o 을 통해, D_recompile_test.o file 을 disassemble 하여 objdump 결과가 출력되는데, 이 부분은 redirection 으로 test.txt 에 저장했습니다.

```
D_recompile_test.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <Operation>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   89 7d fc                mov    %edi,-0x4(%rbp)
   7:   8b 55 fc                mov    -0x4(%rbp),%edx
   a:   89 d0                   mov    %edx,%eax
   c:   b2 02                   mov    $0x2,%dl
   e:   83 c0 01                add    $0x1,%eax
  11:   83 c0 01                add    $0x1,%eax
  14:   83 c0 01                add    $0x1,%eax
  17:   83 c0 01                add    $0x1,%eax
  1a:   83 c0 02                add    $0x2,%eax
  1d:   83 c0 03                add    $0x3,%eax
  20:   83 c0 01                add    $0x1,%eax
  23:   83 c0 02                add    $0x2,%eax
  26:   83 c0 01                add    $0x1,%eax
  29:   83 c0 01                add    $0x1,%eax
  2c:   6b c0 02                imul   $0x2,%eax,%eax
  2f:   6b c0 02                imul   $0x2,%eax,%eax
  32:   6b c0 02                imul   $0x2,%eax,%eax
  35:   83 c0 01                add    $0x1,%eax
  38:   83 c0 01                add    $0x1,%eax
  3b:   83 c0 03                add    $0x3,%eax
  3e:   83 c0 01                add    $0x1,%eax
  41:   83 c0 01                add    $0x1,%eax
  44:   83 c0 01                add    $0x1,%eax
  47:   83 c0 03                add    $0x3,%eax
  4a:   83 c0 01                add    $0x1,%eax
  4d:   83 c0 01                add    $0x1,%eax
  50:   83 c0 02                add    $0x2,%eax
  53:   83 c0 01                add    $0x1,%eax
  56:   83 c0 01                add    $0x1,%eax
  59:   83 c0 01                add    $0x1,%eax
  5c:   83 c0 01                add    $0x1,%eax
  5f:   83 c0 01                add    $0x1,%eax
  62:   f6 f2                   div    %dl
  64:   f6 f2                   div    %dl
  66:   83 e8 01                sub    $0x1,%eax
  69:   83 e8 01                sub    $0x1,%eax
  6c:   83 e8 03                sub    $0x3,%eax
  6f:   83 e8 01                sub    $0x1,%eax
  72:   83 e8 01                sub    $0x1,%eax
  75:   83 e8 01                sub    $0x1,%eax
  78:   83 e8 03                sub    $0x3,%eax
  7b:   83 e8 01                sub    $0x1,%eax
  7e:   83 e8 01                sub    $0x1,%eax
  81:   83 e8 02                sub    $0x2,%eax
  84:   83 e8 01                sub    $0x1,%eax
  87:   83 e8 01                sub    $0x1,%eax
  8a:   83 e8 01                sub    $0x1,%eax
  8d:   83 e8 01                sub    $0x1,%eax
  90:   83 e8 01                sub    $0x1,%eax
  93:   6b c0 02                imul   $0x2,%eax,%eax
  96:   6b c0 02                imul   $0x2,%eax,%eax
  99:   6b c0 02                imul   $0x2,%eax,%eax
  9c:   83 c0 01                add    $0x1,%eax
  9f:   83 c0 01                add    $0x1,%eax
  a2:   83 c0 03                add    $0x3,%eax
  a5:   83 c0 01                add    $0x1,%eax
  a8:   83 c0 01                add    $0x1,%eax
  ab:   83 c0 01                add    $0x1,%eax
  ae:   83 c0 03                add    $0x3,%eax
