운영체제 실습

[Assignment #4]

Class : 목3

Professor : 최상호 교수님

Student ID : 2021202045

Name : 김예은

Introduction

이번 4차 프로젝트는 2차 과제에서 구현한 system call 336번을 후킹하여 프로세스의 가상메모리의 정보를 출력하는 부분(3-1)과 memory swapping policy 4가지 (optimal, fifo, lru, clock) 알고리즘을 구현하는 부분(3-2)으로 나뉘어져 있다. 가상 메모리의 경우 프로세스의 모든 정보를 가지고 있는 task_struct의 mm_struct 부분을 통해 접근할 수 있으며 이를 통해 가상 메모리 주소, 데이터주소, 코드주소, 힙주소, 정보의 원본 파일의 전체경로를 출력한다. 3-2의 경우 input file로부터 page frame의 크기(수)와 page reference string을 읽어와 이를 바탕으로 알고리즘 별 성능을 평가한다.

Result

(4-1)

```
[72118.889808] ######## Loaded files of a process 'test(49395)' in VM ########
[72118.889812] mem(55a8fcfdc000~55a8fcfdd000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /home/os2021202045/ASS4/4-1/test
[72118.889821] mem(55a8fcfdd000~55a8fcfde000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /home/os2021202045/ASS4/4-1/test
[72118.889827] mem(55a8fcfde000~55a8fcfdf000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /home/os2021202045/ASS4/4-1/test
[72118.889832] mem(55a8fcfdf000~55a8fcfe0000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /home/os2021202045/ASS4/4-1/test
[72118.889837] mem(55a8fcfe0000~55a8fcfe1000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /home/os2021202045/ASS4/4-1/test
[72118.889842] mem(7f2c0fe67000~7f2c0fe69000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72118.889848] mem(7f2c0fe89000~7f2c10001000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72118.889882] mem(7f2c10001000~7f2c1004f000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72118.889887] mem(7f2c1004f000~7f2c10053000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72118.889892] mem(7f2c10053000~7f2c10055000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72118.889897] mem(7f2c1006b000~7f2c1006c000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72118.889909] mem(7f2c1006c000~7f2c1008f000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72118.889915] mem(7f2c1008f000~7f2c10097000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72118.889979] mem(7f2c10098000~7f2c10099000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72118.889983] mem(7f2c10099000~7f2c1009a000) code(55a8fcfdc000~55a8fcfdd215) data(55a8fcfdfdb0~55a8fcfe0010) heap(55a8fe5d9000~55a8fe5d9000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72118.889987] ############################################################

[72167.914628] ######## Loaded files of a process 'test(49398)' in VM ########
[72167.914631] mem(561074b0c000~561074b0d000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /home/os2021202045/ASS4/4-1/test
[72167.914638] mem(561074b0d000~561074b0e000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /home/os2021202045/ASS4/4-1/test
[72167.914641] mem(561074b0e000~561074b0f000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /home/os2021202045/ASS4/4-1/test
[72167.914644] mem(561074b0f000~561074b10000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /home/os2021202045/ASS4/4-1/test
[72167.914647] mem(561074b10000~561074b11000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /home/os2021202045/ASS4/4-1/test
[72167.914651] mem(7f51ca7c5000~7f51ca7e7000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72167.914654] mem(7f51ca7e7000~7f51ca95f000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72167.914658] mem(7f51ca95f000~7f51ca9ad000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72167.914661] mem(7f51ca9ad000~7f51ca9b1000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72167.914664] mem(7f51ca9b1000~7f51ca9b3000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/libc-2.31.so
[72167.914667] mem(7f51ca9c9000~7f51ca9ca000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72167.914670] mem(7f51ca9ca000~7f51ca9ed000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72167.914674] mem(7f51ca9ed000~7f51ca9f5000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72167.914677] mem(7f51ca9f6000~7f51ca9f7000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72167.914680] mem(7f51ca9f7000~7f51ca9f8000) code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) /usr/lib/x86_64-linux-gnu/ld-2.31.so
[72167.914683] ############################################################
```

task_struct의 vm_area_struct를 가리키는 mm_struct를 통해 가상 메모리에 접근했다. vm_start, vm_end를 통해 가상 메모리의 페이지 시작과 끝 부분을 출력하게 했고, start_code와 end_code를 통해 code area의 주소를 출력하게끔 했으며, start_data, end_data를 통해 data 영역 주소를, start_brk와 brk를 통해 heap area의 주소 영역을 출력하게끔 했다. 원본 파일의 경로를 찾기 위해서 d_path라는 함수를 사용했으며, mmap의 vm_file의 f_path를 찾아내어 256(파일이름사이즈크기) 크기의 p_buf를 만들어 파일명을 복사하게끔 했다. 가상 메모리는 vm_next, vm_prev를 통해 다음과 이전 페이지를 연결해놓았는데 vm_next가 없다면 마지막 노드라는 의미이므로 그때까지 가상메모리를 scan하도록 while문을 짰다.

두 번째 결과화면을 토대로 설명을 해보자면, 첫 줄에 code(561074b0c000~561074b0d215) data(561074b0fdb0~561074b10010) heap(56107685c000~56107685c000) 이고 mem은 각각 코드, 데이터, 힙 영역을 감싸는 영역인 mem(561074b0d000~561074b0e000),mem(561074b0e000~561074b0f000),mem(561074b0f000~561074b10000),mem(561074b10000~561074b11000) 으로 출력되는 것을 확인할 수 있다.

파일의 경로 또한, 우선 내가 실행한 file_varea.c파일이 있는 폴더의 경로를 출력해주고 그 다음부터는 프로세스가 실행중에 동적으로 매핑한 파일들로, libc.so는 C표준 라이브러리, ld-linux.so는 동적 링커이다.

