**Operating Systems**

**Spring, 2020**

**School of Software, CAU**

**Project #2**

**- Memory Allocation –**

[Pintos 소스코드 분석 및 구현 보고서]

학번: 20150804

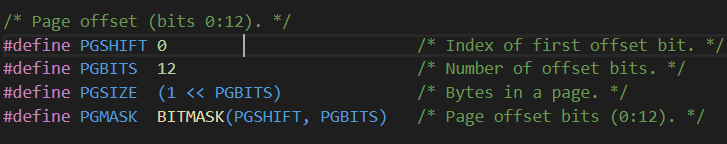
이름: 정 민 준

**1. Pintos 소스코드 분석 보고서**

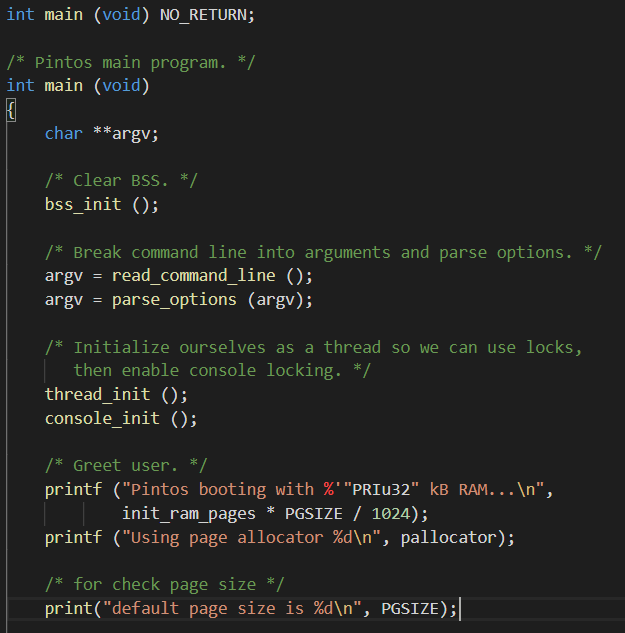
**1.1 메모리 할당 및 관리 시스템 분석**

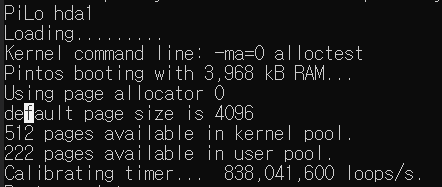
**1.1.1 threads/vaddr.h에 정의된 Pintos 메모리 시스템의 기본 페이지 크기**

- vaddr.h 소스코드를 확인해보겠습니다.



- 다음과 같이 PGSIZE로 페이지 크기가 선언되어 있음을 확인할 수 있습니다. 그러면 이제 main code에 PGISZE를 출력해보겠습니다. (아래 사진 제일 마지막 줄에 코드 추가)

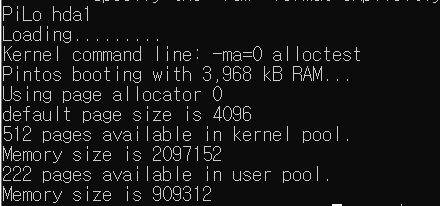


- 실행 결과

- 다음과 같이 4096 byte임을 확인할 수 있습니다.

**1.1.2 커널에서 사용 가능한 메모리 크기**

- palloc\_init 함수를 보면 init\_pool 함수를 통해 kernel pool과 user pool 을 초기화하는 코드를 확인할 수 있습니다. 따라서 pool에서의 메모리 크기를 확인해보기 위해 init\_pool 함수에 page\_cnt \* PGSIZE곱을 출력하는 코드를 추가하였습니다.



- kernel pool, user pool의 메모리를 합쳐 2097152 + 909312 bytes 메모리 크기만큼 커널에서 사용 가능함을 확인할 수 있습니다.

**1.1.3 palloc\_get\_multiple() / palloc\_free\_multiple() 함수 동작 방식**

**- palloc\_get\_multiple :**

파라미터는 flags 와 page\_cnt 를 받습니다. flags 를 통하여 user pool, kernel pool 중에 선택하게 됩니다. 그리고 페이지를 할당하는 동안 critical section을 보호하기 위하여 Lock 을 걸고 작업이 끝나면 Lock 을 해제합니다. Lock 이 걸린 동안은 bitmap\_scan\_and\_flip 함수를 통해 해당 pool에서 사용 가능한 페이지 인덱스를 page\_idx 에 저장하게 됩니다. page\_idx 에 PGSIZE만큼 곱하여 page의 메모리를 설정합니다. 만약 page\_idx 값이 올바르지 않다면 NULL값을 할당합니다. 마지막으로 page를 반환하게 됩니다.