  b1:   83 c0 01                add    $0x1,%eax
  b4:   83 c0 01                add    $0x1,%eax
  b7:   83 c0 02                add    $0x2,%eax
  ba:   83 c0 01                add    $0x1,%eax
  bd:   83 c0 01                add    $0x1,%eax
  c0:   83 c0 01                add    $0x1,%eax
  c3:   83 c0 01                add    $0x1,%eax
  c6:   83 c0 01                add    $0x1,%eax
  c9:   f6 f2                   div    %dl
  cb:   f6 f2                   div    %dl
  cd:   83 e8 01                sub    $0x1,%eax
  d0:   83 e8 01                sub    $0x1,%eax
  d3:   83 e8 03                sub    $0x3,%eax
  d6:   83 e8 01                sub    $0x1,%eax
  d9:   83 e8 01                sub    $0x1,%eax
  dc:   83 e8 01                sub    $0x1,%eax
  df:   83 e8 03                sub    $0x3,%eax
  e2:   83 e8 01                sub    $0x1,%eax
  e5:   83 e8 01                sub    $0x1,%eax
  e8:   83 e8 02                sub    $0x2,%eax
  eb:   83 e8 01                sub    $0x1,%eax
  ee:   83 e8 01                sub    $0x1,%eax
```

```
 c40:	89 55 fc             	mov    %edx,-0x4(%rbp)
 c43:	8b 45 fc             	mov    -0x4(%rbp),%eax
 c46:	5d                   	pop    %rbp
 c47:	c3                   	retq

0000000000000c48 <main>:
 c48:	55                   	push   %rbp
 c49:	48 89 e5             	mov    %rsp,%rbp
 c4c:	48 83 ec 20          	sub    $0x20,%rsp
 c50:	48 c7 45 f0 00 00 00 	movq   $0x0,-0x10(%rbp)
 c57:	00
 c58:	c7 45 e8 00 00 00 00 	movl   $0x0,-0x18(%rbp)
 c5f:	ba 80 03 00 00       	mov    $0x380,%edx
 c64:	be 00 10 00 00       	mov    $0x1000,%esi
 c69:	bf d2 04 00 00       	mov    $0x4d2,%edi
 c6e:	e8 00 00 00 00       	callq  c73 <main+0x2b>
 c73:	89 45 ec             	mov    %eax,-0x14(%rbp)
 c76:	8b 45 ec             	mov    -0x14(%rbp),%eax
 c79:	ba 00 00 00 00       	mov    $0x0,%edx
 c7e:	be 00 00 00 00       	mov    $0x0,%esi
 c83:	89 c7                	mov    %eax,%edi
 c85:	e8 00 00 00 00       	callq  c8a <main+0x42>
 c8a:	48 89 45 f8          	mov    %rax,-0x8(%rbp)
 c8e:	8b 45 e8             	mov    -0x18(%rbp),%eax
 c91:	8d 50 01             	lea    0x1(%rax),%edx
 c94:	89 55 e8             	mov    %edx,-0x18(%rbp)
 c97:	48 63 d0             	movslq %eax,%rdx
 c9a:	48 8b 45 f8          	mov    -0x8(%rbp),%rax
 c9e:	48 01 c2             	add    %rax,%rdx
 ca1:	48 8b 45 f0          	mov    -0x10(%rbp),%rax
 ca5:	0f b6 00             	movzbl (%rax),%eax
 ca8:	88 02                	mov    %al,(%rdx)
 caa:	48 8b 45 f0          	mov    -0x10(%rbp),%rax
 cae:	48 8d 50 01          	lea    0x1(%rax),%rdx
 cb2:	48 89 55 f0          	mov    %rdx,-0x10(%rbp)
 cb6:	0f b6 00             	movzbl (%rax),%eax
 cb9:	3c c3                	cmp    $0xc3,%al
 cbb:	75 d1                	jne    c8e <main+0x46>
 cbd:	48 8b 45 f8          	mov    -0x8(%rbp),%rax
 cc1:	48 89 c7             	mov    %rax,%rdi
 cc4:	e8 00 00 00 00       	callq  cc9 <main+0x81>
 cc9:	bf 00 00 00 00       	mov    $0x0,%edi
 cce:	e8 00 00 00 00       	callq  cd3 <main+0x8b>
 cd3:	b8 00 00 00 00       	mov    $0x0,%eax
 cd8:	c9                   	leaveq
 cd9:	c3                   	retq
```

