**OS (Operating System)**

**Second Term**

**작업환경: ubuntu-14.04.4-desktop-amd64,**

**VirtualBox-5.0.16-105871-Win**

**컴퓨터학과**

**2014210057 최준명**

**제출 일자: 2016.06.10**

1. **주제**

가상 메모리의 개념 이해 및 procfs를 활용하여 메모리 정보를 출력하는 프로그램 구현

1. **목적**

이번 과제는 procfs을 활용 하여 현재 구동중인 프로세스의 메모리 정보를 출력하는 프로그램을 제작 하는 것 이다. Procfs는 /proc에 위치하고 있는 가상 파일 시스템이다. /proc 디렉토리에 들어있는 파일들은 현재 실행중인 커널 및 프로세스에 대한 정보를 담고있다.

1. **Paging**

리눅스에서 페이징 기법(Paging)이란 가상기억장치를 모두 같은 크기의 블록으로 편성하여 운용하는 기법을 의미한다. 이때의 일정한 크기를 가진 블록을 페이지(Page)라고 한다. 운영체제는 CPU는 주소 공간을 페이지 단위로 나누고 실제기억공간은 페이지 크기와 같은 프레임으로 나누어 사용한다. 즉 프레임(Frame)이란 물리 메모리를 일정된 한 크기로 나눈 블록을 의미한다.

1. **워킹셋(Working set)**

Working Set을 설명하기 위해서는 우선 두가지 개념을 먼저 설명해야 하는데 첫번째 개념은 가상주소공간으로 어플리케이션이 사용할 수 있는 최대 주소 공간을 의미하는 용어이고 두번째는 Page Faults로 자신의 주소공간에는 존재하지만 RAM에는 현재 없는 데이터나 코드에 접근을 시도하였을 경우 발생하는 현상을 의미하는데 이것이 발생하면 다른 위치에서 이 데이터를 RAM으로 가져와 계속 실행할 수 있도록 만들어줍니다. 이 행위는 MMU라는 장치가 관리하여 페이지 폴트 처리기가 이 기능을 하도록 합니다.

Working Set이란 현재사용하고 있는 프로세스 전용으로 할당된 물리적 메모리 페이지 그룹을 프로세스의 Working set이라고 합니다. 위에서 설명한 Page Fault가 발생할 때 마다 Working Set이 증가되고 여유페이지의 수가 감소하면 Working Set도 감소합니다. 메모리가 완전 부족한 상황을 만들지 않기 위해서 항상 여유공간을 만들어 둬야 하고 Working Set에서 Page를 삭제하여 여유공간으로 변경하여 주어야합니다.

물리 메모리와 가상 메모리는 ‘가’에서 언급한 Page라는 단위로 구성되는데 이번 3차과제 환경의 Page 단위는 4KBYTE였습니다.

1. **쓰래싱(Thrashing)**

리눅스환경에서 Virtual Memory를 사용하다 보면 실제 Physical Memory보다 더욱 큰 공간에서 사용할 수 있어서 효율적인데 이것이 과하게 된다면 실제 실행하는 시간보다 Page Replacement를 하는 시간이 더 많아지고 결국 CPU사용률이 떨어지게 된다. 이렇게 page Fault가 연속적으로 발생하여 실행시간보다 Page Replacement(Swap)시간이 더 큰 상황을 쓰래싱(Thrashing)이라고 한다. 즉 Swapping이 너무 자주발생하여 현재작업을 실행하는데 RAM이 부족한 경우를 의미합니다. 쓰레싱이 발생하면 시스템이 처리할 수 있는 량보다 더 많은 것을 무리하게 실행시키려하여 프로그램 처리속도가 급격히 떨어지고 극단적 경우 아무런 작업도 하지 못하게 됩니다.

이 Thrashing을 해결하기 위한 방법으로 Working Set Model(작업 설정 모델)이란 것을 적용할 수 있는데 이 Working Set Model이란 프로그램의 수행과정을 지역적 개념으로 사용하여 프로세스가 많이 참조하는 페이지를 메모리 공간의 Working Set Model로 구성합니다. 그리고 Working set의 크기가 실제 메모리 용량보다 커진다면 하나의 Process를 종료하는 방식을 의미한다.