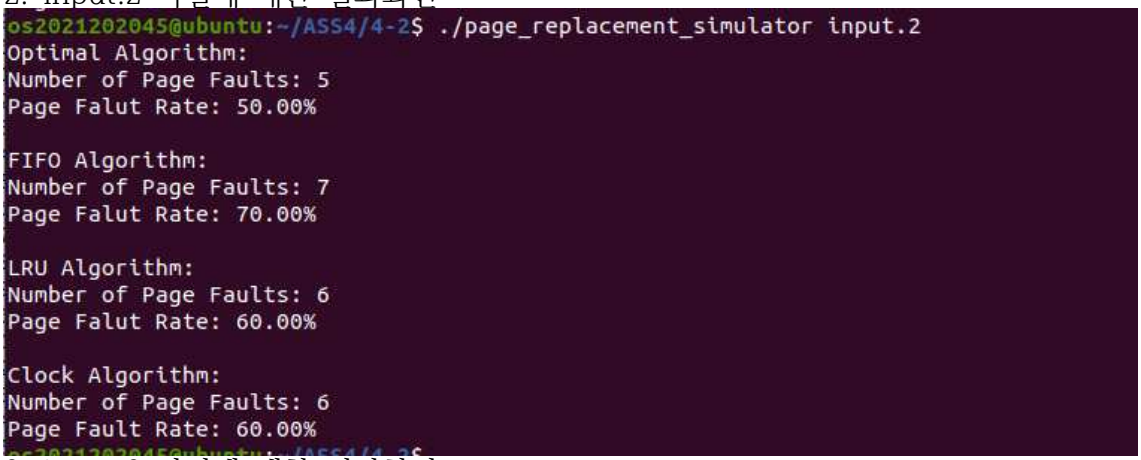
파일 매핑이란, 실제 파일을 가상 메모리에 매핑하여 파일의 내용을 메모리에서 접근할 수 있게 하는 방식으로 디스크에 있는 파일 데이터를 물리->가상 메모리 주소로 연결한다. 익명 매핑이란, 파일과 연결이 되지 않고, 프로그램 실행 중 동적 메모리를 관리한다.

맨처음에는 vm->file이 null 아닐 때 주소를 출력하게끔하는 코드가 없었는데, 출력을 할 때 익명 매핑이 된 부분들에서 오류가 생기거나 너무 많이 결과값이 출력되었다. 또한 힙 영역은 프로그램이 실행중일 때 동적으로 매핑되는 익명 매핑 부분인데 file 매핑 기반으로 코드를 짰을 때 잘 실행되는 부분은 아직도 왜인지 모르겠다.

(4-2)

1. input.1 파일에 대한 결과화면

2. input.2 파일에 대한 결과화면



3. input.3 파일에 대한 결과화면

정적배열로 코드를 구현했기 때문에 page frame의 수는 1000으로 가정해서 배열을 선언했다. LRU, Optimal의 경우 page frame에 저장되어 있는 page number가 다음에 언제 참조될지 또는 이전에 언제 참조 되었는지 위치 index를 저장하기 위해 when_recall이라는 배열을 통해 이를 저장하게끔 하였고, clock의 경우 use bit를 page frame이 기억하고 있어야 하므로 use_bit 배열을 하나 만들어줘서 replacement function을 따로 구현했다. 하나의 string마다 page frame 크기만큼 같은 숫자가 있는 지 scan하는 로직을 네 알고리즘 모두 사용하였다.

알고리즘 성능의 경우 input.1에서 page fault 비율로 계산 해 봤을 때,
FIFO > LRU > CLCOK > OPTIMAL 알고리즘 순서대로 page fault가 덜 났다.
input.2에서는 FIFO > LRU = CLOCK > OPTIMAL, input.3에서는 FIFO = LRU > CLOCK > OPTIMAL 순서대로 page fault가 일어났다.

즉, 어떤 경우에서든지, fifo가 가장 page fault가 많았고, optimal이 가장 page fault가 덜 난다. 하지만 optimal의 경우, 미래의 reference string을 안다는 가정 하에 짠 알고리즘으로 실제로는 다른 알고리즘을 평가하는 기준으로 쓰인다.

고찰

4-1의 경우, 맨 처음 mem과 data, heap, code 가상 메모리 출력 부분이 이해가 안 되어 이해하는 데 시간이 좀 소요가 되었다. 또한 위에서 적었듯이 맨 처음에 vm_file 부분을 이용하지않고 코드를 짤 때 익명 매핑이 된 가상 메모리까지 출력이 되어 결과가 많고, 의도한 대로 되지 않았지만, vm_file을 통해 파일 매핑이 된 가상 메모리 주소만 출력하게끔 해주었다. 4-2의 경우, 하나의 string마다 page frame 크기만큼 같은 숫자가 있는 지 scan하는 로직은 네 알고리즘 모두 같게 사용하고 이 부분만 만들면 replacement function만 바꿔주면 되어서 로직자체는 간단했지만, input.2, input.3에서 계속 예외 상황이 나타나서 코드 수정이 많이 일어났다. 특히 맨처음 빈 배열에 fault가 일어나는 cold-start 부분을 따로 로직을 빼서 구현했다가 cold-start 때 hit이 일어나는 input.3에서 예외가 발생해 다시 코드를 수정했다. clock algorithm의 경우 use bit 개념이 생소했지만 한번 이해하니 hit, fault가 일어날 때 use bit를 update해주는 로직만 추가하면 되어서 생각보다 간단했다.