위 화면은 dump 뜬 파일의 일부분 화면입니다. 위에서 redirection 을 통해 test.txt 파일에 저장하였기 때문에 test.txt 파일에서 확인할 수 있었습니다.

D_recompile_test.c 에서 add, sub, div, imul 명령어가 계속해서 반복됨을 확인할 수 있습니다. 이는 objdump 에서 Add 명령어가 나오면 83, imul 명령어가 나오면 6b 가 반복해서 나오는 것을 확인한 결과, Add 명령어는 0x83, imul 명령어는 0x6b 로 매칭됨을 확인할 수 있습니다. 저장 위치는 0xc0 이고, 세번째 자리에 위치한 숫자는 얼마만큼의 수를 덧셈이나 곱셈 연산을 수행할지를 확인하는 숫자라는 것 역시 확인할 수 있었습니다. 이를 통해 instruction 이 중복되는 명령어끼리 합쳐 줌으로써 최적화를 진행할 수 있을 것이라고 생각했고 이를 진행했습니다.

다음으로 강의자료에서 To-do List 에 맞게 진행하였습니다. 먼저 shared memory 에서 컴파일 된 코드에 접근하고 code section 의 함수는 마음대로 수정할 수 없기 때문에 compiled_code 에 권한을 부여했습니다. mmap 함수를 사용하여 읽기와 쓰기가 가능한 공유 메모리를 할당합니다.

```c
void drecompile_init(uint8_t *func)
{
    compiled_code = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, 0, 0);
}
```

```c
void* drecompile(uint8_t* func)
{
    compiled_code = func;

    #ifdef dynamic
```

compiled_code 가 func 가 가리키는 메모리 주소를 가리키도록 하는 것이므로, 이후에 compiled_code 를 통해 메모리에 접근하면 func 가 가리키는 메모리에 접근하는 것과 동일하게 동작합니다. 이는 컴파일된 코드가 메모리에 있는 특정 위치에 위치하고 있다고 가정할 때, 해당 코드를 실행할 때 사용하는 것입니다. 여기서 compiled_code 가 함수 코드를 가리키게 되면, 이 포인터를 통해 해당 함수 코드를 호출하거나 실행할 수 있게 됩니다. 다음으로 add, sub, imul, div 연산이 중복으로 나오는 경우에는 각각의 명령어를 하나로 합쳐 최적화를 진행했습니다.

```c
    mprotect(compiled_code, PAGE_SIZE, PROT_READ | PROT_EXEC);
```

mprotect 함수를 통해 PAGE_SIZE 크기 만큼의 메모리 영역에 대하여 메모리를 읽고 메모리에서 코드를 실행 가능하게 권한을 변경했습니다.

```
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ make clean
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ gcc -o test2 D_recompile_test.c
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ ./test2
Data was filled to shared memory.
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ make
gcc -o drecompile D_recompile.c
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ ./drecompile
total execution time: 0.000001829 sec
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ make clean
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ ./test2
Data was filled to shared memory.
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ make dynamic
gcc -Ddynamic -o drecompile D_recompile.c
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ ./drecompile
total execution time: 0.000000170 sec
```

먼저 캐시 및 버퍼를 지우고 최적화하기 전의 결과를 확인합니다. 정확한 비교를 진행하기 위해 캐시 및 버퍼를 다시 지우고 최적화 후의 결과를 확인합니다.

```bash
$ test_50.sh
#!/bin/bash
for var in {1..50}
do
    make clean && make && ./test2 && ./drecompile >> result.txt;
done

for var in {1..50}
do
    make clean && make dynamic && ./test2 && ./drecompile >> optimal_result.txt;
done

```

두개의 실행결과를 50 번 비교하기 위해 위에서 진행한 모든 과정을 스크립트를 따로 작성하여 결과를 txt 파일에 따로 작성하도록 하였습니다.

```
os2019202005@ubuntu:~/Downloads/linux-4.19.67/os4/4-2$ bash test_50.sh
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
[sudo] password for os2019202005:
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
echo 3 | sudo tee /proc/sys/vm/drop_caches
3
gcc -o drecompile D_recompile.c
Data was filled to shared memory.
rm -rf D_recompile D_recompile
sync
```

위에서 작성한 스크립트를 실행한 화면입니다.

