운영체제 Lab 4 보고서

# Goal 1 – Add a system call int printpt(int pid).

### 분석

Goal 1의 목표는 printpt(pid)를 만들어 프로세스의 page table을 출력하는 것이다.

page table을 출력하기 위해선 우선 page directory의 주소를 알아야 하는데 해당 주소는 proc 구조체에 들어있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

xv6에서는 아래와 같이 Two-Level Page Table을 사용한다

텍스트, 스크린샷, 폰트이(가) 표시된 사진

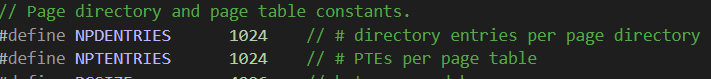
자동 생성된 설명

Page Directory index와 Page Table index는 아래 매크로를 통해 가상 주소에서 얻어낼 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Page Directory와 Page Table의 개수는 각각 2^10 = 1024개이며 해당 상수도 정의되어 있다.



가상 메모리에서 유저 영역은 KERNBASE 바로 밑 까지의 영역이므로 0x0 ~ 0x7fffffff까지이다.



vm.c의 walkpgdir 함수는 주어진 주소 va의 page를 찾아서 page가 할당되어 있으면 page table을 반환하고 할당되어 있지 않으면 해당 page table을 할당한 뒤 반환해주는 함수다.

프로세스의 page table을 찾는 부분이 우리가 만들어야 할 page table 출력과 굉장히 유사한 것을 확인할 수 있어 해당 코드를 참조하기로 했다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

### 수행

printpt 시스템 콜을 만들었다.

