

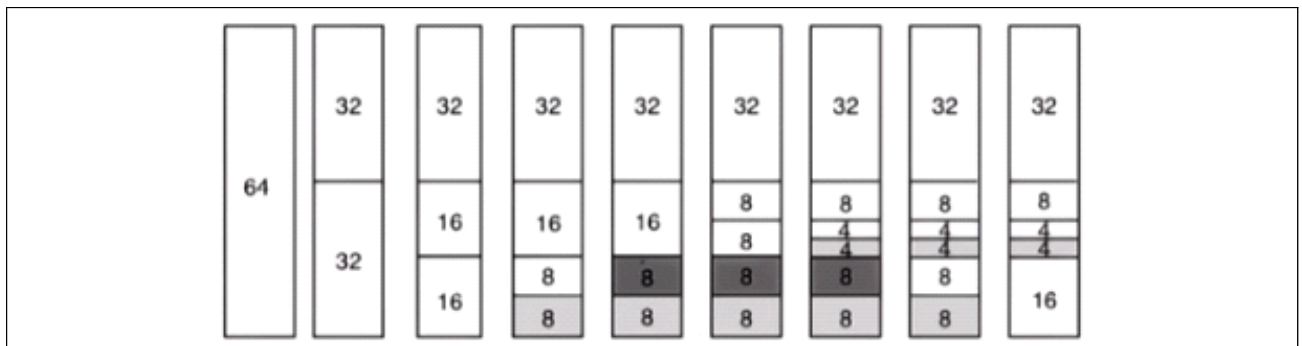
## 과제 #6 : System Call

### ○ 과제 목표

- 시스템 콜 호출 과정 이해
- Buddy 메모리 할당 기법을 이용한 메모리 할당 방법 이해
- SSU OS에서 시스템 콜 malloc(), free() 구현

### ○ 기본 배경 지식

- 커널 영역과 사용자 영역
  - ✓ 운영체제에서 사용자 프로그램이 구동될 때, 파일을 읽고 쓰거나 화면에 메시지를 출력하는 등 커널 영역의 기능들이 필요
  - ✓ 운영체제는 사용자 프로그램이 시스템에 치명적인 영향을 미치는 것을 방지하기 위해 커널 영역과 사용자 영역을 구분하여 시스템을 제어
- 시스템 콜
  - ✓ 시스템 콜은 사용자 영역이 커널 영역의 기능들을 사용할 수 있게 해주는 인터페이스
  - ✓ 크게 프로세스 제어, 파일 조작, 장치 관리, 정보 유지, 통신 유형으로 분류
- Buddy 메모리 할당 기법
  - ✓ [https://en.wikipedia.org/wiki/Buddy\\_memory\\_allocation](https://en.wikipedia.org/wiki/Buddy_memory_allocation)



<그림 1> Linux 시스템 상의 Buddy 메모리 할당 기법의 예시

[출전 : <https://www.blaenkdenum.com/notes/linux/>]

