로고이(가) 표시된 사진

자동 생성된 설명

운영체제

Assignment 4 과제

수업 명 : 운영체제

과제 이름 : assignment4

담당 교수님 : 최상호 교수님

학 번 : 2019202005

이 름 : 남종식

**Introduction**

4-1과제는 PID를 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 2차 과제에서 작성한ftrace 시스템 콜을 file\_varea로 wrapping 하여 사용합니다. 위 정보들을 출력하기 위해서 task\_struct에 대해 잘 살펴보아야 합니다.

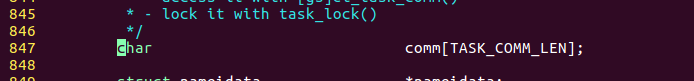
4-2과제는 공유 메모리에 존재하는 코드에 대해서 최적화를 진행하여 최적화 후의 실행결과를 최적화 전의 실행결과와 비교해야 합니다. 불필요하게 반복되는 과정을 없앤 후 dynamic recompilation을 통해 최적화를 진행합니다.

**Result**

**4-1과제**

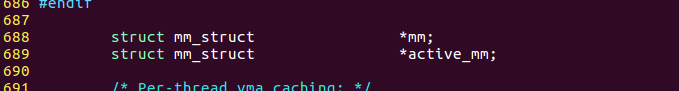
먼저 과제에서 요구하는 프로세스의 정보에 대해서 출력하기 위해서 task\_struct에서 찾아보았습니다.

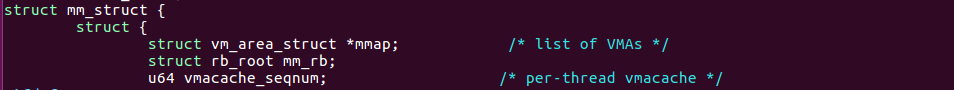
**프로세스의 이름과 pid**

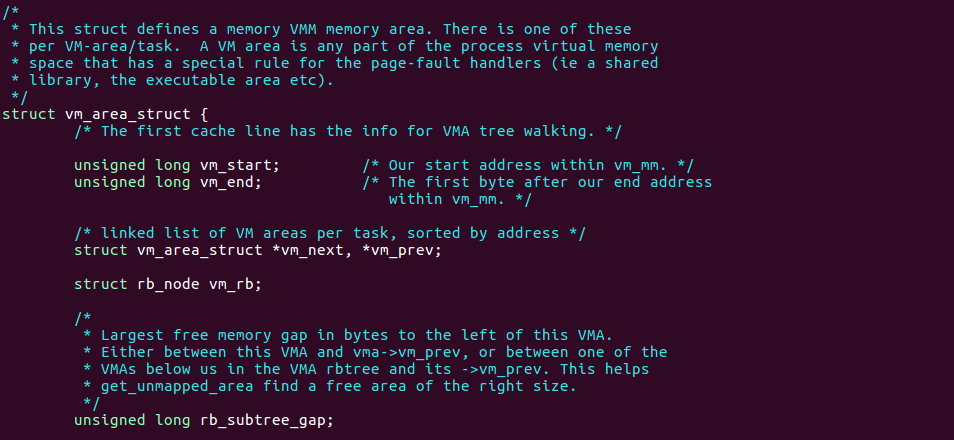
****

Comm을 통해 프로세스의 이름을 알 수 있습니다.

**정보가 위치하는 가상 메모리 주소**

****

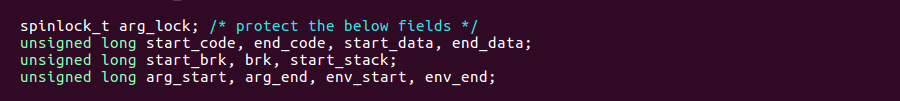
****



struct mm\_struct은 리눅스 커널에서 메모리 관리 정보를 담는 구조체입니다. 이 구조체는 각각의 프로세스에 대한 메모리 관리 정보를 저장합니다.