syscall.h

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

syscall.c





usys.S



defs.h

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

user.h

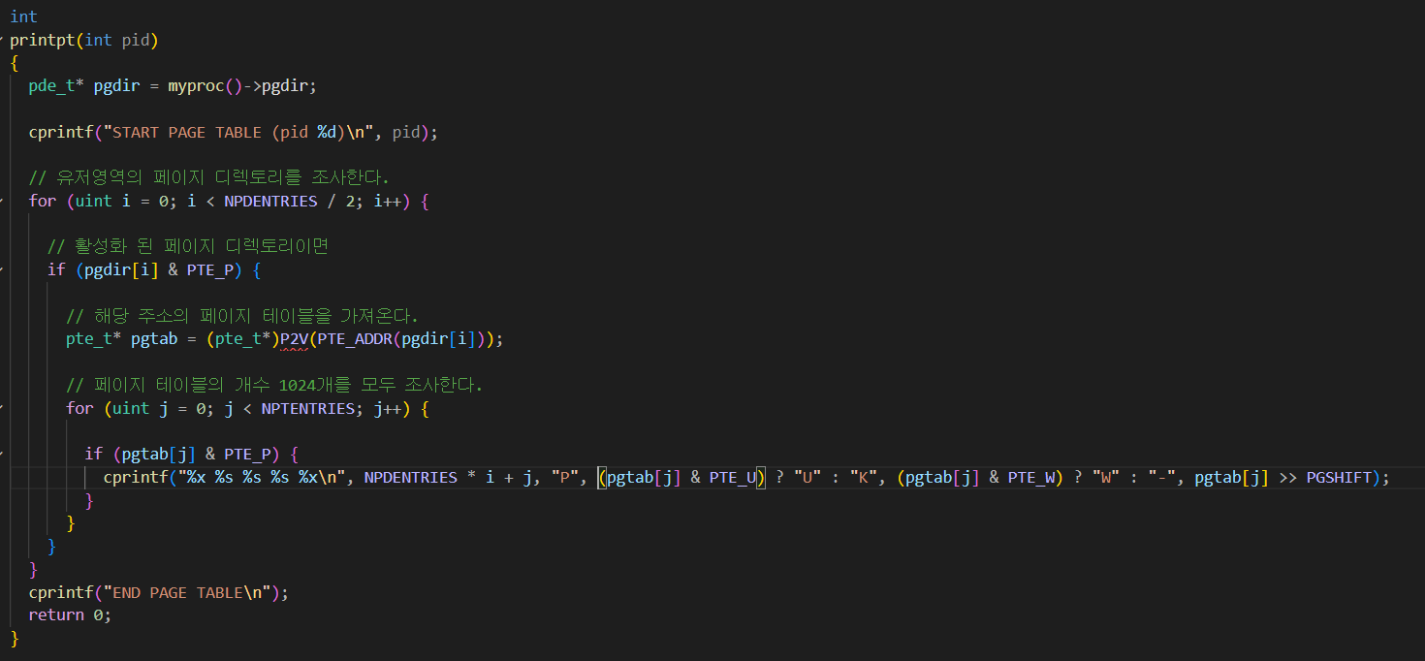


sysproc.c

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

proc.c



pgdir 변수에 현재 프로세스의 pgdir의 주소를 넣어주었다.

이후 반복문을 가상 메모리의 유저 영역의 Page Table 개수인 512만큼 돌아주면서 해당 페이지 디렉토리가 활성화 되어있는지 확인한다.

활성화된 Page Directory의 상위 20비트가 Page Table의 주소이므로 PTE\_ADDR을 통해 Page Table의 첫 주소를 찾고 해당 주소는 가상 주소이므로 실제 주소로 바꾸기 위해 P2V 매크로를 사용했다.

한 Page Table안에 Page Table Entry가 1024개 있으므로 1024번 반복문을 돌면서 활성화된 Page Table Entry를 찾아주었다.

이후 활성화된 PTE의 VPN, present bit, user-mode bit, write bit, PFN을 출력해주었다.

printpt의 작동 확인을 하기 위해 recurse 명령어를 추가하기 위한 작업을 했다.

Makefile

폰트, 그래픽, 스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

recurse.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

recurse.c에서 printpt 주석을 지우고 xv6를 실행했다.

recurse 100을 했을 때 활성화 되어있는 PTE을 출력하는 것을 확인할 수 있다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

# Goal 2 – Rearrange stack.

### 분석 - 1

exec.c에서 아래 부분은 stack과 guard를 할당하는 부분이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

sz는 pcb에서 확인할 수 있는데 프로세스 메모리의 크기이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

해당 내용을 바탕으로 stack과 guard를 어떻게 할당하는지 살펴보면

우선 다음 페이지의 주소를 찾고

sz = PGROUNDUP(sz);

해당 페이지부터 2개의 페이지를 할당한다.

  if((sz = allocuvm(pgdir, sz, sz + 2\*PGSIZE)) == 0)

    goto bad;

allocuvm에서는 mem에 kalloc을 하여 실제 메모리 frame 하나를 할당 받아   
memset으로 초기화 하고 mappages를 통해 PTE에 매핑을 한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

다시 돌아가 2개의 페이지를 할당 받으면 clearpteu를 통해 user-mode의 비트 PTE\_U를 0으로 지워 kernel-mode로 바꾸어 guard를 만들어 주고

clearpteu(pgdir, (char\*)(sz - 2\*PGSIZE));

stack pointer를 sz로 만들어준다.

sp = sz;

sz의 값은 allocuvm에서 반환된 값이 newsz, 즉 기존의 sz + 2 \* PGSIZE이므로 스택 다음 페이지의 시작주소를 가지고 있기 때문에 해당 위치부터 stack pointer가 시작하면 처음 값을 넣을 때 stack의 최상단에서 시작하는 것을 알 수 있다.

exec.c로 돌아와서 이후 명령어 실행할 때 사용하는 인자들을 스택에 저장해준 뒤   
다음과 같이 pcb에 pgdir과 sz를 trap frame에 eip와 esp의 값을 넣어주는 것을 확인할 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

### 수행 - 1

가장 먼저 한 일은 stack의 위치를 맨 위로 올려주었다.

