2021 Fall System Programming

Malloc Lab 1

2021. 11. 22

유주원
clearlyhunch@gmail.com

Embedded System Lab.
Dept. of Computer Science & Engineering
Chungnam National University

실습 소개

- 과목 홈페이지
  - 충남대학교 사이버캠퍼스 (http://e-learn.cnu.ac.kr/)

- 연락처
  - 유주원
  - 공대 5호관 533호 임베디드 시스템 연구실
  - clearlyhunch@gmail.com
    - Email 제목은 **'[시스템프로그래밍][01][학번_이름]용건'** 으로 시작하도록 작성

개요

- 실습 명
  - Malloc Lab
- 목표
  - Dynamic Allocator를 구현
- 구현사항
  - malloc, realloc, free를 구현
  - 다양한 알고리즘을 적용하여 높은 성능을 갖도록 함

Malloc Lab

- Malloc Lab에서는 C언어로 dynamic storage allocator를 구현하게 된다.
- 이를 위해 각자의 malloc, free, realloc, calloc 알고리즘을 구현해야 한다.
- 또한, 메모리 공간의 생성 디자인을 고려하면서 정확하고 효율이 좋은 빠른 allocator를 구현해야 한다.
- Malloc Lab은 총 4가지의 방식으로 진행된다.
  - Naive
  - Implicit
  - Explicit
  - Segregated
    - Segregated는 실습 일정상 진행하지 않음
- 1주차에서는 Naive와 implicit을 구현한다.

CNU Embedded System Laboratory

Malloc Lab – Free Block 검색 알고리즘

- Free block은 현재 프로그램에서 할당 가능한 유휴공간을 의미한다. 따라서 메모리 할당을 하기 위해서는 free block을 검색해야 한다.
- Free block을 찾기 위한 알고리즘은 다음과 같다.
  - First fit
    - list의 처음부터 찾아 나가며, 첫 번째로 찾은 free block을 선택
    - 모든 블록을 스캔 하는데, 선형적인 시간(linear time)이 소모

```
p = start;
while       ( ( p < end ) &&       // not passed end
              ( ( *p & 1 ) ||      // already allocated
              ( *p <= len ) ) )    // too small
p = p + ( *p & -2 )                // goto next block
```

  - Next fit
    - First fit과 같게 동작하나 이전에 수행한 탐색 위치에서부터 free block을 찾기 시작
    - 단편화가 심함
  - Best fit
    - list에서 할당하고자 하는 크기와 가장 가까운 크기의 free block을 선택하여 할당
    - 단편화가 적음
    - 일반적으로 first fit 방식보다 느림

Malloc Lab – 구성 요소

- Malloc Lab 파일 복사 및 압축해제
  - cp /home/sys01/sys01/week09/malloclab-handout.tar ~
  - tar xvf malloclab-handout.tar
- Malloc Lab 구성

파일	설명
mm-naive.c	제공된 코드로 malloc과 realloc이 구현되어 있음
mm-implicit.c	implicit free list 방식으로 malloc, realloc, free 구현
mm-explicit.c	explicit free list 방식
mm-seglist.c	segregated free list 방식(구현 안함)
memlib.c	메모리 시스템을 시뮬레이션 하기 위한 모듈
mdriver.c	malloclab을 테스트 하기 위한 프로그램 코드
traces	malloclab test case를 모아둔 디렉터리

CNU Embedded System Laboratory

Malloc Lab – 설정 및 빌드

- Malloc Lab은 각 allocation 방식에 맞게 설정 후 빌드를 해주어야 한다.
  - 아래는 각 allocation 방식 설정 방법이다.

Allocation 방식	make 명령어
Naive	make naive
Implicit	make implicit
Explicit	make explicit

  - make 할 때, 자동으로 'ln-s' 명령을 통해서 각 allocation 소스코드의 링크 파일(mm.c)을 만든다.
  - 따라서 각 allocation 방식을 테스트 할 때마다 make [allocation] 해주어야 한다.
  - 설정 후에 'make'명령을 입력하여 컴파일 한다.
- make 명령통해 각 방식 별로 컴파일 하기 이전에 'make clean' 명령을 통해 이전에 컴파일 된 파일들을 제거하고 수행해준다.