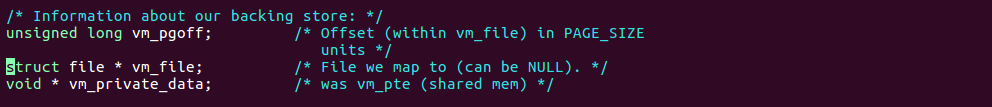
Mmap은struct vm\_area\_struct 연결 리스트를 가리키는 포인터입니다. 각각의 vm\_area\_struct 구조체는 가상 주소 공간에서 특정한 메모리 영역에 대한 정보를 저장합니다. 이 리스트를 통해 프로세스의 가상 주소 공간에 할당된 메모리 영역들을 순회할 수 있습니다.

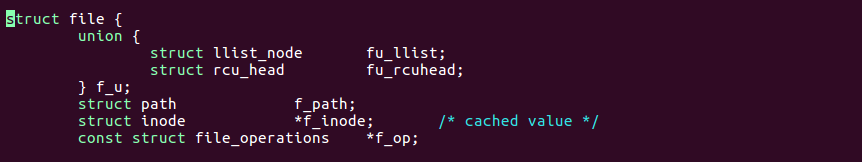
**프로세스의 데이터 주소, 코드 주소, 힙 주소**



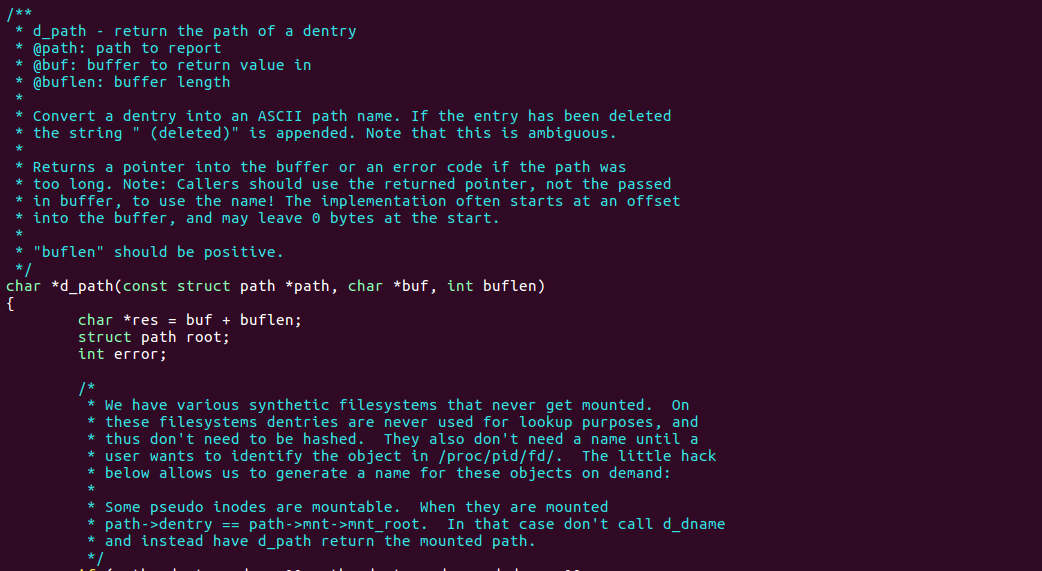
start\_code, end\_code, start\_data, end\_data는 실행 코드와 데이터의 시작과 끝을 가리키는 값들입니다. 이 정보는 프로세스의 메모리 레이아웃을 나타냅니다. start\_brk, brk는 힙 영역의 시작과 끝을 가리키는 값입니다. 동적으로 할당되는 메모리가 여기에 위치합니다.

**정보의 원본 파일의 전체 경로**

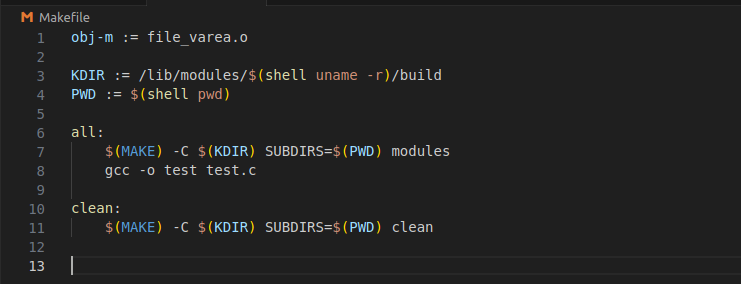




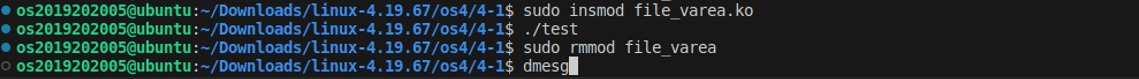
mnt는 파일 시스템을 나타내는 구조체로, 해당 파일의 마운트 정보를 가리킵니다. dentry는 파일이나 디렉토리를 나타내는 구조체로, 디렉토리 엔트리 정보를 가리킵니다