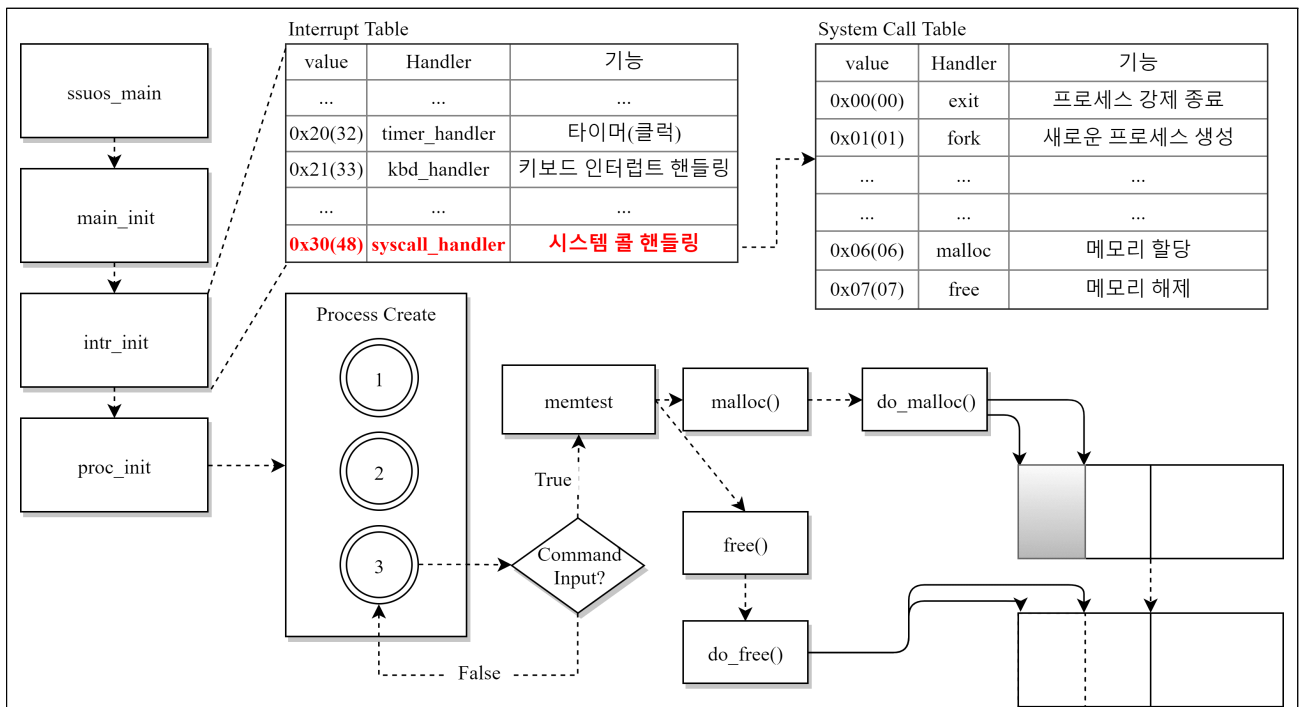
- ✓ Buddy 메모리 할당 기법은 메모리 요청을 만족하도록 적절하게 메모리를 여러 부분으로 나누는 메모리 할당 알고리즘
- ✓ 메모리의 크기를 절반씩 분할을 하면서 가장 잘 맞는 크기의 메모리를 검색하여 반환
- ✓ 본 과제에서는 Kilobyte단위가 아닌 byte단위를 사용

### ○ 과제 요약

- 과제 준비
  - ✓ 운영체제 공지사항 게시판에서 과제 6 소스코드 다운로드

- (1) 시스템 콜 추가
  - ✓ src/kernel/arch/interrupt.c : 시스템 콜 핸들러(syscall\_handler()) 구현
  - ✓ src/kernel/arch/syscall.c : malloc(), free() 구현, 시스템 콜 테이블에 새로 추가된 시스템 콜 추가 및 시스템 콜 매크로 함수 구현
  - ✓ src/kernel/arch/do\_syscall.c : do\_malloc(), do\_free() 시스템 콜 함수 구현
- (2) malloc(), do\_malloc() 구현
  - ✓ Buddy 메모리 할당 기법을 적용하여 메모리 할당
  - ✓ malloc() 함수 및 malloc() 함수의 실제 동작을 수행할 do\_malloc() 함수 구현
  - ✓ src/kernel/include/mem/ 하위에 존재하는 헤더파일을 참조하여 적절한 구조체 및 전역변수를 사용하여 구현
- (3) free(), do\_free() 구현
  - ✓ malloc()을 호출하여 할당한 메모리를 해제
  - ✓ free() 함수 및 free() 함수의 실제 동작을 수행할 do\_free() 함수 구현
  - ✓ src/kernel/include/mem/ 하위에 존재하는 헤더파일을 참조하여 적절한 구조체 및 전역변수를 사용하여 구현

#### ○ 과제 내용



<그림 2> SSU OS의 시스템 콜 호출 과정

- SSU OS의 시스템 콜 처리 과정
  - ✓ `intr_init()`에서 시스템 콜을 처리하기 위한 `syscall_handler()`를 IDT에 등록

- ✓ 각 시스템 콜을 처리하기 위한 시스템 콜 테이블 정의
- ✓ proc\_init()(pid 0)에서 프로세스 0, 1, 2(pid 1, 2, 3)을 생성, 프로세스 1, 2는 수면 상태, 프로세스 3은 쉘 프로세스로써 사용자 입력을 대기
- ✓ 향후 프로세스 0, 1, 2는 데몬 프로세스 역할을 함. 현재 구현에서는 단순 수면 상태로 되어있음
- ✓ 사용자 입력 시 명령어에 따라서 시스템 콜 호출
- ✓ 시스템 콜 호출 시 처리 함수 syscall\_handler() 동작
- ✓ 본 과제에서는 테스트를 위한 자체 구현한 명령어인 memtest를 사용
- ✓ 쉘 프로세스에서 memtest 명령어를 입력하면 malloc()과 free() 함수를 지정된 순서로 호출
- ✓ malloc() 호출 시 do\_malloc() 함수를 호출하여 Buddy 메모리 할당 기법에 알맞게 메모리를 할당
- ✓ free() 호출 시 do\_free() 함수를 호출하여 할당한 메모리를 다시 해제