<최적화 전 경과>

```
total execution time : 0.000001150 sec
total execution time : 0.000001430 sec
total execution time : 0.000001420 sec
total execution time : 0.000001110 sec
total execution time : 0.000001210 sec
total execution time : 0.000001140 sec
total execution time : 0.000001210 sec
total execution time : 0.000001230 sec
total execution time : 0.000001120 sec
total execution time : 0.000001190 sec
total execution time : 0.000001390 sec
total execution time : 0.000001190 sec
total execution time : 0.000001320 sec
total execution time : 0.000001110 sec
total execution time : 0.000001120 sec
total execution time : 0.000001210 sec
total execution time : 0.000001170 sec
total execution time : 0.000001120 sec
total execution time : 0.000001430 sec
total execution time : 0.000001250 sec
total execution time : 0.000001290 sec
total execution time : 0.000001170 sec
total execution time : 0.000001180 sec
total execution time : 0.000001290 sec
total execution time : 0.000001230 sec
total execution time : 0.000001160 sec
total execution time : 0.000001090 sec
total execution time : 0.000001170 sec
total execution time : 0.000001060 sec
total execution time : 0.000001420 sec
total execution time : 0.000001180 sec
total execution time : 0.000001150 sec
total execution time : 0.000001300 sec
total execution time : 0.000001110 sec
total execution time : 0.000001290 sec
total execution time : 0.000001380 sec
total execution time : 0.000001190 sec
total execution time : 0.000001290 sec
total execution time : 0.000001180 sec
total execution time : 0.000001380 sec
total execution time : 0.000001080 sec
total execution time : 0.000001140 sec
total execution time : 0.000001180 sec
total execution time : 0.000001080 sec
total execution time : 0.000001160 sec
total execution time : 0.000001430 sec
total execution time : 0.000001160 sec
total execution time : 0.000001470 sec
```

<최적화 후 결과>

```
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000190 sec
total execution time : 0.000000190 sec
total execution time : 0.000000200 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000170 sec
total execution time : 0.000000180 sec
total execution time : 0.000000200 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000170 sec
total execution time : 0.000000180 sec
total execution time : 0.000000170 sec
total execution time : 0.000000170 sec
total execution time : 0.000000180 sec
total execution time : 0.000000230 sec
total execution time : 0.000000200 sec
total execution time : 0.000000180 sec
total execution time : 0.000000170 sec
total execution time : 0.000000180 sec
total execution time : 0.000000230 sec
total execution time : 0.000000200 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000160 sec
total execution time : 0.000000190 sec
total execution time : 0.000000180 sec
total execution time : 0.000000170 sec
total execution time : 0.000000180 sec
total execution time : 0.000000230 sec
total execution time : 0.000000190 sec
total execution time : 0.000000220 sec
total execution time : 0.000000230 sec
total execution time : 0.000000170 sec
total execution time : 0.000000200 sec
total execution time : 0.000000180 sec
total execution time : 0.000000180 sec
total execution time : 0.000000220 sec
total execution time : 0.000000180 sec
total execution time : 0.000000190 sec
```

스크립트에 작성한대로 txt 파일에 저장된 것을 확인할 수 있으며 이를 표로 나타내고 평균을 구했습니다.