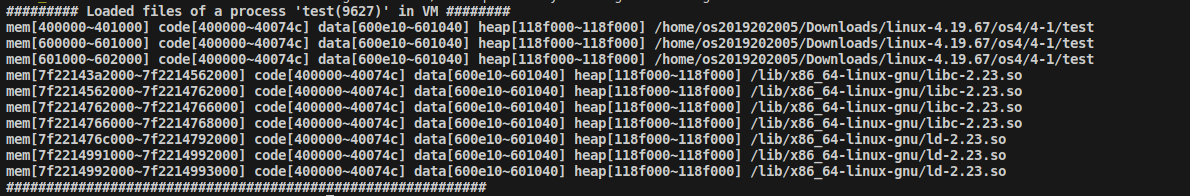
d\_path 함수는 struct path를 받아 파일의 경로를 문자열로 변환하여 반환합니다. 이 함수를 사용하여 파일의 전체 경로를 알아낼 수 있습니다. 예를 들어, 파일이 어떤 디렉터리에 위치하고 있는지를 확인하거나, 파일 시스템의 경로를 문자열로 얻고 싶을 때 사용됩니다.



테스트용 파일도 같이 컴파일 되도록 Makefile을 작성하였습니다.



모듈 적재 후 실행한 다음 삭제 후 출력을 진행했습니다.

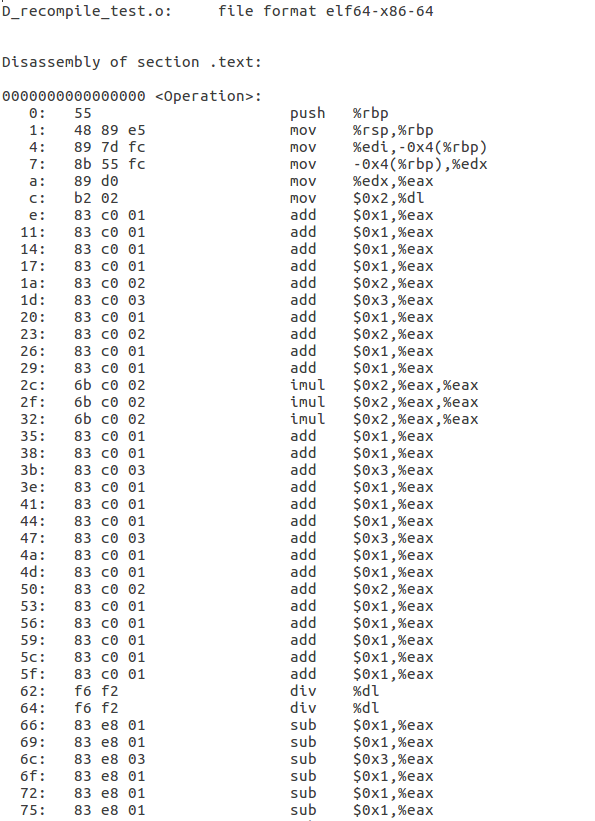
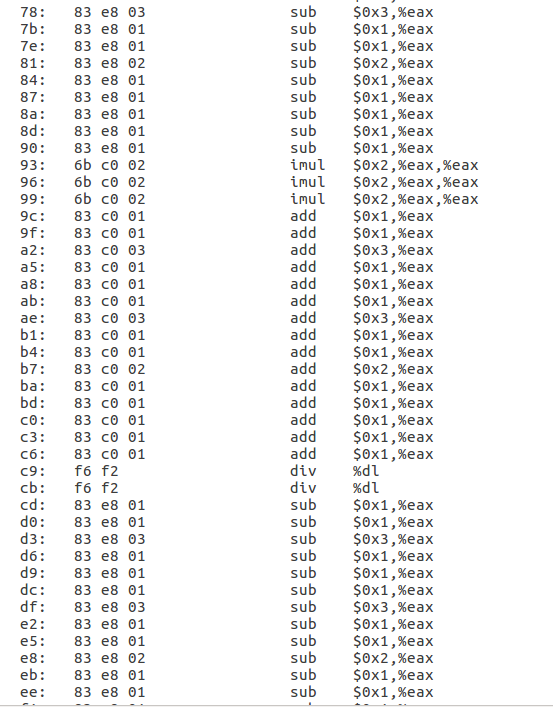