1. **Swap 파티션**

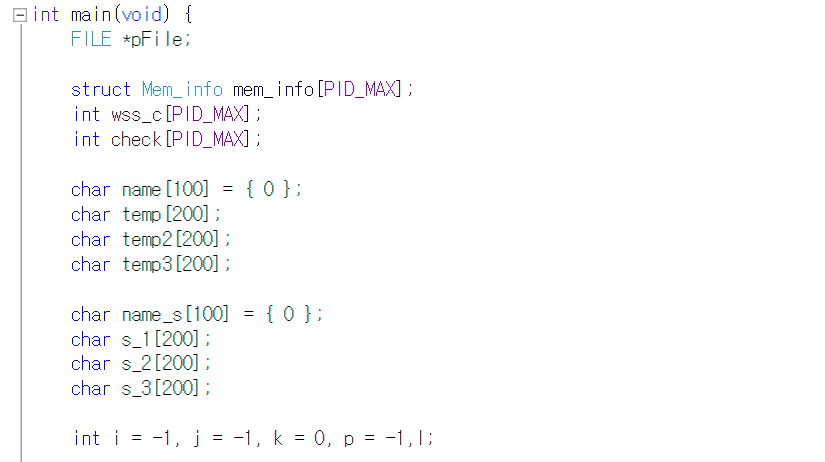
스왑 파티션(Swap Partition)는 컴퓨터 메모리(RAM)이 넘쳤을 때 이것을 막아주는 역할을 합니다. Disk의 경우 자신의 용량을 넘치는 것을 저장할 수 없지만 RAM의 경우 이 SWAP Partition이란 예비 공간을 이용하면 가능합니다. Swap Partition을 사용하면 RAM이 가득 찼을 때 보조 공간을 제공하여 빠른 속도의 메모리가 더 많은 공간을 사용할 수 있게 만들어 주지만 스왑 파티션은 동적으로 크기를 조정할 수 없어서 하드디스크의 공간을 차지하며 하드디스크의 소모율을 높혀 속도가 느려질 수도 있습니다.

