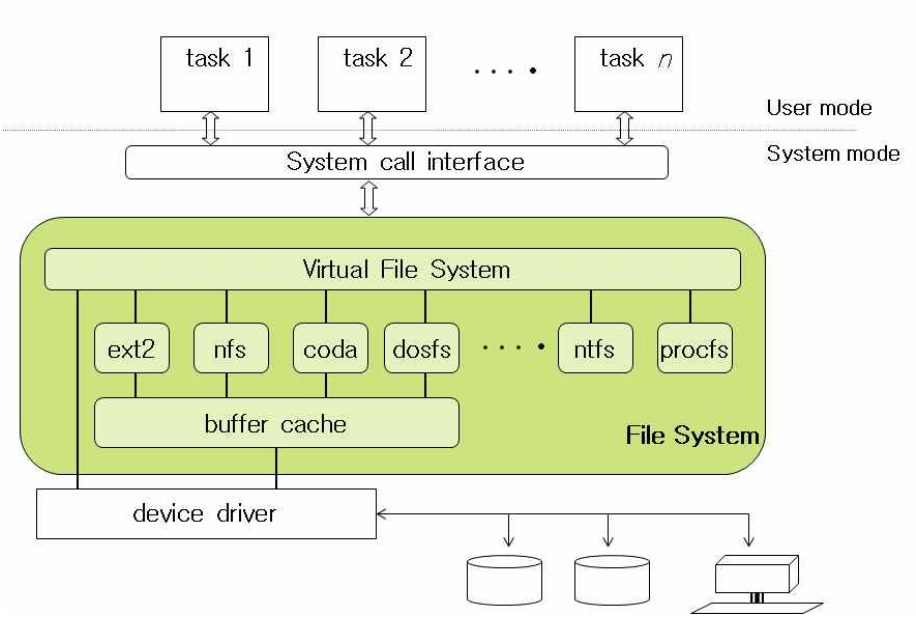
과제 #6 : Virtual File System

○ 과제 목표

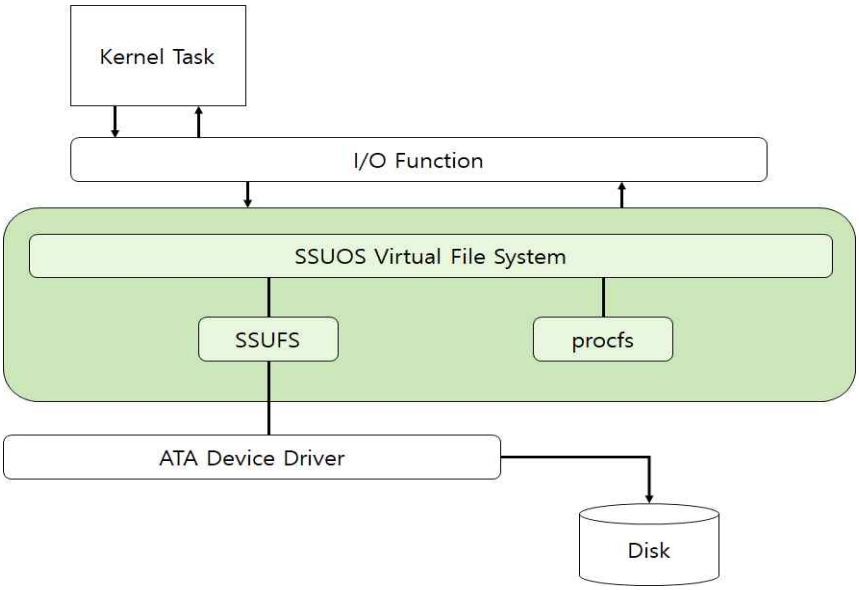
- OS에서 Virtual File System, Proc File System 이해 및 구현
- SSUOS의 VFS에 Proc File System mount 구현