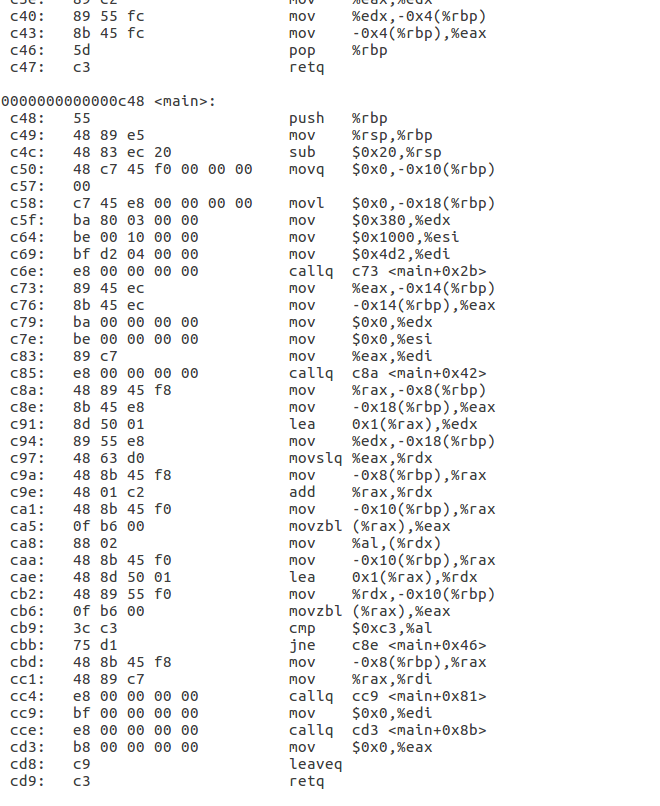
강의 자료 형식에 맞게 과제에서 요구하는 프로세스의 정보를 잘 출력하는 모습을 확인할 수 있습니다.

**4-2과제**



다음은 objdump가 뜨는 과정입니다. 먼저 gcc –c D\_recompile\_test.c을 통해 Object file을 생성 후, objdump –d D\_recompile\_test.o 을 통해, D\_recompile\_test.o file을 disassemble하여 objdump 결과가 출력되는데, 이 부분은 redirection으로 test.txt에 저장했습니다.

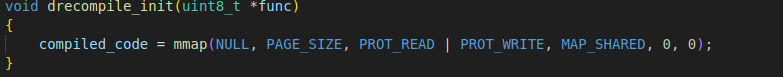
 

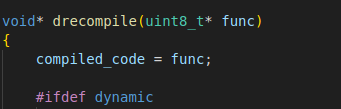


위 화면은 dump뜬 파일의 일부분 화면입니다. 위에서 redirection을 통해 test.txt파일에 저장하였기 때문에 test.txt파일에서 확인할 수 있었습니다.

D\_recompile\_test.c에서 add, sub, div, imul 명령어가 계속해서 반복됨을 확인할 수 있습니다. 이는 objdump에서 Add 명령어가 나오면 83, imul 명령어가 나오면 6b가 반복해서 나오는 것을 확인한 결과, Add 명령어는 0x83, imul 명령어는 0x6b로 매칭됨을 확인할 수 있습니다. 저장 위치는 0xc0이고, 세번째 자리에 위치한 숫자는 얼마만큼의 수를 덧셈이나 곱셈 연산을 수행할지를 확인하는 숫자라는 것 역시 확인할 수 있었습니다. 이를 통해 instruction이 중복되는 명령어끼리 합쳐 줌으로써 최적화를 진행할 수 있을 것이라고 생각했고 이를 진행했습니다.