**- palloc\_free\_multiple :**

파라미터는 page 와 page\_cnt 를 받습니다. 먼저 page 가 NULL 이거나 page\_cnt 가 0이라면 함수를 종료합니다. 그리고 해당 페이지가 어느 pool인지를 확인하는 과정을 거치는데 만약 kernel pool, user pool 둘 중 해당되지 않는다면 NOT\_REACHED() 함수를 호출합니다. 이어서 page\_idx 에 pg\_no 함수를 통하여 virtual page number를 저장합니다. 그리고 해당 page\_idx 부터 할당하고자 하는 page\_cnt 만큼 bitmap\_set\_multiple 함수를 통하여 선택된 pool의 map의 값을 false로 설정합니다.

**1.1.3 Fragmentation 발생**

- 먼저 palloc\_get\_multiple() 함수를 사용하여 메모리를 할당하는 경우를 살펴보겠습니다. 해당 함수를 사용할 경우 외부 단편화(External Fragmentation)이 발생합니다. 메모리를 할당하고 free해주는 작업이 반복적으로 진행되는데 이때 실제론 메모리 공간이 여유가 있지만 요청된 메모리 만큼 연속적으로 할당을 해줄 수 없는 상황이 생깁니다.

- 다음으로 페이지 단위로 메모리를 할당하는 경우를 보겠습니다. 페이지 단위로 할당하게 되면 내부 단편화가 발생할 수 있습니다. 페이지 메모리 크기보다 더 작은 공간을 할당해야 하지만 페이지 단위로 공간을 할당하기 때문입니다. 이런 상황에서는 공간 낭비가 발생합니다.

**1.2 Pintos 커널의 malloc() 구현 방식 분석**

**1.2.1 기본 페이지보다 작은 크기의 메모리 영역을 할당하기 위해 사용되는 방법**

- 페이지 단위가 일정하다고 생각해보면 해당 크기보다 작은 메모리 영역을 할당할 수 없습니다. 이를 해결하기 위해서 일정하지 않은 크기로 메모리를 나누면 문제를 해결할 수 있습니다.

- 예를 들어 지금 메모리가 - 10KB, 10KB, 10K, .. 와 같이 일정한 크기로 공간이 나눠져 있다면 이를   
1KB - 2KB - 4KB .. 로 크기 순서대로 공간을 할당하면 기본 페이지보다 작은 크기의 메모리 영역을 할당할 수 있습니다.

**1.2.2 malloc 관련 함수들의 동작 방식**

**- malloc\_init() :**

먼저 descriptor 구조체 desc를 보겠습니다. 블록 사이즈, 공간당 블록 개수, 할당되지 않은 블록 리스트, 락 구조체를 변수로 가지고 있습니다. Malloc\_init 함수에서는 이 descriptor 구조체를 초기화 시켜주는 작업을 진행하는데 먼저 구조체를 선언하고 block\_size, blocks\_per\_arena, free list, lock 순차적으로 초기화를 해줍니다.

**- malloc() :**

malloc 함수에서는 size 변수를 받습니다. 그리고 해당 사이즈만큼 블록을 반환하거나 메모리가 충분치 않다면 null pointer를 반환합니다.

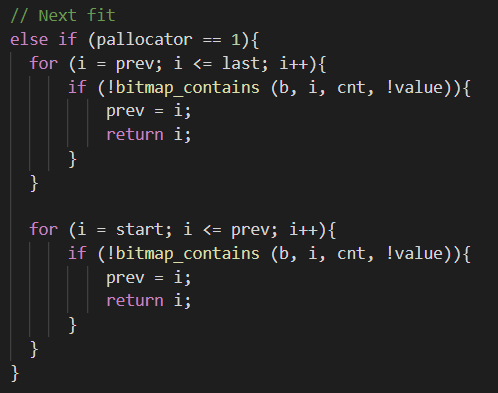
먼저 desc, block, arena 구조체 변수를 선언합니다. 그리고 만약 size가 0이라면 NULL을 반환합니다. for문을 통해서 해당 size를 할당할 수 있는 크기를 가진 desc를 찾습니다. 그리고 palloc\_get\_multiple 함수를 통해 size만큼 연속된 메모리를 할당하게 됩니다. 이때 만약 선택된 desc에 free\_list가 없다면 Lock을 건 상태에서 page를 새로 할당한 후 Lock을 풀어줍니다.