○ 기본 배경 지식

- 일반적인 파일시스템 구조



- SSUOS Virtual File System



✓ SSUOS는 VFS를 통해 SSUFS, procfs를 지원함

✓ SSUFS와 procfs는 각각 Linux의 ext2와, procfs를 간단하게 만든 구조임

- inode 정의

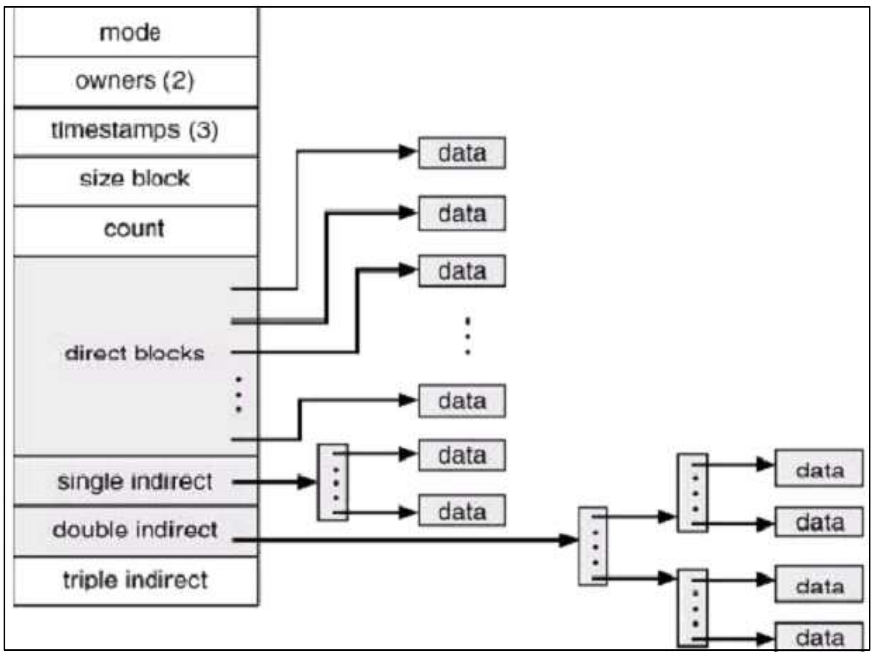
✓ Linux에서 파일을 관리하기 위한 객체로, 파일이 새로 생성되면 만들어짐

✓ 파일의 모든 정보를 관리함

- 파일에 속한 블록 위치 (index block 방법과 유사)
- 파일 소유자 및 접근 권한
- 파일 시간 정보
- 파일 유형 : 커널은 정규 파일 뿐만 아니라 디렉토리, 디바이스, 파이프, 소켓 등도 파일이라는 추상화 객체로 관리

✓ Disk에 정적으로 존재

✓ inode 구조의 예



- vnode 정의(UFS, Unix File System)