1. **본인이 작성한 어플리케이션 소스 코드에 대한 설명**
2. **Struct구조체**



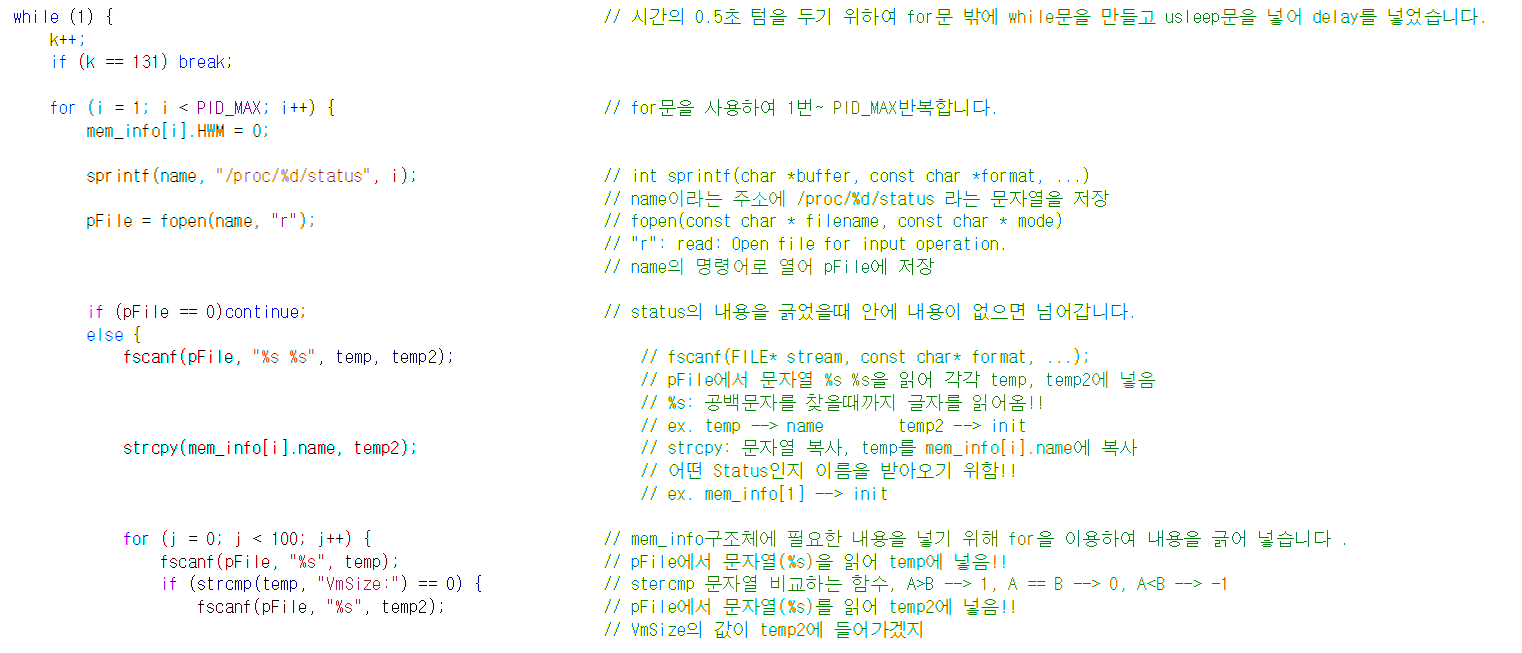
이번과제에서 편의를 위하여 구조체를 만들었는데 과제에서 뽑아낼 정보들로 구성 