최적화 전 결과		최적화 후 결과
0.00000115		0.00000018
0.00000143		0.00000018
0.00000142		0.00000018
0.00000111		0.00000018
0.00000121		0.00000019
0.00000114		0.00000019
0.00000121		0.0000002
0.00000123		0.00000018
0.00000112		0.00000018

0.00000119		0.00000017
0.00000139		0.00000018
0.00000119		0.0000002
0.00000132		0.00000018
0.00000111		0.00000018
0.00000112		0.00000017
0.00000121		0.00000018
0.00000117		0.00000017
0.00000112		0.00000017
0.00000143		0.00000018
0.00000125		0.00000023
0.00000129		0.0000002
0.00000117		0.00000018
0.00000118		0.00000017
0.00000129		0.00000018
0.00000123		0.00000023
0.00000116		0.0000002
0.00000109		0.00000018
0.00000117		0.00000018
0.00000106		0.00000018
0.00000142		0.00000018
0.00000118		0.00000018
0.00000115		0.00000018
0.0000013		0.00000016
0.00000111		0.00000019
0.00000129		0.00000018
0.00000138		0.00000017
0.00000119		0.00000018
0.00000129		0.00000023
0.00000118		0.00000019
0.00000138		0.00000022
0.00000108		0.00000023

0.00000114		0.00000017
0.00000118		0.0000002
0.00000108		0.00000018
0.00000116		0.00000018
0.00000143		0.00000022
0.00000116		0.00000018
0.00000147		0.00000019
0.00000115		0.00000023
0.00000115		0.00000019
0.0000012206	평균	0.000000188

위 표에서 두 결과의 평균을 비교했을 때 최적화 후의 실행 시간이 대략 6.5 배 정도 빨라진 것을 확인할 수 있습니다.

고찰

4-1 과제에서 처음에 코드와 데이터 영역의 처음과 끝이 start_code, end_code, start_data, end_data 이렇게 이루어진 것을 확인하고 힙영역을 출력할 때 start_brk, end_brk 로 접근하여 오류가 났습니다. 끝을 brk 로 접근했어야 하는데 위에서 정보들을 잘 찾아 놓고 제 멋대로 접근하려 해서 오류가 났었는데 이 점은 에러 메시지에서 빠르게 오류 원인에 대해 확인할 수 있어서 금방 해결할 수 있었습니다. 그리고 위 정보들을 찾기 위해서 이번에도 cscope 를 사용했는데 이제는 많이 익숙해져서 금방 찾을 수 있었습니다.

4-2 과제를 진행하면서 공유 메모리에 대한 개념에 대해서 다시 공부해볼 수 있었던 좋은 기회였습니다. 저번학기에서 시스템프로그램이 수업에서 처음 배우고 이번학기에서 운영체제 수업 때 다시 배웠는데 저번 학기 때 상대적으로 많이 어려워했던 기억이 있어서 좀 더 자세히 공부하려고 노력했습니다. 그리고 동적 재컴파일이라는 개념에 대해서는 처음 접했는데 이는 프로그램 실행 중에 일부를 다시 컴파일하여 생성된 코드를 최적화할 수 있는 기능입니다.

일반적으로 컴파일은 프로그램이 실행되기 전에 이루어지는 작업이라고 알고 있었는데 이런 기술도 있다는 점을 이번 과제를 통해 알게 되어 신기했습니다. 그래서 이에 대해 좀 더 찾아보았는데 이 동적 재컴파일은 주로 에뮬레이터나 가상환경 같은 특정 환경에서만 활용된다고 한다는 것도 알았습니다.

마지막으로 리눅스를 사용하면서 처음으로 스크립트를 새로 작성하여 실행까지 해보았는데 처음 해보는 내용에 비해 어렵지 않았으며 필요한 상황이 또 있다면 유용하게 잘 사용할 것 같다고 생각했습니다.

Reference

강의자료 2023-2_OSLab_11_ Shared Memory

강의자료 2023-2_OSLab_Assignment_4

위키피디아 동적 재컴파일

https://ko.wikipedia.org/wiki/%EB%8F%99%EC%A0%81_%EC%9E%AC%EC%BB%B4%ED%8C%8C%EC%9D%BC

스크립트 실행하기

https://gracefulprograming.tistory.com/109