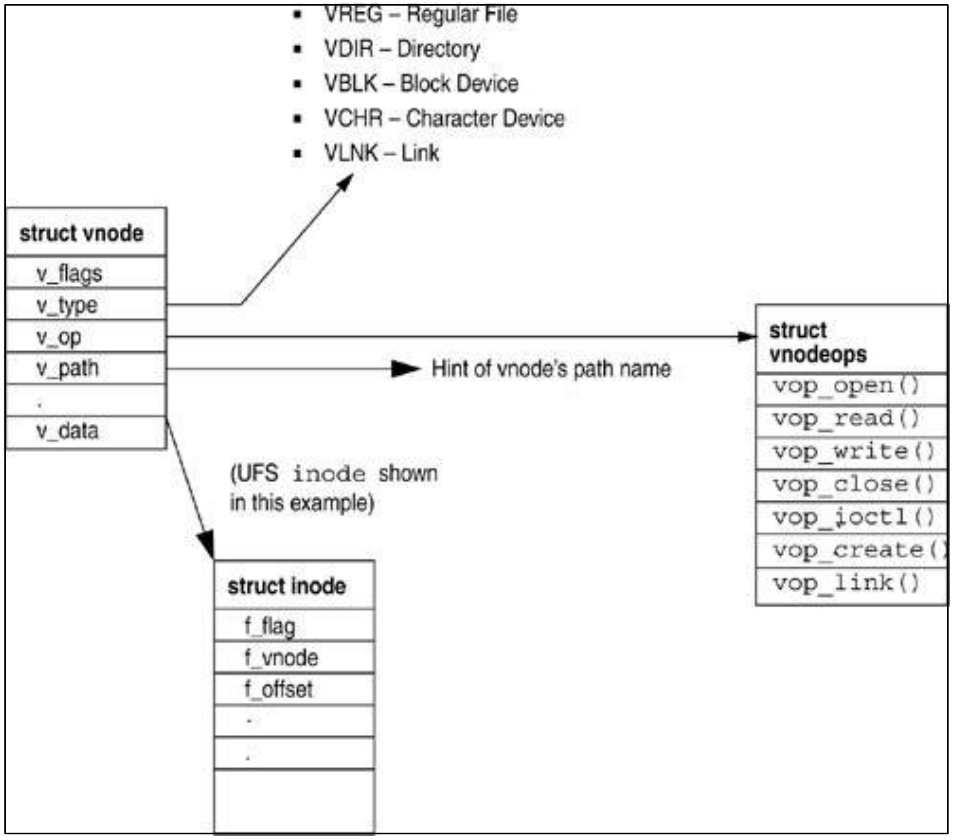
✓ 여러 타입의 File System을 지원하기 위해 사용

✓ 다른 디스크 파티션은 다른 타입의 File System을 가질 수 있으나 mount시 하나의 File System처럼 보여야 함

✓ 파일에 대한 모든 연산을 해당 File System의 적절한 함수로 전달

✓ 메모리에 존재

✓ vnode 구조체의 예



- procfs(Linux)

✓ 프로세스와 다른 시스템 정보를 계층적 파일 구조 형식으로 보여주는 File System

- 프로세스의 통계적 정보
- 하드웨어 정보
- 프로세스 실행 정보

✓ 물리적인 장치를 필요로 하지 않고, 메모리에만 존재함

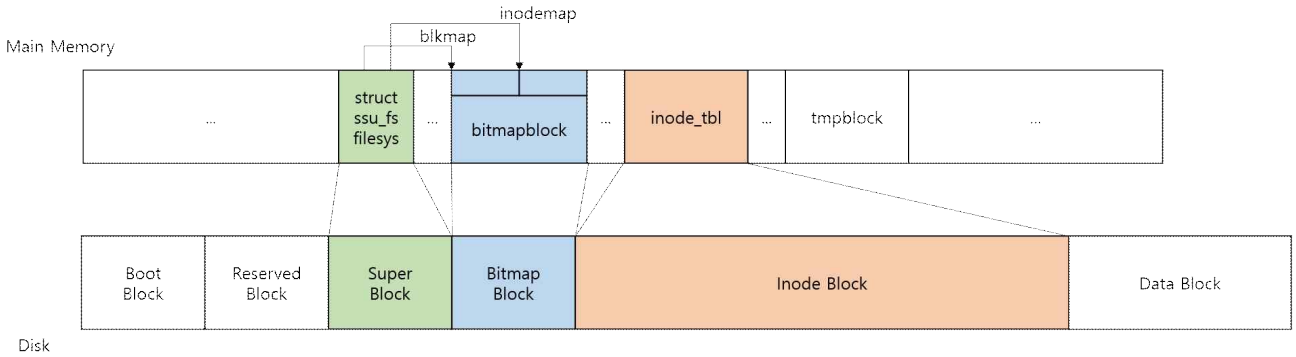
✓ 디렉토리 이동, 입출력을 통해 프로세스의 정보를 확인 가능

✓ procfs 구조

/proc/meminfo	메모리 사용의 세부 사항
/proc/version	커널 버전
/proc/devices	설정되어 있는 장치 목록
/proc/dma	DMA 채널 관련 정보
/proc/modules	현재 사용 중인 커널 모듈 목록
/proc/interrupts	현재 사용 중인 인터럽트
/proc/filesystems	설정된 파일 시스템 목록
숫자 디렉토리	Process PID로 하위 디렉토리에 process 관련 상세 정보 포함