되어있습니다. Name, Virtual, Physical, Shared, HWM, wss으로 구성되어있습니다.

1. **변수 모음**



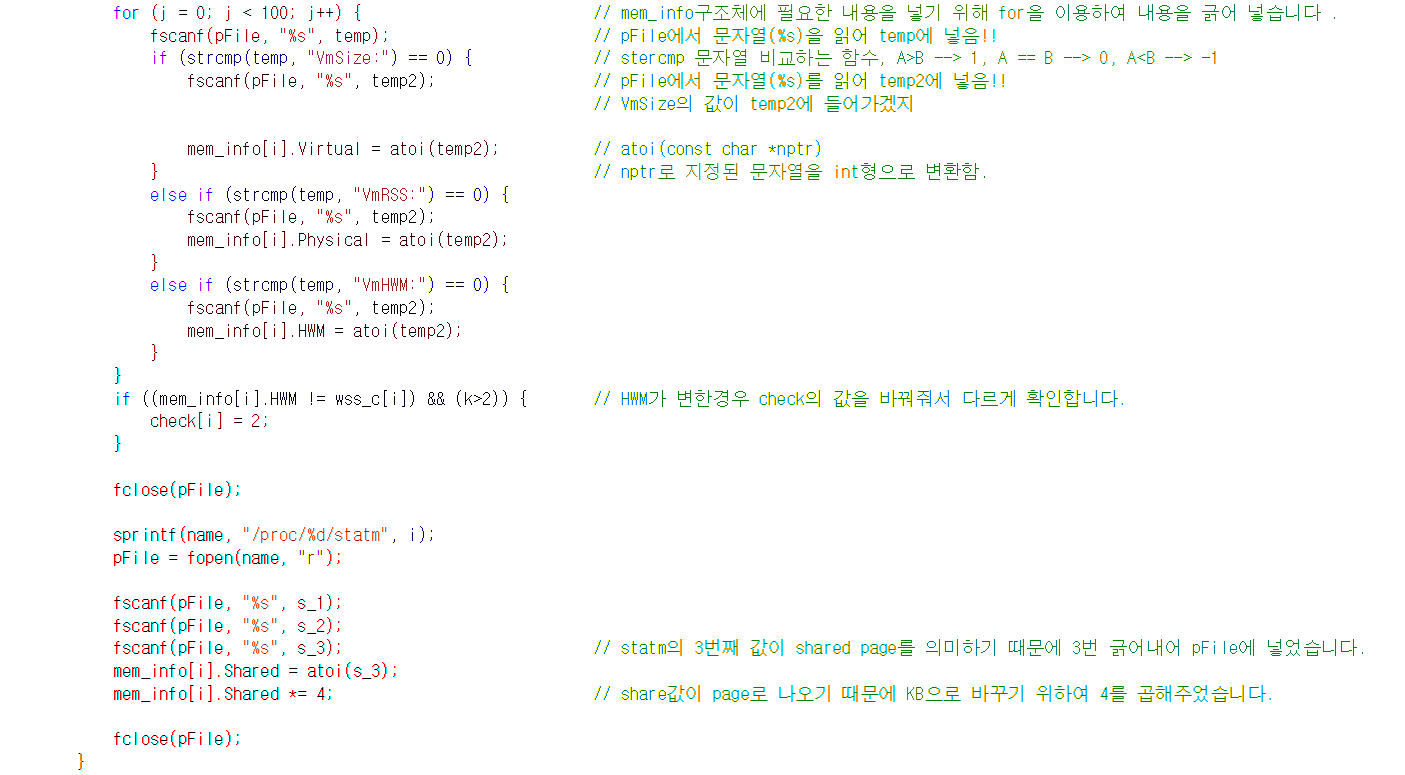
Read\_mem\_info프로그램에서 사용하는 변수 모음입니다.

