**Operating System**

**- 가상메모리의 이해와 procfs를 통한 분석 –**

**2013210111**

**남세현**

1. **환경**

VMWare 11 에서 Fedora 14, Linux-2.6.35.6 을 사용

1. **리눅스에서 Paging의 자세한 설명**

리눅스는 임베디드, 네트워크, 정말 많은 분야에 많이 사용됩니다. 다양한 시스템을 지원하는데, 리눅스만의 페이징 기법이 그것을 도와주고 있습니다.

리눅스는 인텔의 2단계 페이징에서 1단계를 추가한 3단계 페이징 기법을 사용한다고 합니다. 현재 과제에서 사용한 2.6 커널에서 부터는 4단계 페이징도 지원합니다.

페이징 테이블을 3단계로 나뉘어서 사용하는 이유는 하나의 페이지 테이블을 사용하는 경우 테이블의 크기가 커지므로 이를 유지하기 위한 메모리 공간이 많이 필요하기 떄문입니다.

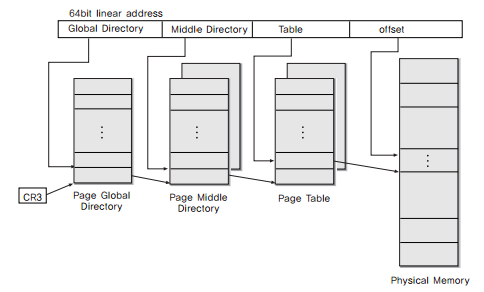


Figure 1 : 3단계 페이징

4단계 페이징 : pgd, pud, pmd, pte

페이징 단계는 커스터마이징할 수 있는데, 2단계만 사용하는 경우 pud와 pmd는 생략하고 해당 엔트리는 pgd 내의 엔트리를 가리키도록 설정됩니다. 3단계만 사용할 경우 pud를 생략합니다.

Pte는 최종 페이지의 프레임 정보를 저장하는 페이지 테이블(page table)입니다. 페이지 테이블에는 페이지의 물리 주소(프레임)과 함께 해당 페이지에 대한 접근 정보를 포함합니다.

접근 정보로는 present, accessed, read/write, dirty입니다.

1. **워킹 셋(Working set)에 대한 자세한 설명**

가상 메모리, 즉 페이징을 사용하면 순수하게 물리 메모리에 직접 접근하는 것보다 속도가 느릴 수 밖에 없습니다. 왜냐하면 페이지 폴트, 페이지 스왑 등 과정을 거쳐야 하기 때문입니다.

그러므로 그것을 보완하기 위해서 프로세스 워킹 셋 이란 것을 도입합니다. 프로세스가 참조하는 데이터를 물리 메모리 영역에 상주시켜서 페이지 폴트, 페이지 스왑 등의 과정 없이 바로 데이터를 참조할 수 있게 하는 것입니다. 일종의 캐싱이라고 볼 수 있겠습니다.

프로세스는 자신만의 프로세스 워킹 셋을 가지고, 프로세스 워킹 셋에는 해당 프로세스가 자주 참조하는 코드 페이지와 데이터 페이지가 포함되어 있습니다.

프로세스가 프로세스 워킹 셋에 존재하지 않는 페이지를 참조하게 되면 그때 페이지 폴트가 발생하고, 캐싱 미스가 났을 때 처리하는 것 처럼 프로세스 워킹셋에 그 해당 페이지를 추가합니다.

메모리가 부족한 상황에서는 프로세스 워킹 셋의 크기를 줄이기도 합니다. 1. 오랫동안 참조되지 않은 페이지, 2. 자주 참조되지 않은 페이지 를 제거합니다. Temporary and Spatial Locality에 따라 적용되는 것 입니다.

워킹 셋의 크기 자체는 교수님이 수업에서 얘기한 대로, 표준 등으로 정해지지 않았습니다.

1. **스래싱(Thrashing)에 대한 자세한 설명**

스래싱은 컴퓨터가 흔히 말하는 ‘렉’을 먹어서 수행력이 현저하게 떨어지는 상황을 이야기 합니다. 그 원인으로는

1. 메모리 부족으로 인해 하드디드크 – 램 사이의 페이징
2. 페이징이 너무 빈번하게 일어나는 경우

이 있습니다.

만약 메모리가 부족한 상태에서 많은 메모리를 필요로 하는 여러개의 프로세스가 현재 올라와 있다고 합시다. 그런 경우 컨텍스트 스위치를 할 때마다 부족한 메모리를 충당하기 위해서 스왑이 자주 일어나게 됩니다.