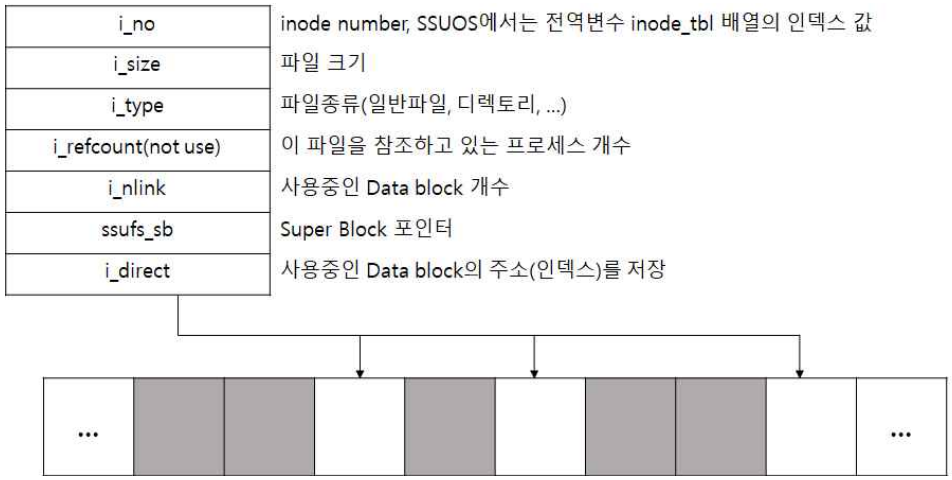
○ 과제 내용

- SSUFS 디스크 블록 구조



- ✓ SSUFS에서는 디스크를 4096B(4KB) 단위로 나누어서 관리
- ✓ 처음 0,1번 블록은 사용하지 않고 2번 블록에 디바이스와 파일시스템에 관한 정보를 담은 super_block구조체가 저장됨
- ✓ 3번블록은 반으로 나눠 앞 2KB는 Block Bitmap, 뒤 2KB는 Inode Bitmap으로 사용
- ✓ 4~7번 블록은 Inode Block, 64B인 inode를 256개까지 저장할 수 있음
- ✓ 8번 블록부터는 Data Block으로, Inode의 directblock으로 할당되어 파일의 내용을 저장하는데 사용됨
- ✓ 현재 프로세스당 사용할 수 있는 메모리가 제한되어 있기 때문에(4KB) 함수내에서 디스크 블록 만큼의 버퍼를 사용하기엔 무리가 있음 -> 전역변수로 선언되어 있는 tmpblock을 사용

- Inode 구조(SSUFS)



- ✓ SSUOS는 간단한 Inode 구조를 제공함(64B)
- ✓ 파일을 생성할때마다 inode가 새로 생성되며, inode bitmap을 통해 사용가능한 inode number를 찾아 할당받음
- ✓ 다른 파일종류에 대해 고려하지 않고 디렉토리만을 사용함
- ✓ ssufs_sb는 SSUFS의 Super Block을 가리키며, Super Block에 Bitmap Block 정보가 있음
- ✓ 일반 파일의 경우 Datablock을 사용하여 파일 내용을 저장하고, 디렉토리의 경우 dirent를 사용하여 디렉토리 정보를 저장