1. **코딩부분\_1**



코딩의 앞부분입니다. for문을 이용하여 전체적인 코드를 PID\_MAX번 돌리고 이것에 0.5초의 텀을 주기 위해서 다시 while문으로 묶고 그부분에 usleep(500\*1000)을 넣어서 0.5초에 한번씩 수행하도록 만들었습니다. 그리고 while문이 한번돌때마다 k란 변수를 둬서 k++을 하여 count를 하였고 일정 이상(k>131)이 되면 while문을 나가서 이 부분을 나가도록 하였습니다. 1분이라 하였지만 약간의 여유를 두기 위해서 k를 131로 하여 130\*0.5(초) = 65초동안 프로그램이 돌아가도록 만들었습니다. PID\_MAX로 묶인 for안에서는 fopen을 이용하여 filesystem을 풀어서 pFile에 넣었습니다. 그래서 이부분이 비었으면 넘어가고 안에 내용이 있으면 주요 구문들을 돌렸습니다.

1. **코딩부분\_2**



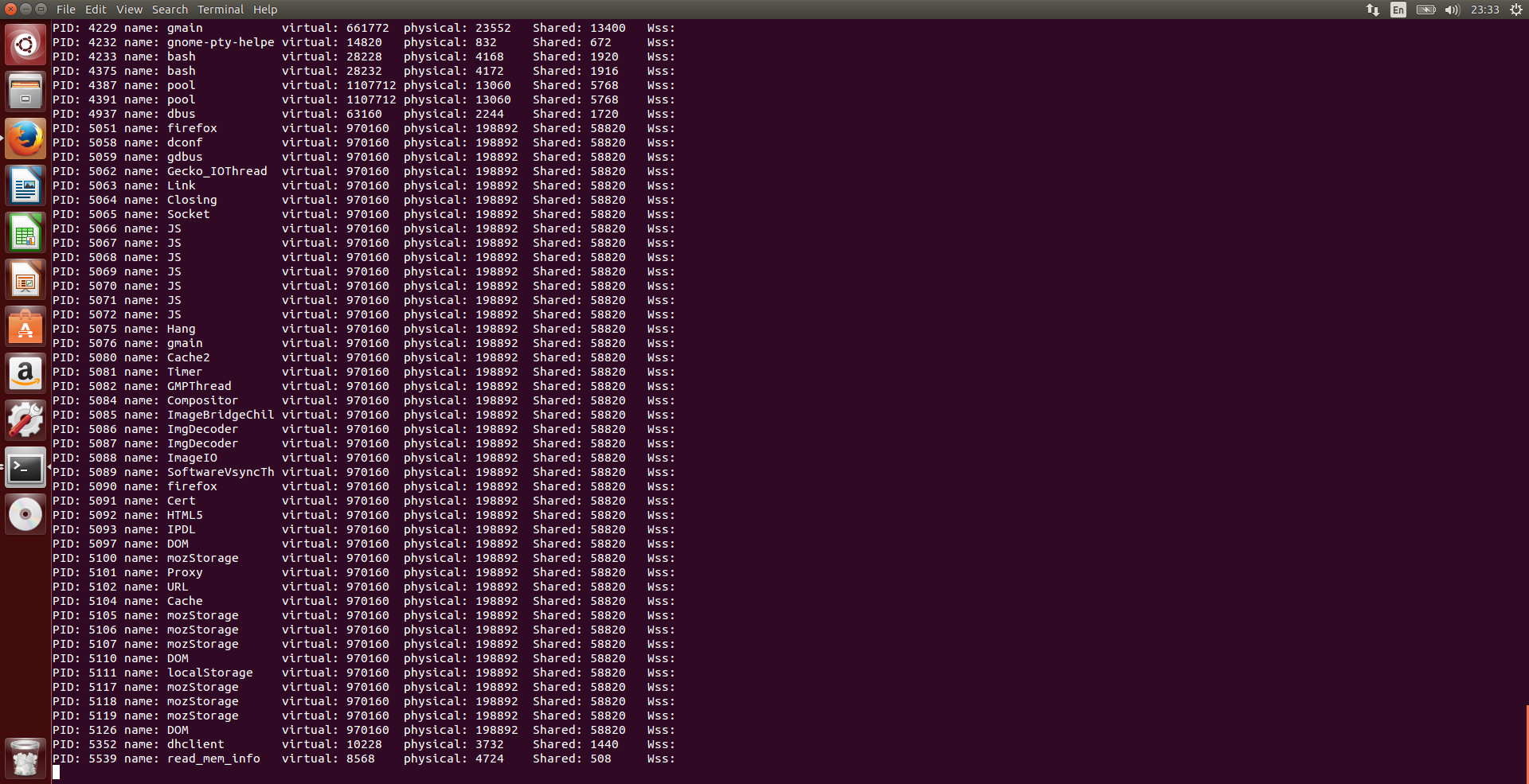
코딩의 뒷부분입니다. for문을 이용하여 이 주요부분을 100번 돌렸습니다. 이 부분에서는 각 나누어진 String과 VmSize, VmRSS, VmHWM을 확인하여 이에 대한 값을 temp2에 넣고 이것을 int형으로 바꾸어 mem\_info[i]의 구조체에 각각의 정보에 넣었습니다.

중간 이후의 부분에서는 statm을 열어서 share값을 구하여 mem\_info[i]의 구조체에 정보에 넣었습니다.