다음으로 강의자료에서 To-do List에 맞게 진행하였습니다. 먼저 shared memory에서 컴파일 된 코드에 접근하고 code section의 함수는 마음대로 수정할 수 없기 때문에 compiled\_code에 권한을 부여했습니다. mmap 함수를 사용하여 읽기와 쓰기가 가능한 공유 메모리를 할당합니다.

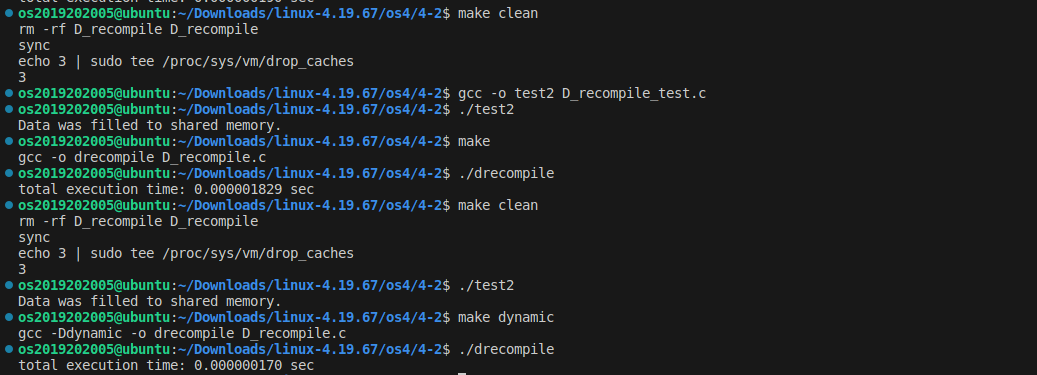




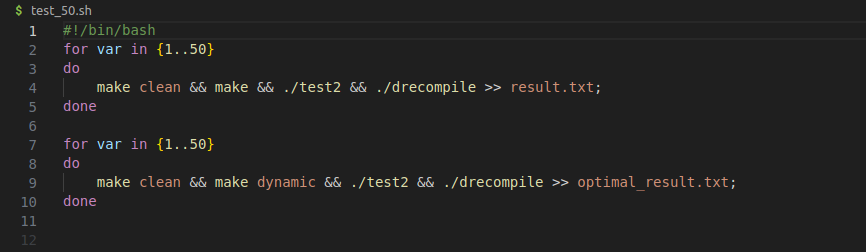
compiled\_code가 func가 가리키는 메모리 주소를 가리키도록 하는 것이므로, 이후에 compiled\_code를 통해 메모리에 접근하면 func가 가리키는 메모리에 접근하는 것과 동일하게 동작합니다. 이는 컴파일된 코드가 메모리에 있는 특정 위치에 위치하고 있다고 가정할 때, 해당 코드를 실행할 때 사용하는 것입니다. 여기서 compiled\_code가 함수 코드를 가리키게 되면, 이 포인터를 통해 해당 함수 코드를 호출하거나 실행할 수 있게 됩니다. 다음으로 add, sub, imul, div 연산이 중복으로 나오는 경우에는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



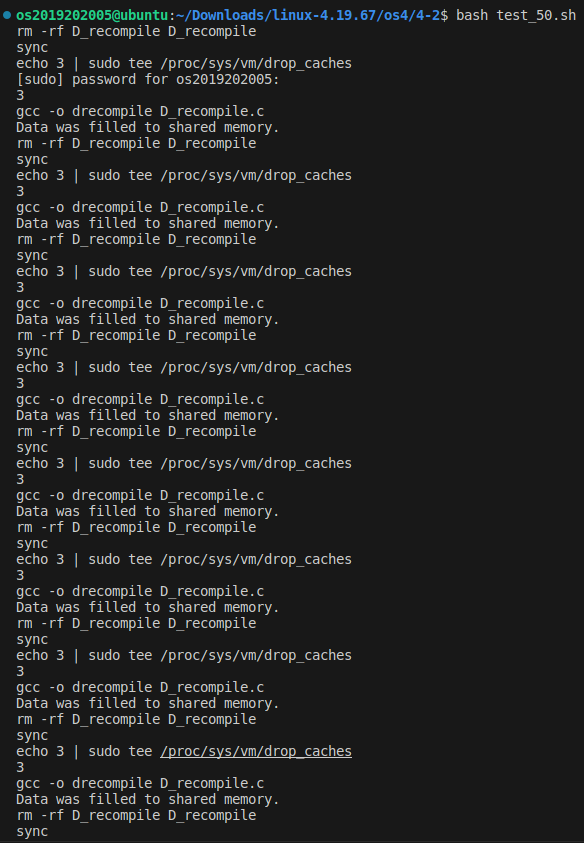
mprotect함수를 통해 PAGE\_SIZE크기 만큼의 메모리 영역에 대하여 메모리를 일고 메모리에서 코드를 실행 가능하게 권한을 변경했습니다.