- 구현해야 할 시스템 콜 함수 프로토타입

malloc0	용도	Buddy 메모리 할당 기법을 이용하여 메모리를 할당해주는 함수
	원형	void* malloc(int size);
free0	용도	malloc() 함수로 할당한 메모리를 해제하는 함수
	원형	void free(void* addr);

#### ○ 과제 수행 방법

- 주어진 상태

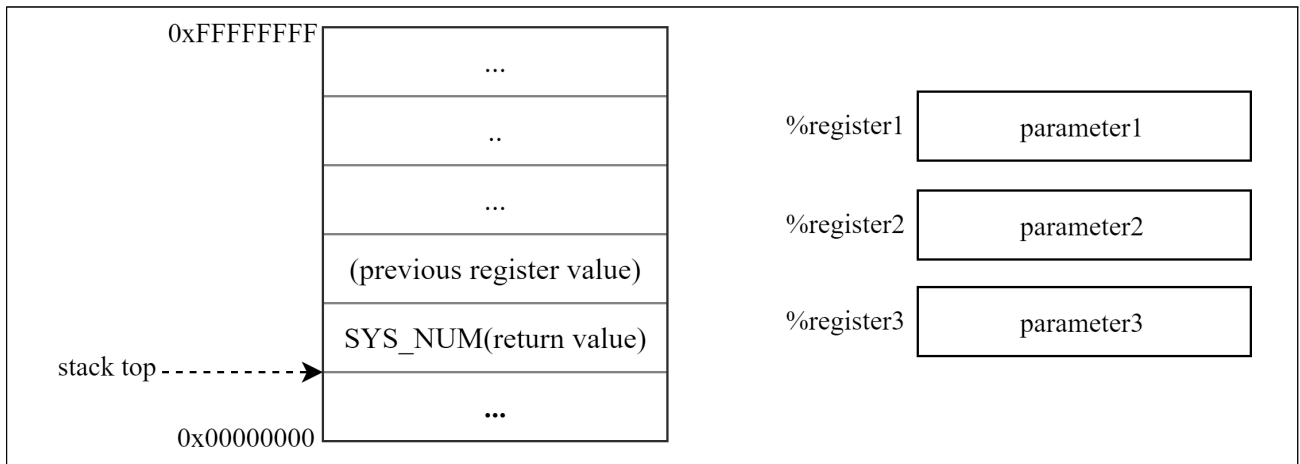
- ✓ 과제 파일을 받아서 그대로 실행한 경우, 키보드 입력을 받는 ssuread() 함수가 수행됨
- ✓ ssuread() 함수는 시스템 콜 함수인데, 시스템 콜 구현이 완전하지 않은 상태이므로 키보드 입력이 되지 않음
- ✓ 시스템 콜 인터럽트 처리를 위한 IDT는 등록되어 있음
- ✓ 현재 시스템 콜 핸들러는 이전 과제와 동일하게 시스템 콜 번호 및 함수 인자를 스택을 통해 받게 구현되어 있음
- ✓ 또한, malloc(), free() 함수는 시스템 콜 테이블에 등록되지 않았음
- ✓ malloc(), free() 함수 및 do\_malloc(), do\_free() 함수 모두 함수 몸체는 구현되어 있지 않음
- ✓ src/kernel/proc/proc.c에 존재하는 malloc\_test() 함수는 수정할 수 없음 (디버깅 시 주석 처리는 가능하나 제출시에는 반드시 초기 상태로 돌려줘야 함)

- 구현 내용

✓ (1) 시스템 콜 추가

- 시스템 콜 호출 시, 시스템 콜 번호는 스택을 통하여 전달하고, 함수 인자는 적절한 레지스터를 통하여 전달하도록 기존의 syscall0 매크로 함수를 참조하여 나머지 매크로 함수를 모두 구현 (sys/kernel/arch/syscall.c)
- 또한, 시스템 콜 핸들러(syscall\_handler())에서도 시스템 콜 번호를 스택을 통해 받고, 함수 인자를 레지스터 값으로 받아오도록 구현(sys/kernel/arch/interrupt.c)
- 리턴 값이 존재하는 시스템 콜을 호출하는 경우, <그림 2> 와 같이 함수의 리턴 값을 시스템 콜 번호를 저장한 위치에 저장되도록 구현
- 위 과정을 구현하기 위해서는 gdb를 통해 디버깅하는 것을 권장함
- 스택에 레지스터 값을 넣는 경우는 기존의 레지스터 값을 넣고 함수 호출이 끝날 때 다시 복원