1. **실행결과 스냅샷**

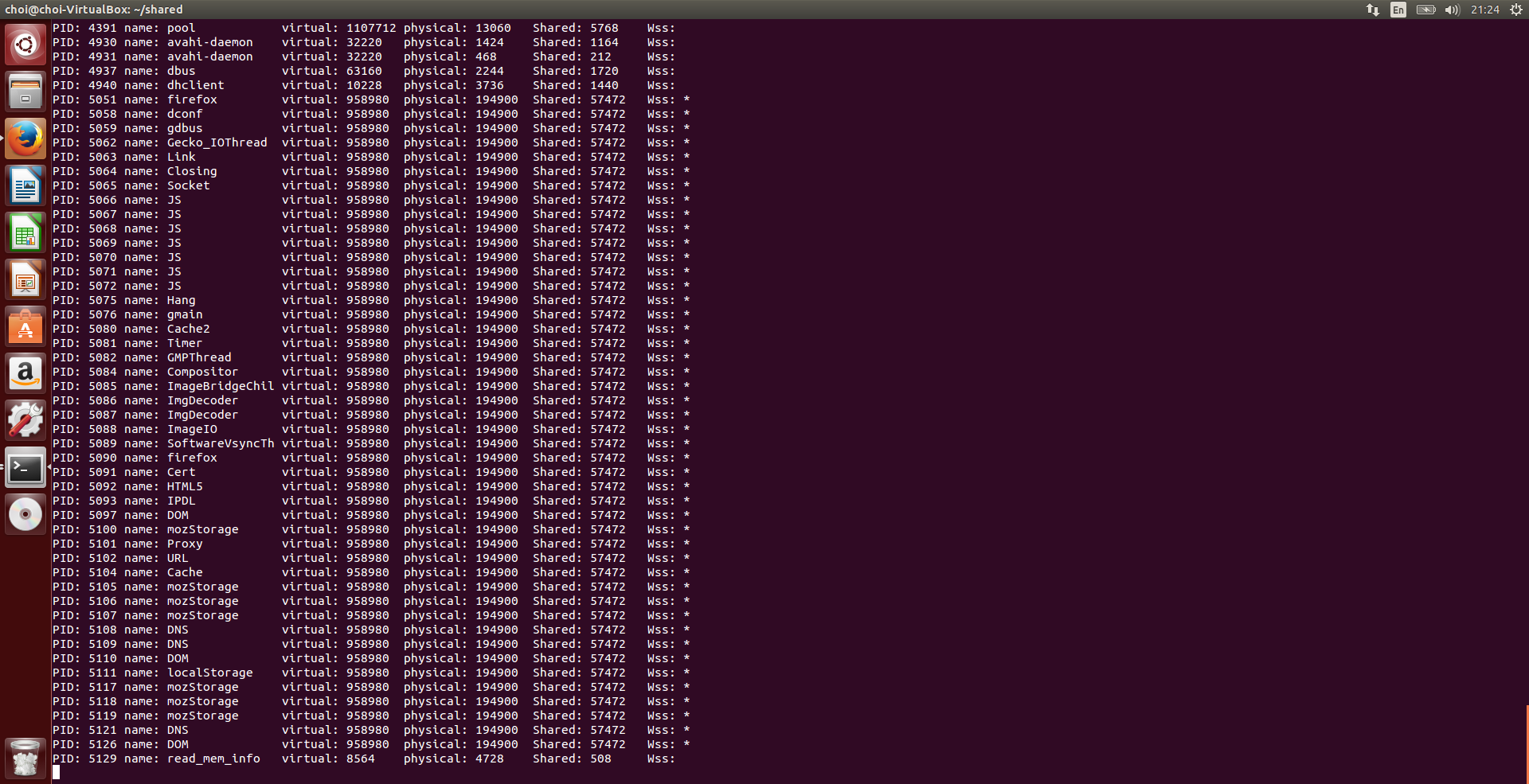
1. ./read\_mem\_info를 이용하여 Application을 실행했을 때 실행화면 입니다.

- Firefox를 키기 전



1. ./read\_mem\_info를 이용하여 Application을 실행했을 때 실행화면 입니다.

- Firefox를 킨 후



1. **어플리케이션 산출물**

****

Application 산출물의 경우 따로 스크린 샷을 찍으려 하다 내용이 너무 많고 Word에 이런 식으로 파일을 올릴 수 있어서 이렇게 올렸습니다.

1. **산출물 excel 그래프 및 분석**

우선 Init은 백그라운드 프로세스로 PID가 1로 가장 우선적으로 가장 먼저 수행하여 안정되게 수행되어 Virtual Memory와 Physical Memory가 일정하게 수행됩니다. FireFox의 경우 메모리의 사용량이 크고 Physical, Virtual Memory의 변화량이 가장 큰 프로세스로 급격한 메모리 증가량으로 인해 Fage Fault가 발생했음을 Ramspeed를 통하여 알 수 있습니다. Read\_Mem\_Info는 제가 만든 Application인데 처음 급격하게 증가한 이후에는 일정하게 사용하는 프로그램으로 Memory의 변화량이 없고 Page Fault가 거의 발 생하지 않았음을 알 수 있습니다. Ramspeed Benchmark는 컴퓨터의 성능(Cahce나 Memory)을 알려주는 Benchmark로 현재 메모리가 사용하고 있는 메모리 사용량을 확인할 수 있습니다. 이번 과제에서는 Ramspeed -b의 1~6 6가지 option을 이용하여 제 Application과 동시에 실행하였습니다.