exec.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

stack을 맨 위로 올린다는 것은 KERNBASE 바로 밑에 stack을 만들어준다는 의미이므로 KERNBASE – PGSIZE를 하여 할당할 page의 시작주소를 찾을 수 있다.

sz의 경우에는 이후 heap을 할당할 때 사용하기 때문에 기존 code나 data 영역 위에서 시작 할 수 있도록 tmp에 다음 페이지의 시작위치를 담아주었다.

allocuvm을 통해 page를 할당한 뒤 sp에 KERNBASE를 넣어 stack 맨 꼭대기부터 시작할 수 있게 했다.

이렇게 넣을 경우 allocuvm에서 문제가 생기는데 stack을 올리면 newsz가 KERNBASE의 값이 나오기 때문에 if (newsx >= KERNBASE)에서 문제가 발생한다.  
그러므로 아래와 같이 등호를 지웠다.

vm.c

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

또한 마지막 pcb와 tf을 갱신하는 부분도 수정하였다.

curproc->sz에 미리 담아두었던 tmp의 값을 넣어주었다.

exec.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

### 분석 - 2

이렇게 실행할 경우 제대로 시작하지 않았다.

다음과 같은 trap이 발생하면서 실행이 되지 않는데 init의 0x357부분에서 trap이 발생하는 것을 확인할 수 있어 init.asm에서 0x357 부분을 살펴보았다.



0x357은 fork함수와 호출 부분이기에 fork 함수를 찾아보았다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

fork함수에서 문제가 발생할 수 있는 부분은 아래와 같았다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

copyuvm은 자식의 pgdir을 d로 새로 만들고 부모 프로세스의 pte를 d에 복사한 뒤 d를 반환하는 함수이다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

해당 함수에서 반복문을 0부터 sz까지 도는데 우리가 스택을 맨 위로 올렸기 때문에 sz까지가 아닌 유저 영역 메모리 전체를 돌 필요가 있었다.

또한 아래부분에서 할당되지 않은 page거나 현재 page가 PRESENT에 있지 않으면 panic을 일으키는데 유저 영역 메모리 전체를 돌면 대부분의 page가 할당되지 않았기 때문에 해당 부분은 건너 뛸 필요가 있었다.

if((pte = walkpgdir(pgdir, (void \*) i, 0)) == 0)

      panic("copyuvm: pte should exist");

if(!(\*pte & PTE\_P))

      panic("copyuvm: page not present");

### 수행 – 2

vm.c - copyuvm

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

PTE의 복사를 유저 영역 전체로 바꿔주고, 할당되지 않은 부분은 continue를 통해 건너 뛰어 주었다.

### 분석 – 3

이번엔 trap이 발생하지 않지만 어디선가 무한 루프를 돌고 있었다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

이에 대한 오류를 추적하는 도중 프로세스가 생성될 때 allocproc()에서 최초 초기화를 해주는데  
프로세스가 return 되어 돌아갈 주소는 다음과 같이 forkret의 주소로 설정된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그래서 forkret의 맨 아래줄에 디버깅을 위해 해당 출력문을 놓고 다시 프로그램을 실행해보았다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

아래와 같이 계속해서 프로세스가 계속해서 생겨나는 것을 확인할 수 있다.

스크린샷, 텍스트, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

정상적인 경우에는 아래와 같이 1번 init과 2번 sh가 생겨나야 하지만 그러지 않은 것을 보았을 때 프로세스 생성과정에서 문제가 생겼다는 것을 확인할 수 있었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그렇게 오류를 추적하던 중 교수님께서 stack위치를 옮기면 side-effect가 생길 수 있으니 해당 부분을 조심하라고 했던 것이 기억났다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