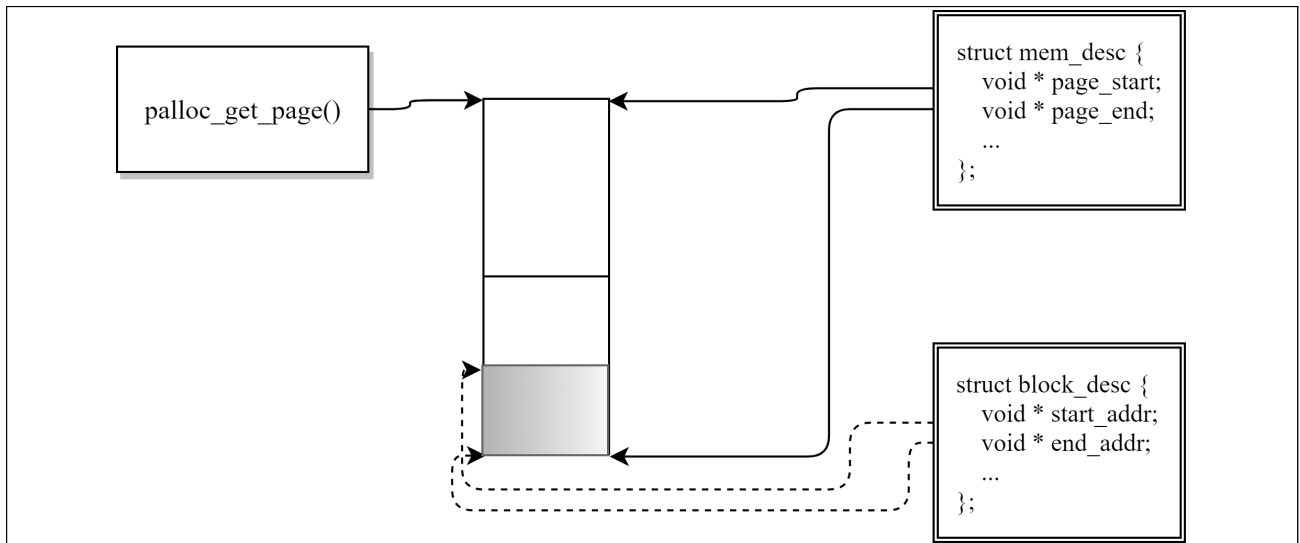
- 하기 위한 경우로 한정. 함수 인자를 전달하는 경우는 허용하지 않음
- `init_syscall()` 함수와 과제 명세에 주어진 '구현해야 할 함수 프로토타입'을 참조하여 시스템 콜 테이블에 `malloc()`, `free()` 시스템 콜을 추가(`sys/kernel/arch/syscall.c`)



<그림 3> SSU OS에서 시스템 콜 호출 시 스택과 레지스터의 상황

✓ (2) `malloc()`, `do_malloc()` 구현

- 현재 SSU OS는 paging 구현이 되어있는 상태(본 과제에서는 과제 #5와는 다른 방향으로 구현되어 있음)
- 프로세스가 동적으로 메모리 할당을 받을 수 있도록 `malloc()` 함수를 구현
- `malloc.h`에 존재하는 메모리 관련 구조체를 참조
- `malloc.c`에 존재하는 함수들을 명세와 주석을 참조하여 구현
- 구현에 사용할 추가적인 함수 정의 가능. 단, 전역 변수는 불가능함
- 커널 시작 시 `init_malloc()` 함수가 실행됨
- `init_malloc()` 함수는 `malloc.h`에 존재하는 전역 변수 초기화를 수행하고 `malloc()` 함수에서 사용할 page를 하나 할당받고 할당 받은 page의 정보를 저장할 `mem_desc` 구조체 초기화를 수행
- `malloc()` 함수를 수행하면 시스템 콜 인터럽트 핸들러에 의하여 `do_malloc()` 함수가 수행됨
- `do_malloc()` 함수에서는 `mem_desc` 구조체를 통해서 지금까지 할당한 block크기 및 위치를 고려하여 요청받은 크기만큼 할당할 수 있는 공간이 있는지를 확인함
- 함수 호출시 요청받은 크기만큼의 공간보다 작지 않은 최소의 2의 n제곱값에 해당하는 공간을 본 과제에서는 block이라고 정의하고, 이에 대한 정보는 `blk_desc` 구조체를 사용하여 관리
- `do_malloc()` 함수에서 유용하게 사용할 수 있는 함수를 `src/kernel/mem/malloc.c`에 프로토타입 및 주석으로 서술해 둬
- block을 할당하는 방법은 Buddy 메모리 할당 기법을 사용하며, 동작 방법은 상위의 '기본 배경 지식' 혹은 아래의 `memtest` 명령어 수행 과정을 확인
- block을 할당해야 하는 상황에서 page 내부에 더 이상 block을 할당할 수 없는 경우, 새로운 page를 할당받아서 `mem_desc`를 초기화
- 새로운 block을 성공적으로 할당한 경우 적절한 문자열 출력과 함께 그 block의 시작주소를 리턴