**- free() :**

Free 함수는 block 포인터 변수를 받고 malloc, calloc 등의 함수로 할당된 메모리를 해제하는 함수 입니다. Lock 을 설정하고 list\_push\_front 로 free 하려는 block을 free list로 push 합니다. 만약 페이지가 쓰이지 않고 있다면 for 문을 통해 해당 block을 palloc\_free\_page 함수를 통해 free 하고 Lock 을 해제합니다.

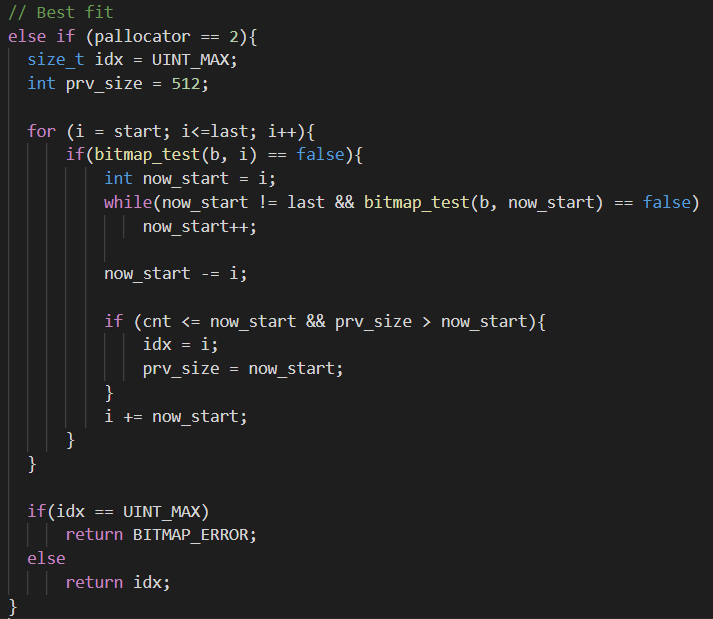
**2. Pintos 메모리 할당 기법 구현 및 테스트**

**2.1.1 Next Fit**



- 이전 할당 위치를 저장하기 위해 prev 전역 변수를 선언하였습니다. 메모리를 이전에 할당했던 인덱스 prev 부터 반복문을 통해 cnt 만큼 할당합니다. 만약 할당하지 못했다면 다시 처음부터 prev인덱스 까지 반복문을 통해 이전 과정을 반복합니다.