보통 이런 경우, 네트워크 쪽에서도 비슷한 일이 많이 발생하는데, 큐잉 이론의 관점에서 보면 시간이 지나갈 수록 문제가 더 심화되는데, 심화되는 증가폭이 지수적으로 증가하기에, 거의 유일한 해결책은 메모리를 많이 차지하는 프로세스를 강제로 죽이거나, 전체 시스템을 리부팅 하는 것입니다.

1. **Swap 파티션에 대한 자세한 설명**

스왑 파티션은 주 메모리에 있는 여역을 보조기억장치에 저장을 할 때 사용되는 파티션입니다.

주로 사용되는 부분은

1. 주 메모리가 가득 찼을 경우 추가 공간을 얻기 위해
2. 최대 절전 기능시 주 메모리 내용을 보조기억장치로 복사

입니다.

90년대에는 정말로 RAM이 많이 비쌌기 때문에, HDD에 추가적인 메모리를 구하는 것이 거의 당연시 되었습니다. 하지만 요즘에는 RAM 디스크 등 오히려 RAM의 가격이 많이 낮아지고 빠른 속도로 연산이 필요한 경우가 늘어서, 아예 스왑을 하지 않기 위해 스왑 파티션을 만들지 않는 경우도 있습니다.

처음 윈도우즈 vista가 나왔을 때, usb 메모리를 꽂으면 “컴퓨터의 성능이 향상될 수 있다, 예, 아니오” 같은 알림창이 떴었습니다. 언듯 들은 얘기로는 그 USB 메모리를 일종의 스왑 파티션으로 이용하겠다는 것 같았습니다.

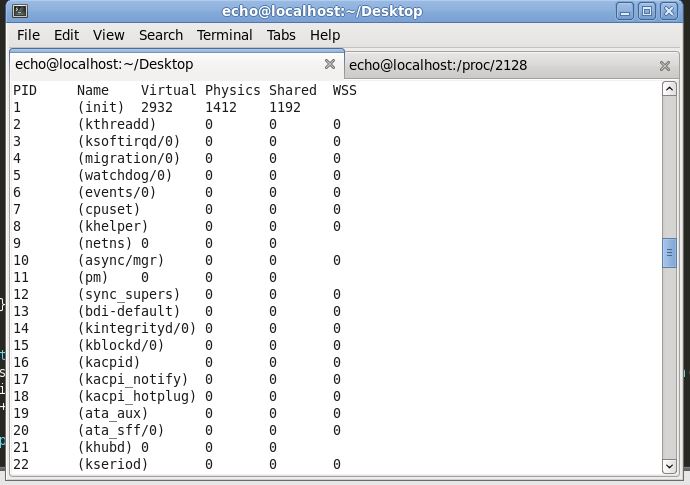
**구현 방법**

1. Opendir를 이용하여 /proc/를 열고
2. Readdir 함수를 이용하여 위에서 오픈한 /proc/ 내부의 entry들을 구합니다.
3. Entry가 PID인지 아닌지 atoi 함수를 이용하여 확인합니다.
4. /proc/PID/stat 파일을 열어서 프로세스의 이름을 구합니다.
   1. Stat의 2번째 항목이 바로 그것임을 알 수 있음.
5. 만약 이 Entry(pid)가 처음 발견된 것이라면, Map(dictionary)에 집어넣습니다.
6. /proc/PID/statm 파일을 열어서 메모리 내용을 구합니다.
   1. 차례대로 1 가상메모리 페이지갯수, 2 물리메모리 페이지갯수, 3 공유메모리 페이지 갯수 임을 알 수 있음.
7. 만약 물리 메모리가 지난번(previous) 물리메모리와 다르다면 체크를 해줍니다.
8. /proc/ 내부에 있는 모든 Entry에 대하여 3~7 과정을 반복합니다.
9. 그 후 Map에 있는 ProcessInfromation들을 화면에 출력합니다.
10. Usleep 함수를 이용하여 0.5초간 sleep을 하고,
11. 1~10과정을 반복합니다.

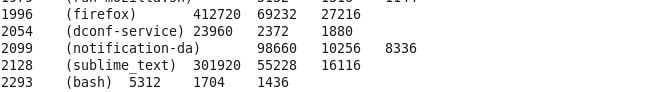
구현된 코드(processInfo.cpp)에 기본적인 주석이 달려있고, 최대한 이해하기 쉽게 작성하였습니다.