- vnode 구조(SSUFS) - 구현할 내용

v_no	vnode number, SSUOS 에서는 전역변수vnode_table의 인덱스
type	파일 종류(일반, 디렉토리, …)
v_parent	부모 vnode
v_name	vnode 이름
v_op	vnode에서 사용 가능한 함수
childlist	자식 vnode의 리스트
elem	childlist에서 사용하는 list element
info	파일시스템마다 필요한 정보를 저장

- v_op를 통해 하부 특정 File System에 맞는 적절한 함수(open, read, write, close, lseek 등 - 제공한 소스코드에는 구현되어 있지 않음)가 호출됨
- info의 경우 SSUFS의 경우 ssufs_inode를, procfs의 경우 각 파일에서 필요한 정보를 저장

○ 과제 수행 방법

- (1) make_vnode_tree() 구현
  - ✓ SSUOS에서는 프로세스의 디렉토리 변경은 vnode 구조체의 포인터를 변경하는 것으로 구현되어 있음
  - ✓ inode_read()를 이용하여 다음 사항 구현
    - SSUFS의 root inode부터 자식 inode를 읽어 vnode tree를 생성
    - root vnode의 parent는 자신이고, 나머지 vnode의 parent는 부모 디렉토리의 vnode임
    - childlist는 해당 디렉토리에 있는 파일의 vnode를 저장
    - v_name은 파일 이름을 저장

- (2) ssufs_mkdir() 구현
  - ✓ SSUOS에서는 현제 디렉토리의 vnode가 가리키는 하부 파일시스템이 SSUFS일 경우 vnode->v_op.mkdir()을 호출하면 SSUFS의 ssufs_mkdir()을 호출하게 구현되어 있음
  - ✓ inode_read(), inode_write()를 이용하여 다음 사항 구현
    - SSUFS에 새로운 디렉토리 생성
    - 생성된 디렉토리는 디스크에 저장되고, vnode tree에도 추가

- (3) procfs 구현
  - ✓ SSUOS에서는 mount 명령어를 통해 procfs를 mount 할 경우 지정된 vnode의 정보가 procfs의 vnode 정보로 변경 됨
  - ✓ SSUOS의 procfs는 mount시 vnode tree를 생성하지 않고, cd 명령어가 실행되었을 경우 해당 디렉토리의 vnode만 생성(mount 이후 프로세스의 변화를 반영하기 위해)함
  - ✓ ls 명령어의 경우 해당 vnode의 childlist를 통해 파일을 보여주는 것이 아니라, vnode마다 다른 함수를 맵핑하여 다음 사항 구현
    - procfs의 mount root에서 ls를 할 경우 proc_process_ls() 실행
    - proc/[pid]에서 ls를 할 경우 proc_process_info_ls() 실행
    - proc/[pid]/cwd, proc/[pid]/root에서 ls를 할 경우 proc_link_fs() 실행
  - ✓ SSUFS의 VFS는 open, read, write를 지원하지 않음. 따라서 procfs의 파일 내용 출력을 위한 cat 명령어를 지원함