Init이나 Firefox, Read\_Mem\_info의 경우 여러 PID가 있어 여러 파일이 실행 중이기 때문에 자주 사용하는 페이지들을 Working Set Model로 구성하여 Page Fault를 줄였는데 Ramspeed의 경우 사용하고 있는 Memory의 크기를 나타내는데 같이 사용하고 있는 파일이 없어서 공통된 Working Set Model을 거의 구성할 수 없고 Virtual Memory와 Physical Memory의 차이가 거의 나지 않음을 알 수 있습니다.

1. **물리메모리의 변화량을 토대로 워킹셋(working set)을 어떻게 정하면 좋을지 분석할 것.**

물리메모리가 급격하게 변화하는 경우 새로 들어오는 정보들이 많아지기 때문에 Page Fault가 많이 발생하게 되고 Working Set이 커져서 RAM이 이용할 수 있는 Memory의 크기를 크게 만들어 줘야합니다. 계속 급격하게 발생하게 된다면 여유 공간이 부족하게 되기 때문에 Working Set Model을 만들어서 프로그램이 많이 참조하는 페이지를 메모리 공간의 Working Set Model으로 따로 구성하여 여유 메모리의 크기를 좀 남겨 줍니다. 그리고 Working set의 크기가 실제 메모리 용량보다 커진다면 하나의 Process를 종료하는 방식을 의미한다.

1. **물리 메모리의 변화량이 메모리 영역 중 어느 영역(stack)에 위치 할지를 보고서에 기기 할 것.**

제가 생각하기에 프로그램의 물리 메모리 변화량은 RAM부분 그리고 process가 실행되면서 구성되는 Stack, Heap, DATA중 DATA부분에 존재해야 합니다. 왜냐하면 Working Set Size는 유동적으로 바로바로 변해줘야 메모리가 범람하는 문제가 발생하지 않기 때문에 접근하기에 쉽고 빠른 위치에 존재해야 하는데.

1. **필요한 Procfs의 Entry**

Virtual Memory - Status의 VmSize(Virtual Memory Size) or Statm의 첫번째 값 \* 4(page이기 때문)

Physical Memory – Status 의 VmRSS(Resident Set Size) or Statm의 두번째 값 \* 4

Shared Memory – Statm의 3번째 값 \*4 (page값 이기 때문)

Working Set Size – Status의 VmHWM(Peak Resident Set Size)의 변화량을 확인함

**과제를 수행하면서 발생한 문제점 및 해결 방법**

이번 과제는 조교님이 쉽다고 해서 방심했는데 생각보다 많이 막혀서 어려웠습니다. 우선 필요한 값을 어디서 구하는지가 어려웠습니다. 다른 것들은 명시되어 있어서 쉬웠는데 Working Set Size가 변하는지를 확인하기 위해 어떤 값을 확인해야 하는지도 어려웠고 코딩을 하는중에 Struct구조체를 2개를 만들어 Working Set Size를 확인 했었는데 이 방법을 사용하면 Segment Fault가 발생하여 원인을 찾아보았지만 결국 못 찾았고 그냥 배열을 만들어 확인하는 방법으로 해결하였습니다. 또 코딩을 하던중 File Open을 하고 File을 닫지않아서 주소의 문제가 발생하였는데 원인을 찾지 못해서 고민하다 결국 확인하고 fclose(pFile)을 사용하여 해결하였습니다. 전반적으로 내용에 대한 문제보다는 코딩적 문제가 많이 발생한 과제였습니다.

참고: Reference

위키백과: <https://ko.wikipedia.org/wiki/%ED%8E%98%EC%9D%B4%EC%A7%95>

Google: <http://egloos.zum.com/anster/v/2134653>

Red Hat Enterprise Linux 4: 시스템 관리 안내서  
<http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-isa-ko-4/s1-memory-virt-details.html>