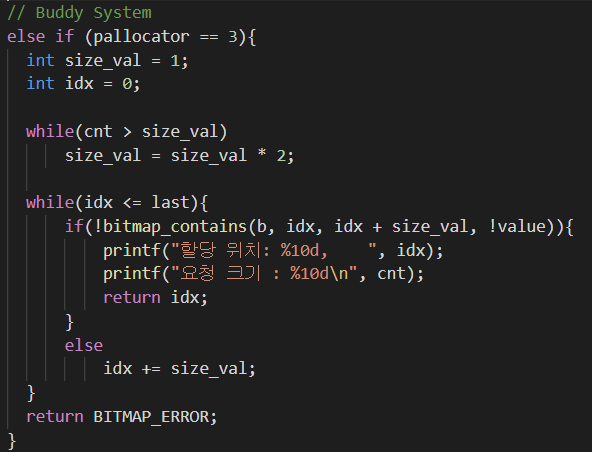
**2.1.2 Best Fit**



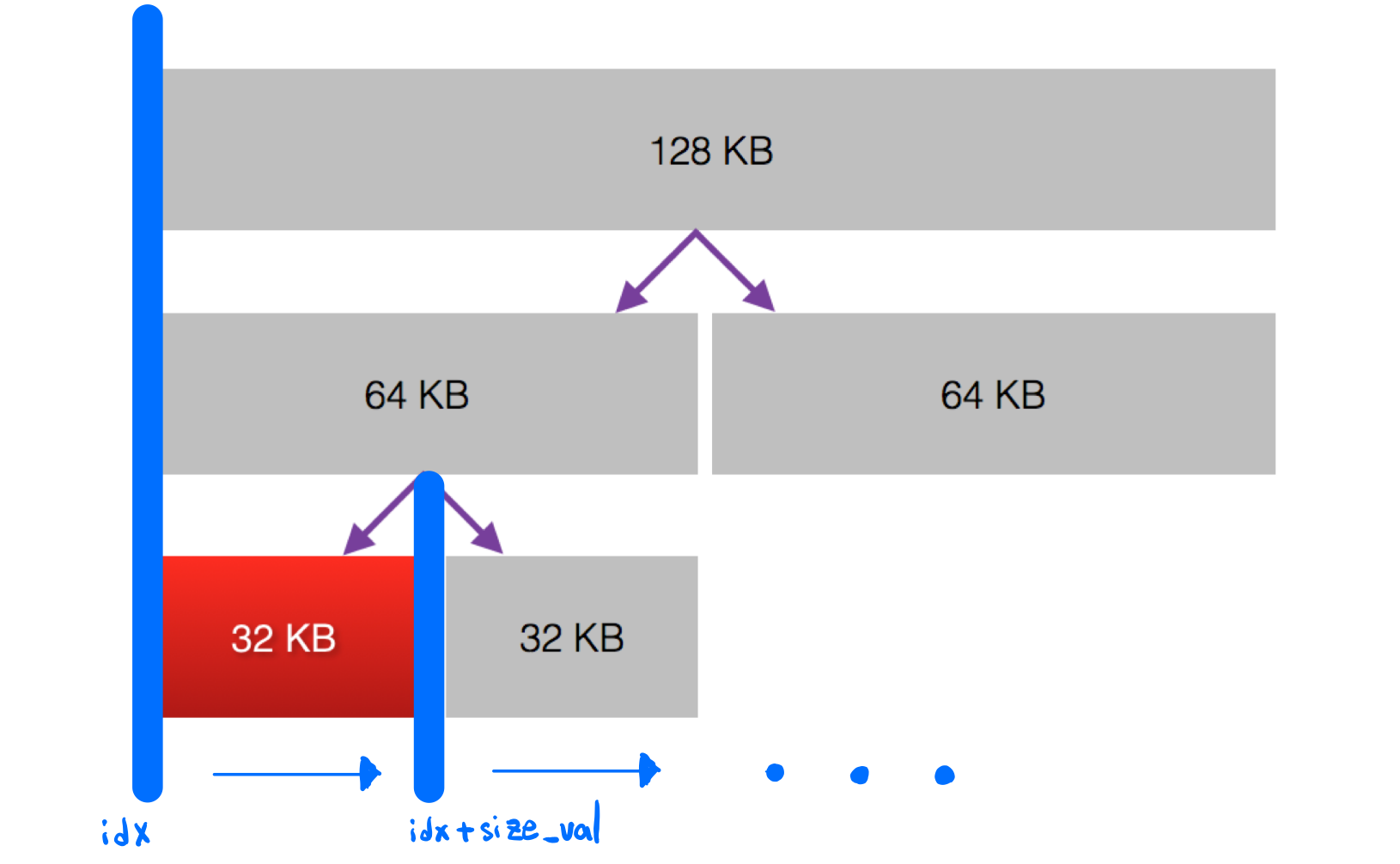
- 반복문은 start부터 시작합니다. 만약 할당이 가능하다면 now\_start에 해당 위치를 저장하고 최대 할당가능한 크기까지 now start 변수를 증가시켜주면서 while문을 통해 확인합니다. 예를 들어서 시작이 3이고 10에서 종료되었다면 7만큼 할당 가능함으로 now\_start에 i를 빼줍니다. prv\_size는 512로 초기화되어 있습니다. 지금 검사한 할당 가능한 연속된 크기가 cnt 보다 크고 prv\_size보다 작다면 idx 변수에 i를 저장하고 prv\_size에는 now\_start를 저장합니다. 그리고 i는 검사했던 크기만큼 skip하여 이와 같은 과정을 반복합니다.

- 이해를 돕기위해 cnt = 5, prv\_size = 7라고 가정하겠습니다. 만약 이후 위 과정을 통해 6 만큼 연속 할당이 가능한 인덱스를 얻었습니다. 그러면 우리는 cnt(=5) < 6, prv\_size(=7) > 6 으로 cnt만큼 할당도 가능하며 이전에 할당했던 크기보다 더 적합한 크기에 다시 할당해줄 수 있습니다.

**2.1.3 Buddy System**



- 먼저 해당 cnt를 할당하기 위해 2의 진수 크기 size\_val를 찾습니다. 첫번째 while문을 통해 cnt가 2의 진수보다 클 때 까지 2를 곱합니다. 이렇게 하면 저희는 검사와 할당을 위한 인덱스들의 정보를 얻을 수 있습니다. 다음 그림과 함께 설명하도록 하겠습니다.