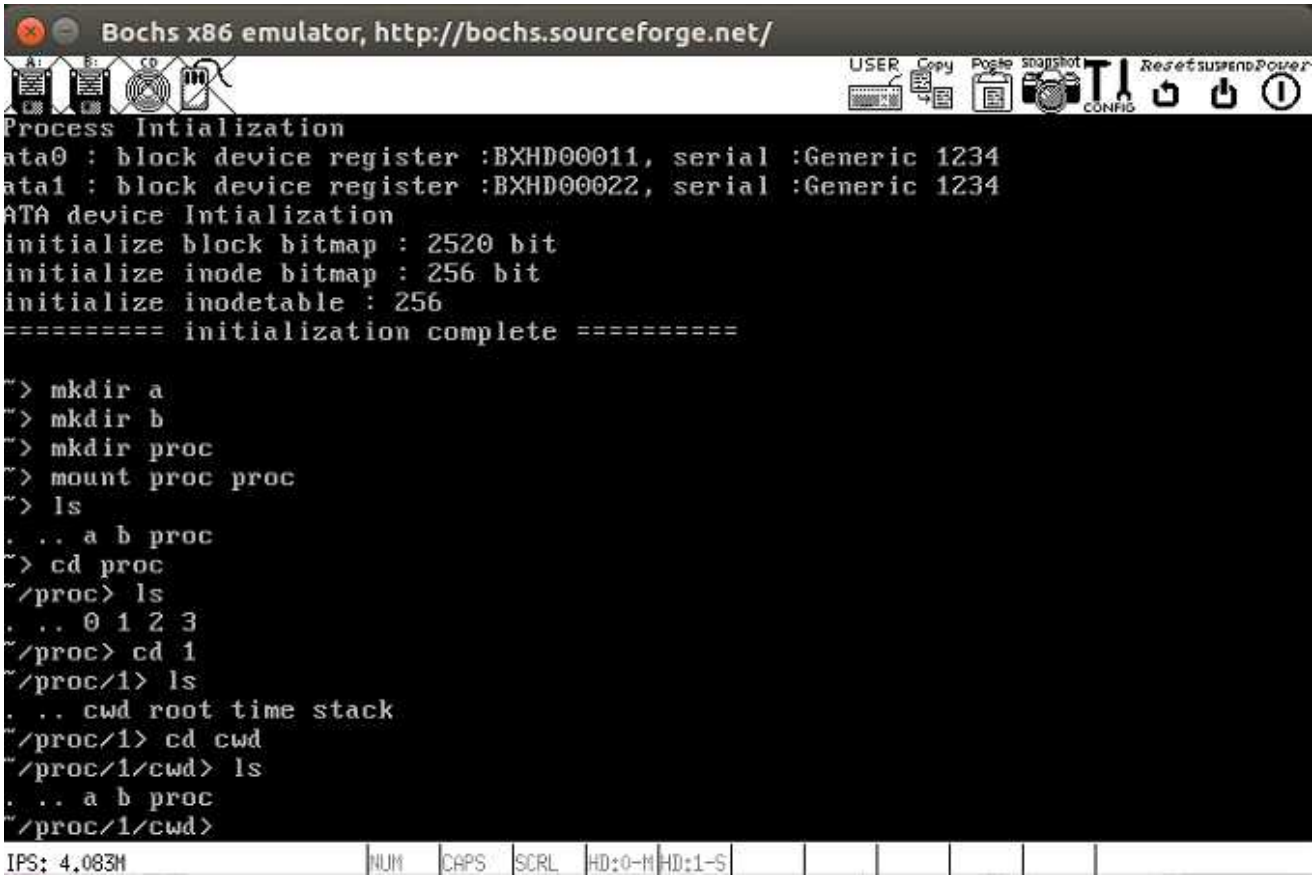
- cat time 입력 시 struct process의 time_used 출력
- cat statck 입력 시 struct process의 stack 출력