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

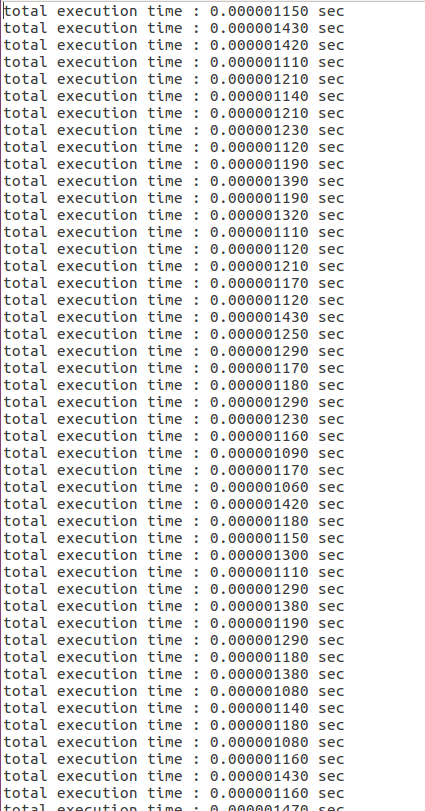
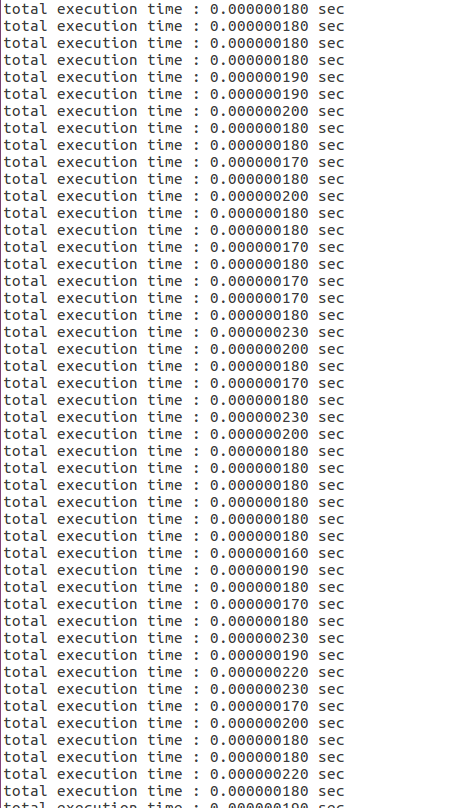


두개의 실행결과를 50번 비교하기 위해 위에서 진행한 모든 과정을 스크립트를 따로 작성하여 결과를 txt파일에 따로 작성하도록 하였습니다.



위에서 작성한 스크립트를 실행한 화면입니다.

<최적화 전 경과> <최적화 후 결과>

스크립트에 작성한대로 txt파일에 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 6.5배 정도 빨라진 것을 확인할 수 있습니다.

**고찰**

4-1 과제에서 처음에 코드와 데이터 영역의 처음과 끝이 start\_code, end\_code, start\_data, end\_data이렇게 이루어진 것을 확인하고 힙영역을 출력할 때start\_brk, end\_brk로 접근하여 오류가 났습니다. 끝을 brk로 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 cscope를 사용했는데 이제는 많이 익숙해져서 금방 찾을 수 있었습니다.

4-2과제를 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>