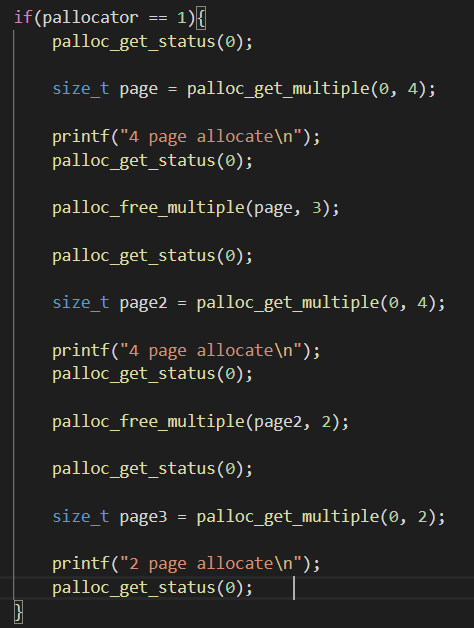
- 반복문에서 idx는 0부터 시작하여 할당가능한지 여부를 판단합니다. 여기서 bitmap\_contains 함수를 통해 idx에서 size\_val 만큼 연속적으로 메모리 할당이 가능하다면 해당 idx를 반환합니다. 만약 끝까지 검사했지만 할당이 불가능하다면 BITMAP\_ERROR을 발생시킵니다.

- 예를 들어 cnt가 125라고 가정하겠습니다. 그렇다면 우리는 size\_val 128을 구할 수 있습니다. 그리고 인덱스를 0, 128, 128+128, 128+128+128 .. 순서로 size\_val만큼의 크기만큼 더해 인덱스를 이동하면서 해당 인덱스에서 할당 가능하다면 idx를 반환시킵니다.

**2.2 테스트 진행**

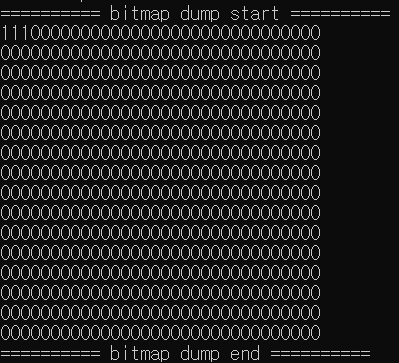
- 모든 테스트 코드는 alloctest.c에 구현하였습니다. Next fit, Best fit, Buddy System 순서로 테스트 코드 및 실행 결과를 설명하겠습니다.

**2.2.1 Next Fit test**

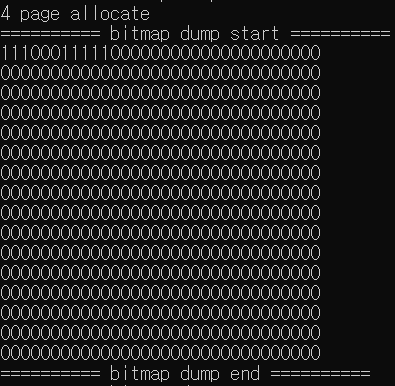
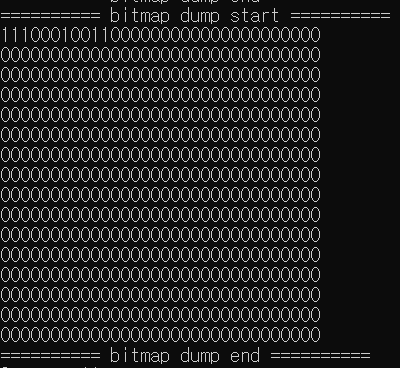


- 다음은 Next Fit의 test code 입니다. 먼저 크기 4 만큼 할당을 하고 3 만큼 free 합니다. 다시 4 만큼 할당하고 2만큼 free합니다. 그리고 2 만큼 할당해주는 과정입니다. 결과를 보겠습니다.

Ⅰ. 초기 상태에서 4만큼 할당하고 3만큼 free한 상태입니다.

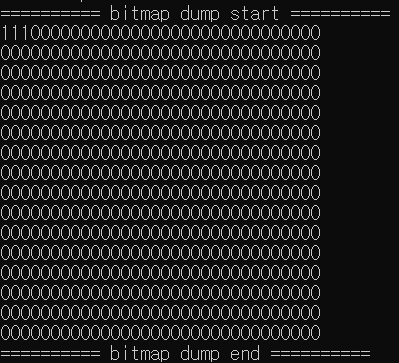
Ⅱ. 다시 4만큼 할당 후 3만큼 free 합니다. 그리고 2만큼 할당하면 제일 뒷 부분에 할당됨을 확인할 수 있습니다.