- 주의사항

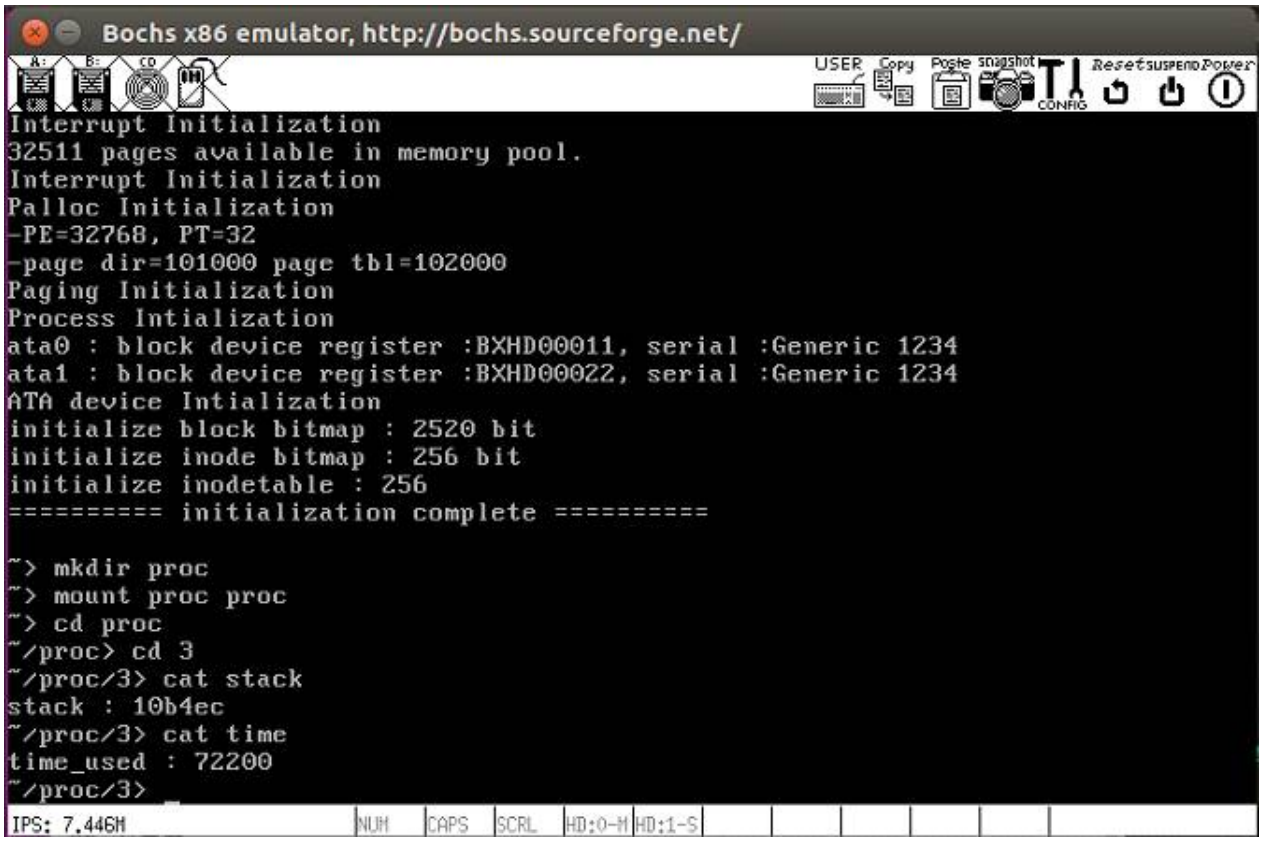
✓ 본 과제에서 프로세스의 가상메모리 크기가 한정되어 있음 -> 함수 내에 tmp[4096] 같이 큰 배열을 잡을 경우 스택오버플로우가 발생할 수 있음
- 단, 허용 가능한 배열의 크기는 프로세스의 코드 및 데이터 크기에 종속적임
- SSUOS에서는 tmpblock[4096]을 전역변수로 선언함

✓ make를 수행하면 disk파일도 초기화 되기 때문에 생성된 디렉토리의 디스크 저장 여부를 확인할 필요 있음
- make run 뒤 다시 make run을 수행한 뒤 생성된 디렉토리의 저장 여부 확인