참고로 *G++*로 컴파일하여야 하며, 페도라 14를 깔 때 같이 깔리는 G++ 버전으로도 돌아감을 확인했습니다.

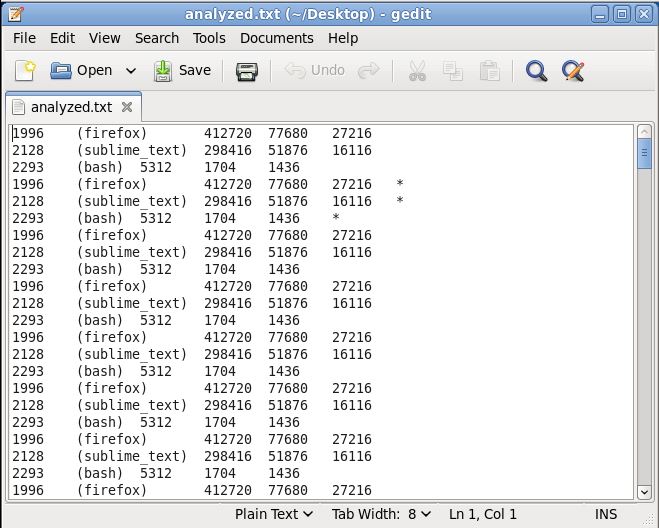
**실행 화면**



1. 첫 부분에는 커널 프로세스들이 존재



1. 뒤로 가면 유저 프로세스가 등장.



1. 결과물을 .txt로 저장하여 분석

**결과 분석**

1. **수행 내용**

저는 파이어 폭스를 켜서 신문 기사를 읽고, Sublime Text에서 새로운 창들을 마구잡이로 열고, 페도라에 기본적으로 깔려있는 스도쿠 게임을 하였습니다.

1. **그래프**

**분석 1. Sublime**

서브라임의 경우 중간에 갑자기 메모리가 2배로 뛰어 오름을 볼 수 있습니다. 하지만 실제 Physical 메모리는 서서히 증가하였습니다.

저 당시 저는 새로운 탭을 계속 생성하였습니다. 새로운 탭을 위한 메모리가 할당되어야 하는데, 매 탭마다 계속 할당하는 것이 아니라, 필요할 때 현재의 2배 크기로 한번에 크게 할당한 것을 볼 수 있습니다.

그러므로 메모리가 증가된 Area는 Heap이라고 추정할 수 있습니다.

충분히 피지컬 메모리가 작기 때문에, 워킹 셋을 전부로 잡아도 큰 문제가 없을 것으로 보입니다.

**분석 2. Sudoku**

전형적인 게임 프로그램들의 그래프를 보여주고 있습니다.

대부분의 게임 프로그램들은 중간에 동적 메모리 할당을 할 수 없습니다. 물론 기술적으로 할 수는 있지만, 메모리 동적 할당을 하는 순간 렉이 걸리기 때문에, 사용자에게 안좋은 경험을 줄 수 있습니다.

그리고 애초에 스도쿠의 경우 큰 메모리가 필요하지 않고, 게임 중간에 새로운 메모리의 할당이 필요하지 않습니다. 그래서 저런 그래프가 나온 것 같습니다.

맨 처음에 증가되는 부분은 STACK과 HEAP 영역이며, 전체 영역을 전부 사용하기 때문에 모든 데이터가 working set에 올라가져있어도 무방할 것 같습니다.

**분석 3. 파이어폭스**

대망의 파이어폭스입니다. 파이어폭스의 경우 지난번에 종료된 탭들이 다시 재생성되었습니다.

그래서 처음에 급격하게 증가함을 볼 수 있고, 제가 그 탭을 종료하였기 때문에 잠시 감소함을 볼 수 있습니다.

중간 부분에 웹툰을 보기 위해 페이지를 옮겼는데, 그 때 웹툰 이미지 파일들에 대한 메모리의 필요로 HEAP 영역에 메모리가 추가 할당됨을 볼 수 있습니다.

그래프의 후반부분은 스도쿠를 하기 위해 잠시 백그라운드에 내려놓았을 때 입니다. 백그라운드에 있을때에는 아무런 메모리의 변화가 없는 것을 확인할 수 있습니다.

파이어폭스의 경우 정말 필요한 핵심 코어부분과, 그렇지 않은 부수적인 부분으로 나눌 수 있습니다. 정확한 수치는 예측하기 힘들지만, 아마 맨 앞부분에서 탭을 종료하였을 때 남은 부분 정도를 워킹셋으로 하면 성능 향상이 될 것이라 추측합니다.