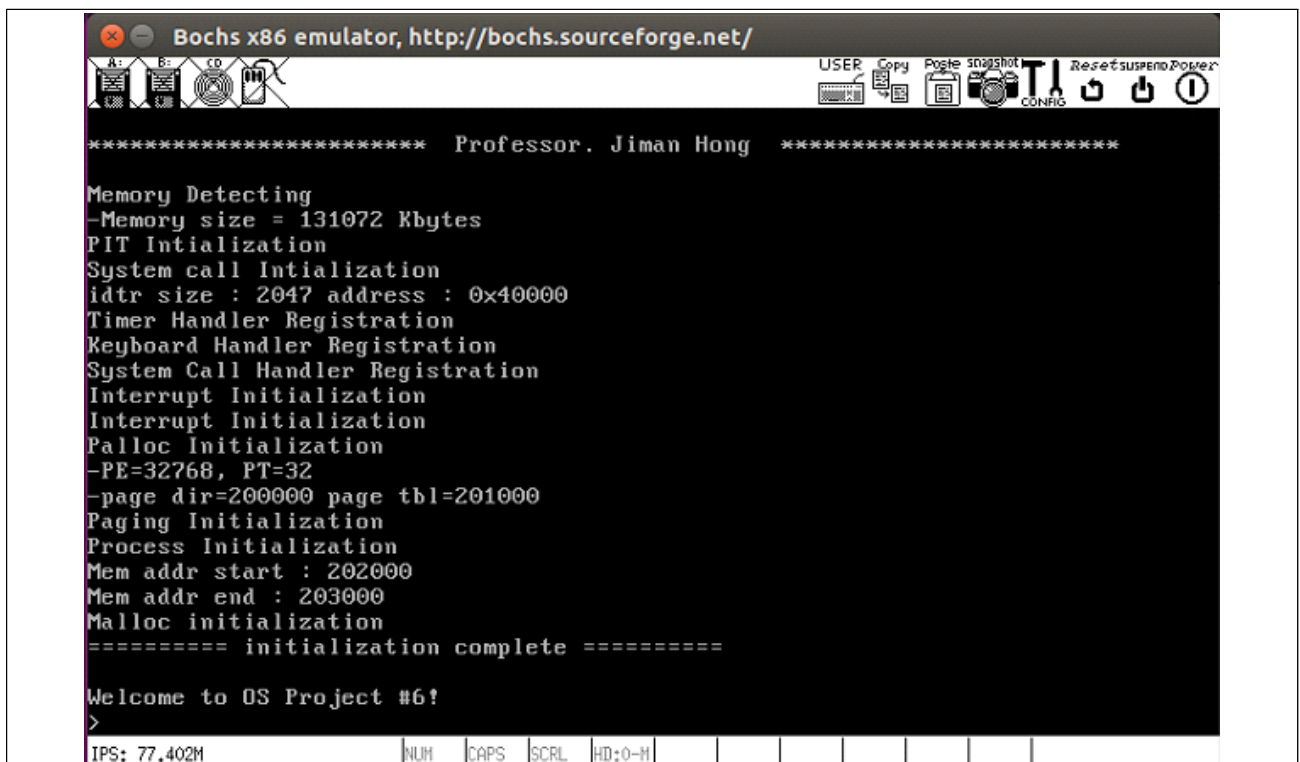
<그림 4> malloc() 함수에 사용되는 메모리 관련 구조체 overview