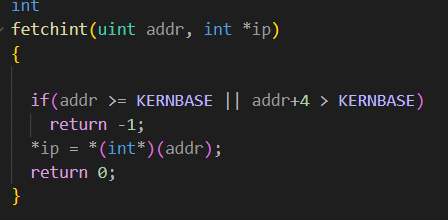
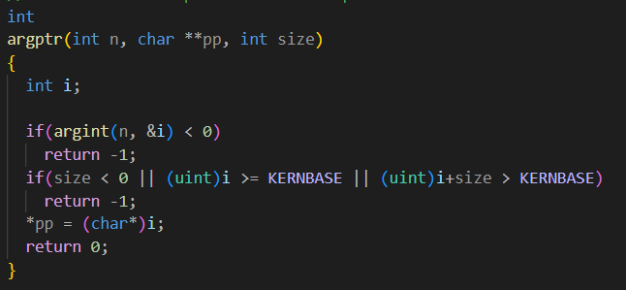
자동 생성된 설명

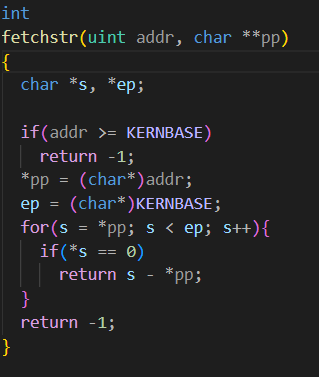
fetchint, fetchstr, argptr을 보면 모두 esp에서 받아오는 주소가 sz를 넘으면 -1을 반환한다.   
하지만 스택을 올렸기 때문에 sz랑 비교하는 것이 아닌 스택의 최 상단 kernbase와 비교하여야 한다.

### 수행 – 3

syscall.c에서 인수를 옮기는 fetchint, fetchstr, argptr에서 현재 process의 sz의 값을 이용하는 부분을 KERNBASE와 비교하게 만들었다.

syscall.c



정상적으로 쉘이 시작하고 recurse 100을 했을 때 가장 밑에 code, 맨 위에 stack 1page 생성되는 것을 확인할 수 있다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

# Goal 3 – Demand paging for stack growth

### 분석

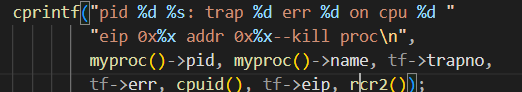
recurse를 일정 값 이상 실행하면 아래와 같이 할당되지 않은 메모리를 참조하여 trap 14가 발생한다.

그러므로 trap 14가 발생했을 때 새로운 page를 할당해주면 된다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

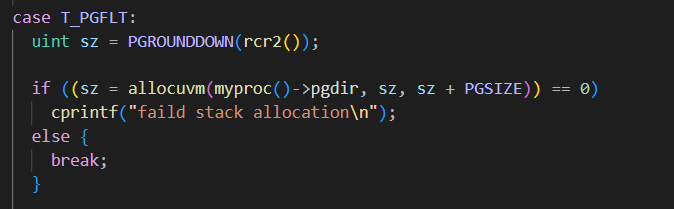
trap이 발생했을 때 cr2 레지스터에 현재 fault가 난 주소가 들어있으니 새로운 스택을 할당할 때 이를 이용하면 된다.



### 수행

trap.c에서 trap 14에 대한 case를 추가해주고 trap 14가 발생하면 새롭게 스택을 추가해주었다.

trap.c



cr2레지스터에 현재 page fault 지점이 있으므로 fault난 지점의 page를 할당해주면 된다.

그러므로 PGROUNDDOWN 매크로를 이용해 fault난 page의 시작 주소를 알아 내고, 해당 부분부터 1개의 page를 할당했다.

만약 스택을 새로 할당하는데 실패하면 break를 넣지 않아 default 처리로 넘어가 프로세스가 중지될 수 있게끔 했다.

recurse 2000을 했을 때에도 trap 14로 프로세스가 중지되지 않고 새롭게 stack이 할당되어 돌아가는 것을 확인할 수 있다.

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명