CNU Embedded System Laboratory

Malloc Lab – 설정 및 빌드

- Allocation 방식 설정 및 빌드

```
clock.c   fcyc.h    ftimer.h       memlib.c       mm.h           mm-seglist.c
clock.h   fsecs.c   Makefile       memlib.h       mm-implicit.c  README
config.h  fsecs.h   malloclab.pdf  mm.c           mm-naive.c     traces
fcyc.c    ftimer.c  mdriver.c      mm-explicit.c  mm-orig.c
sys02@localhost:~/malloclab-handout$ make naive
rm -f mm.c mm.o; ln -s mm-naive.c mm.c
```

- 빌드 결과

```
sys02@localhost:~/malloclab-handout$ ls
clock.c   fcyc.h    ftimer.h       memlib.c       mm.h           mm-seglist.c
clock.h   fsecs.c   Makefile       memlib.h       mm-implicit.c  README
config.h  fsecs.h   malloclab.pdf  mm.c           mm-naive.c     traces
fcyc.c    ftimer.c  mdriver.c      mm-explicit.c  mm-orig.c
```

  - mm.c의 상단에 주석으로 아래와 같이 적혀있다.
    - mm-naive.c - The fastest, least memory-efficient malloc package.

CNU Embedded System Laboratory

Malloc Lab – 설정 및 빌드

- 각 방식에 따라 한번에 빌드하는 방법

방식	make 명령어
Naive	make clean; make naive; make
Implicit	make clean; make implicit; make
Explicit	make clean; make explicit; make

CNU Embedded System Laboratory

Malloc Lab – 필수 사항

- 각 방식 별, 소스파일 상단에 학번과 이름을 입력한다.
  - mm-naive.c
  - mm-implicit.c
  - mm-explicit.c

```
/*
 * mm-implicit.c - an empty malloc package
 *
 * NOTE TO STUDENTS: Replace this header comment with your own header
 * comment that gives a high level description of your solution.
 *
 * @id : 학 번
 * @name : 이 름
 */
```

- 해당 파일 외에는 절대 수정을 하지 않는다.(mm.h 등)
- 외부 메모리 관리 라이브러리 및 시스템 콜 사용 불가.
- 배열, 트리, 리스트 같은 전역 자료 구조의 선언 금지.

CNU Embedded System Laboratory

Malloc Lab – 점수 확인

- Malloc Lab 테스트
  - ./mdriver 를 통해 구현 내용을 테스트 한다.
  - 여러 개의 trace들을 실행시켜 정확성(correctness), 공간 활용도(utilization), 처리량(throughput)을 계산한다.

```
[c0000000000@eslab malloclab-handout]$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
   yes     94%      10  0.000000 68394 ./traces/malloc.rep
   yes     77%      17  0.000000 91250 ./traces/malloc-free.rep
   yes    100%      15  0.000000 51924 ./traces/corners.rep
 * yes     71%    1494  0.000016 94386 ./traces/perl.rep
 * yes     68%     118  0.000001 93179 ./traces/hostname.rep
 * yes     65%   11913  0.000103115651 ./traces/xterm.rep
 * yes     23%    5694  0.000064 88723 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000069 84858 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000076 87679 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000063 85902 ./traces/expr-bal.rep
 * yes      0%   14400  0.000143100760 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000064 75161 ./traces/random-bal.rep
 * yes     55%    6000  0.000056107450 ./traces/binary-bal.rep
10         41%   62295  0.000654 95214

Perf index = 26 (util) + 40 (thru) = 66/100
```

naive 를 테스트 한 결과

CNU Embedded System Laboratory

Malloc Lab – 점수 확인

- Malloc Lab 특정 파일 테스트
  - 테스트에 이용되는 trace 파일을 이용해서 개별적으로 테스트가 가능하다.
  - ./mdriver -f [파일경로/파일명]

```
[cOOOOOOOOO@eslab malloclab-handout]$ ./mdriver -f ./traces/coalescing-bal.rep
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
 * yes     0%   14400  0.000148 97587 ./traces/coalescing-bal.rep
 1         0%   14400  0.000148 97587

Perf index = 0 (util) + 40 (thru) = 40/100
```

Test case인 coalescing를 naive 에서 테스트 한 결과

Malloc Lab – 진행 참고 사항

- mdriver의 옵션을 잘 사용할 줄 알아야 한다.
- gdb를 사용해서 변화하는 값들을 살펴보면서 진행해야 한다.
- malloc에 사용 가능한 여유 공간을 효율적으로 사용해야 한다.
  - Space Utilization
    - 요청을 처리하기 위해서 얼마의 메모리 공간을 사용하는가
  - Speed
    - 단위 시간 동안 allocate/free를 얼마나 처리할 수 있는가

- 위의 Space utilization과 Speed 사이의 trade-off에 대해서 이해해야 한다.
- 구조체를 많이 사용 할 수록 space utilization이 떨어지고, 그렇지 않으면 speed가 떨어진다.

Naive

기본 방식

CNU Embedded System Laboratory

Malloc Lab – 메모리 할당 관련 함수

- Malloc Lab에서는 아래와 같은 메모리 시뮬레이션 함수를 사용한다.
  - memlib.c에 구현되어 있다.