✓ (3) free(), do\_free() 구현

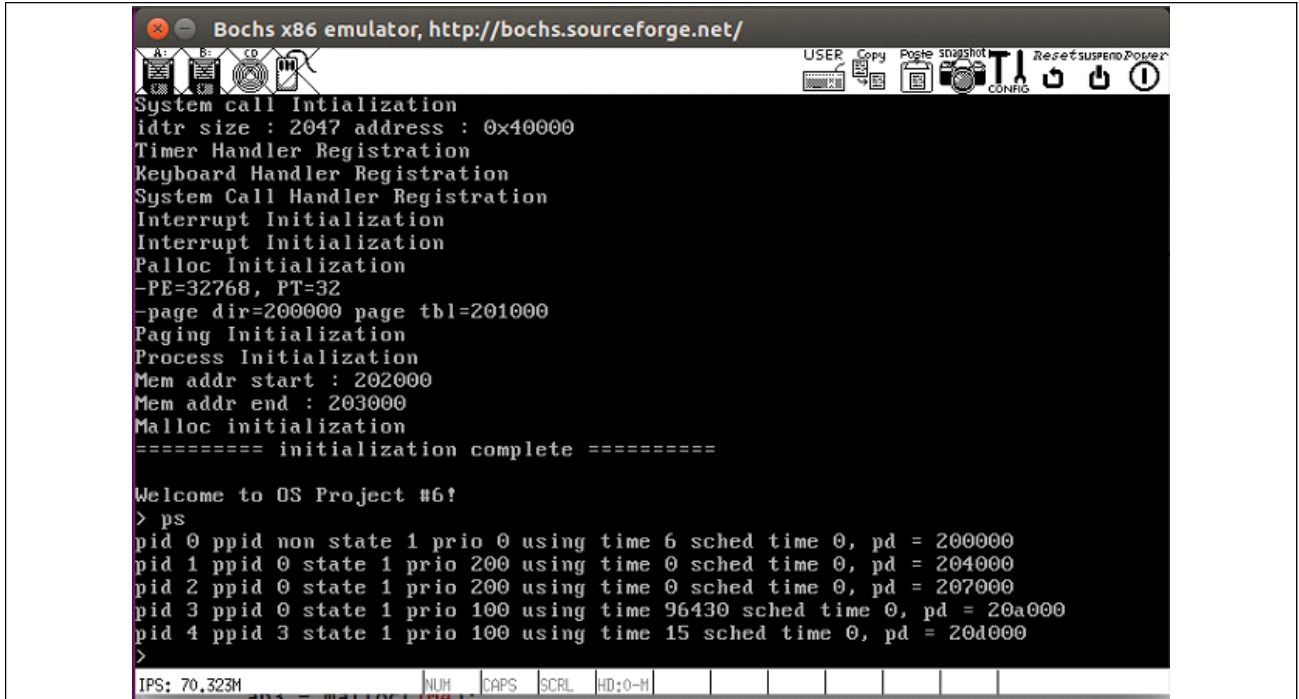
- malloc() 함수로 할당 받은 메모리 주소를 인자로 받으면, 해당 메모리 주소와 동일한 시작 주소를 가진 block이 할당되었는지 여부를 확인
- block이 할당되었던 것을 확인되면 그 메모리의 시작 주소~끝 주소를 해제
- 주어진 문자열의 가변 인자를 적절히 수정하여 알맞은 값을 출력할 것
- malloc.h의 구조체를 참조하여 구현할 것