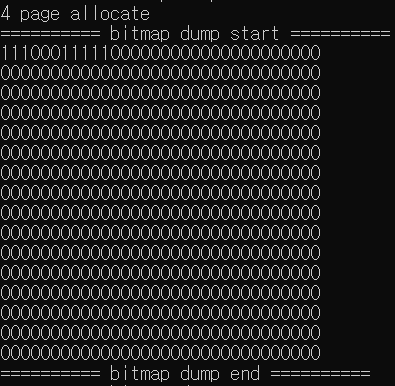
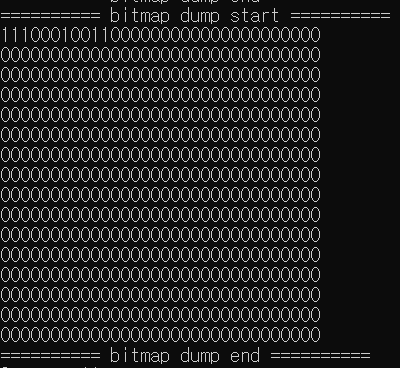
**2.2.2 Best Fit test**

- Best Fit test code는 이전 Next Fit test code와 동일합니다. 결과 차이를 확인해 보겠습니다.

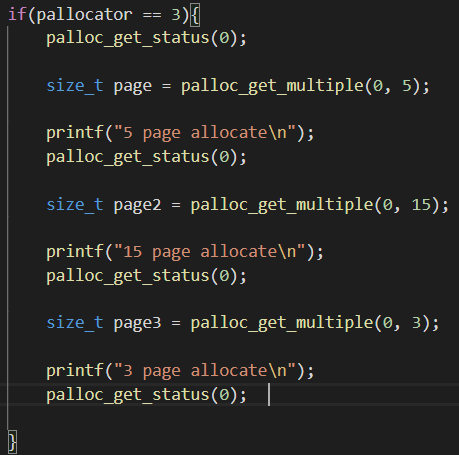
Ⅰ. 초기 상태에서 4만큼 할당하고 3만큼 free한 상태입니다.

Ⅱ. 다시 4만큼 할당 후 3만큼 free 합니다. 그리고 2만큼 할당하면 크기가 3, 2 공간중 크기가 2인 공간에 할당됨을 확인할 수 있습니다.

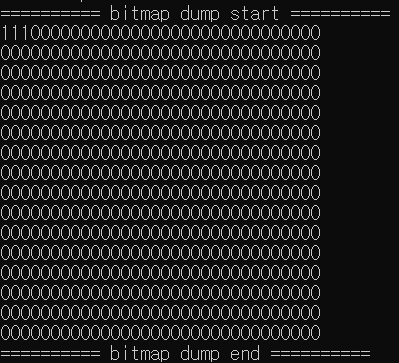
  

**2.2.1 Buddy System test**



- 다음은 Buddy System test code 입니다. 먼저 크기 5를 할당, 이어서 크기 15할당, 크기 3을 할당하는 코드입니다. 순차적으로 결과를 확인해 보겠습니다.

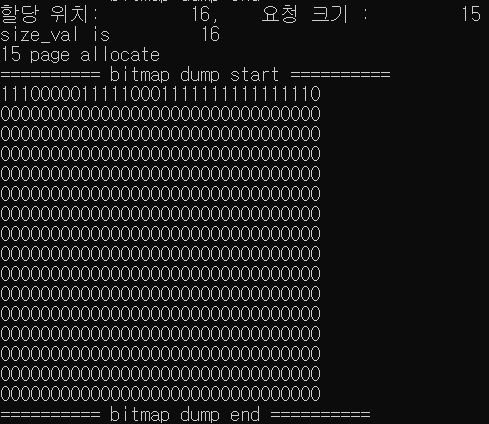
Ⅰ. 초기 상태 입니다.



Ⅱ. 크기 5를 할당했습니다. 5와 가까운 2의 진수 size\_val은 8로 계산되었습니다.



Ⅲ. 크기 15를 할당했습니다. 15와 가까운 2의 진수 size\_val은 16으로 계산되었습니다.



Ⅳ. 크기 3을 할당했습니다. 3와 가까운 2의 진수 size\_val은 4로 계산되었습니다. 이때 앞의 여유있는 공간에 할당됨을 볼 수 있습니다.