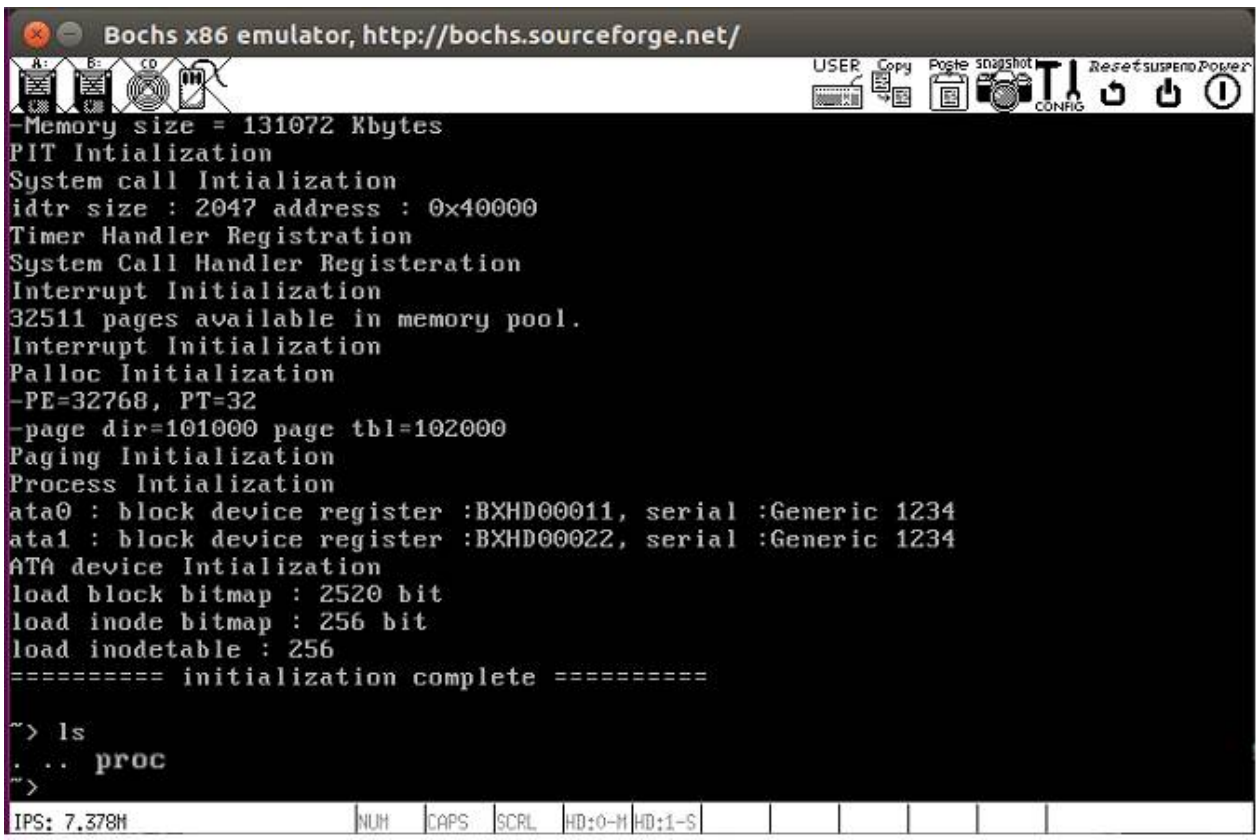
○ 과제 수행 결과



<그림 1> 디렉토리 생성, procfs 마운트 및 하위 디렉토리 탐색



<그림 2> 프로세스의 스택 및 실행 틱 출력



<그림 3> proc 디렉토리 생성 후 재부팅 화면

○ 과제제출 마감

- 2018년 11월 11일 (일) 23시 59분까지 제출

○ 배점 기준

- 보고서 15%
  - ✓ 개요 2%
  - ✓ 상세 설계 명세(기능 명세 포함) 10%
  - ✓ 실행 결과 3%
- 소스코드 85%
  - ✓ 컴파일 여부 5%(설계 요구에 따르지 않고 설계된 경우 0점 부여)
  - ✓ 실행 여부 80%(make_vnode_tree() 구현 15% + ssufs_mkdir() 구현 20% + procfs 구현 45%)

○ 최소 구현사항

- make_vnode_tree() 구현, ssufs_mkdir() 구현, procfs() 구현