○ 과제 수행 결과

- 과제를 완전히 수행하고 나서 최초 실행 시



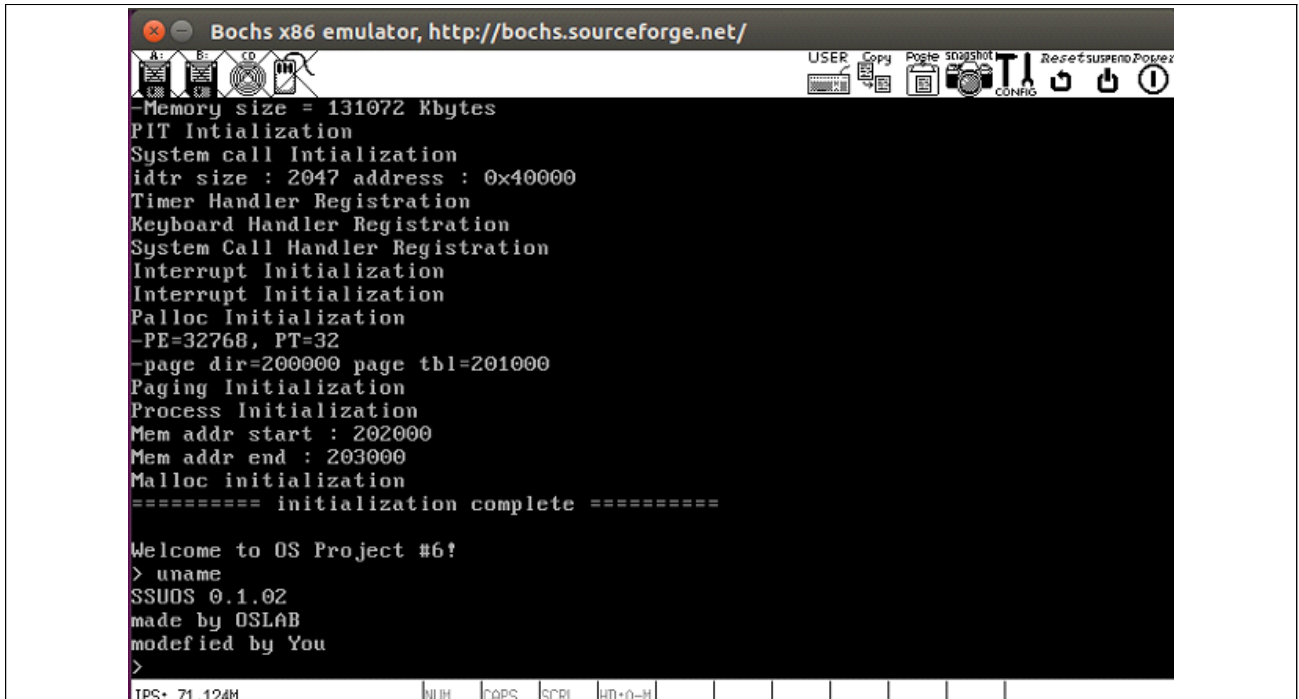
- ps 명령어 입력 시



```
Bochs x86 emulator, http://bochs.sourceforge.net/
System call Initialization
idtr size : 2047 address : 0x40000
Timer Handler Registration
Keyboard Handler Registration
System Call Handler Registration
Interrupt Initialization
Interrupt Initialization
Palloc Initialization
-PE=32768, PT=32
-page dir=200000 page tbl=201000
Paging Initialization
Process Initialization
Mem addr start : 202000
Mem addr end : 203000
Malloc initialization
===== initialization complete =====

Welcome to OS Project #6!
> ps
pid 0 ppid 0 non state 1 prio 0 using time 6 sched time 0, pd = 200000
pid 1 ppid 0 state 1 prio 200 using time 0 sched time 0, pd = 204000
pid 2 ppid 0 state 1 prio 200 using time 0 sched time 0, pd = 207000
pid 3 ppid 0 state 1 prio 100 using time 96430 sched time 0, pd = 20a000
pid 4 ppid 3 state 1 prio 100 using time 15 sched time 0, pd = 20d000
>
IPS: 70.323M
```

- uname 명령어 입력 시



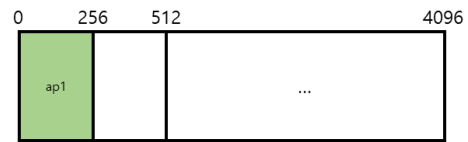
```
Bochs x86 emulator, http://bochs.sourceforge.net/
Memory size = 131072 Kbytes
PIT Initialization
System call Initialization
idtr size : 2047 address : 0x40000
Timer Handler Registration
Keyboard Handler Registration
System Call Handler Registration
Interrupt Initialization
Interrupt Initialization
Palloc Initialization
-PE=32768, PT=32
-page dir=200000 page tbl=201000
Paging Initialization
Process Initialization
Mem addr start : 202000
Mem addr end : 203000
Malloc initialization
===== initialization complete =====

Welcome to OS Project #6!
> uname
SSUOS 0.1.02
made by OSLAB
modified by You
>
IPS: 71.124M
```

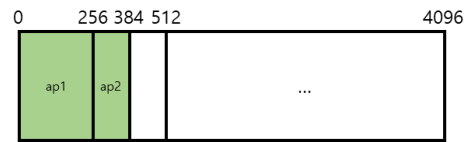
- memtest 명령어 입력 시

- ✓ memtest 명령어는 <그림 5>와 같은 순서로 메모리를 할당하고 해제함
- ✓ memtest는 src/kernel/proc/proc.c의 malloc\_test() 함수를 수행함
- ✓ 아래 <그림 5>는 memtest 명령어를 호출하였을 때 수행되는 malloc(), free() 함수가 한 페이지 내부에서 메모리를 할당하고 해제하는 과정을 나타냄  
(실제 할당받은 페이지에 대한 주소는 고려하지 않았음)

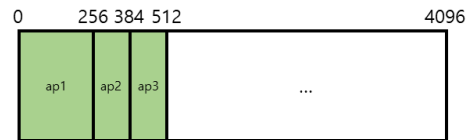
ap1 = malloc(256);



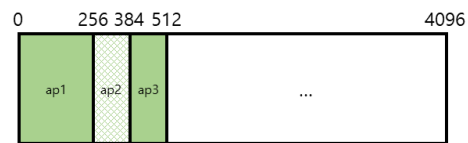
ap2 = malloc(128);



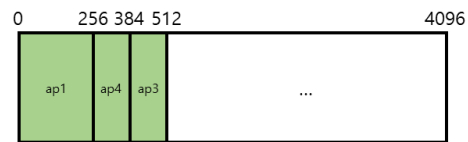
ap3 = malloc(104);



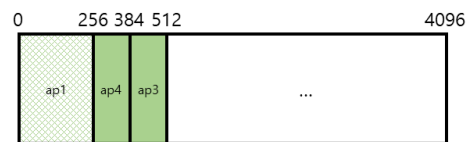
free(ap2);



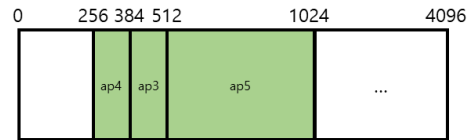
ap4 = malloc(124);



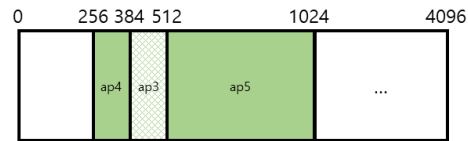
free(ap1);



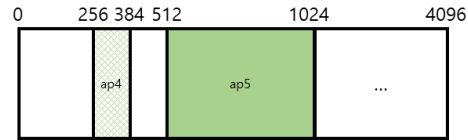
ap5 = malloc(356);



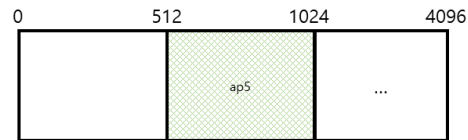
free(ap3);



free(ap4);



free(ap5);



<그림 5> memtest 명령어의 함수 호출 과정

```

Bochs x86 emulator, http://bochs.sourceforge.net/
::First Allocation::
Allocating size... : 256
Memory Allocated Address : 202000 ~ 202100
::Second Allocation::
Allocating size... : 128
Memory Allocated Address : 202100 ~ 202180
::Third Allocation::
Allocating size... : 104
Memory Allocated Address : 202180 ~ 202200
::Fourth Allocation::
Memory Deallocated size : 128(128)
Memory Deallocated Address : 202100
Allocating size... : 124
Memory Allocated Address : 202100 ~ 202180
::Fifth Allocation::
Memory Deallocated size : 256(256)
Memory Deallocated Address : 202000
Allocating size... : 356
Memory Allocated Address : 202200 ~ 202400
Memory Deallocated size : 128(104)
Memory Deallocated Address : 202180
Memory Deallocated size : 128(124)
Memory Deallocated Address : 202100
Memory Deallocated size : 512(356)
Memory Deallocated Address : 202200
IPS: 79.921M

```

<그림 6> malloc(), free() 구현 후 memtest를 실행한 결과

○ 과제 제출

- 2017년 11월 8일 (수) 23시 59분 59초까지 과제 게시판으로 제출

○ 최소 기능 구현

- (1) 시스템 콜 추가
- ps, uname 명령어가 성공적으로 실행되도록 구현
- (2) malloc() 구현

○ 배점 기준

- 보고서 20점
  - ✓ 개요 3점
  - ✓ 상세 설계 명세(기능 명세 포함) 7점
  - ✓ 구현 방법 설명(코드 주석 포함) 8점
  - ✓ 실행 결과 2점
- 소스코드 80점
  - ✓ 컴파일 여부 10점(설계 요구에 따르지 않고 설계된 경우 0점 부여)
  - ✓ 실행 여부 70점(시스템 콜 테이블에 malloc(), free() 함수 추가 5점 + 인자가 있는 시스템콜 (fork(), exit() 등) 호출을 위한 매크로 함수(syscall1, syscall2, ...)를 구현 30점 + Buddy 메모리 할당 기법을 적용하여 do\_malloc() 함수 구현 25점 + do\_free() 함수 구현 10점)