함 수	설 명
void *mem_sbrk(int incr)	heap의 크기를 incr 만큼 증가시켜 공간을 할당
void *mem_heap_lo(void)	heap의 첫 번째를 가리키는 포인터를 반환
void *mem_heap_hi(void)	heap의 마지막을 가리키는 포인터를 반환
size_t mem_heapsize(void)	heap의 현재 크기를 반환
size_t mem_pagesize(void)	시스템의 page 크기를 반환하는 함수 (linux 시스템에서는 4K)

CNU Embedded System Laboratory

Malloc Lab – Naive의 사용 매크로

- naive에서 사용하는 매크로
  - mm-naive.c 파일을 열어 정의된 매크로를 확인한다.

```
/* single word (4) or double word (8) alignment */
#define ALIGNMENT 8

/* rounds up to the nearest multiple of ALIGNMENT */
#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)


#define SIZE_T_SIZE (ALIGN(sizeof(size_t)))

#define SIZE_PTR(p)  ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
```

  - Alignment를 double word로 설정하여 allocation 한다.
  - Allocation 하는데 필요한 각 사이즈를 매크로로 정의하였다.
  - SIZE_T_SIZE는 본 실습 환경에서는 8이다.(64bit 환경)

- 각 매크로의 연산과정에 대하여 이해하는 것이 중요하다.
  - 보고서에 해당 내용 첨부하여 코드 설명

CNU Embedded System Laboratory

Malloc Lab - Naive의 구성 함수

- malloc()
  - naive 에서 malloc 함수는 단순히 heap을 늘려가며 공간을 할당하는 방식을 사용한다.

```
/*
 * malloc - Allocate a block by incrementing the brk pointer.
 *      Always allocate a block whose size is a multiple of the alignment.
 */
void *malloc(size_t size)
{
  int newsize = ALIGN(size + SIZE_T_SIZE);
  unsigned char *p = mem_sbrk(newsize);
  //dbg_printf("malloc %u => %p\n", size, p);

  if ((long)p < 0)
    return NULL;
  else {
    p += SIZE_T_SIZE;
    *SIZE_PTR(p) = size;
    return p;
  }
}
```

  - mem_sbrk()함수는 memlib.c 안에 정의되어 있음.

CNU Embedded System Laboratory

Malloc Lab - Naive의 구성 함수

- mem_sbrk()
  - mem_sbrk()함수를 통해서 새로운 메모리를 heap 영역에 추가하도록 구성되어 있다.

```
/*
 * mem_sbrk - simple model of the sbrk function. Extends the heap
 *    by incr bytes and returns the start address of the new area. In
 *    this model, the heap cannot be shrunk.
 */
void *mem_sbrk(int incr)
{
    char *old_brk = mem_brk;

    if ( (incr < 0) || ((mem_brk + incr) > mem_max_addr)) {
        errno = ENOMEM;
        fprintf(stderr, "ERROR: mem_sbrk failed. Ran out of memory...\n");
        return (void *)-1;
    }
    mem_brk += incr;
    return (void *)old_brk;
}
```

CNU Embedded System Laboratory

Malloc Lab - Naive의 구성 함수

- realloc()
  - realloc()함수를 호출하여 이전 블록의 크기를 바꾸고, 원본 값을 크기가 바뀐 블록에 다시 저장한다.
  - 이전 공간을 free()함수를 사용하여 해제한다.
  - memcpy(void *dest, void *src, size_t size) : src의 데이터를 dest로 size만큼 복사한다.

```
/*
 * realloc - Change the size of the block by mallocing a new block,
 *      copying its data, and freeing the old block.  I'm too lazy
 *      to do better.
 */
void *realloc(void *oldptr, size_t size)
{
  size_t oldsize;
  void *newptr;

  /* If size == 0 then this is just free, and we return NULL. */
  if(size == 0) {
    free(oldptr);
    return 0;
  }

  /* If oldptr is NULL, then this is just malloc. */
  if(oldptr == NULL) {
    return malloc(size);
  }

  newptr = malloc(size);

  /* If realloc() fails the original block is left untouched  */
  if(!newptr) {
    return 0;
  }

  /* Copy the old data. */
  oldsize = *SIZE_PTR(oldptr);
  if(size < oldsize) oldsize = size;
  memcpy(newptr, oldptr, oldsize);

  /* Free the old block. */
  free(oldptr);

  return newptr;
}
```

CNU Embedded System Laboratory

Malloc Lab – Naive 빌드 및 점수 확인

- mm-naive.c 컴파일

방식	make 명령어
Naive	make clean; make naive; make
Implicit	make clean; make implicit; make
Explicit	make clean; make explicit; make

- naive 테스트 실행
  - ./mdriver를 통해 naive 코드를 수행한다.

```
[cOOOOOOOOO@eslab malloclab-handout]$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
   yes     94%      10  0.000000 68394 ./traces/malloc.rep
   yes     77%      17  0.000000 91250 ./traces/malloc-free.rep
   yes    100%      15  0.000000 51924 ./traces/corners.rep
 * yes     71%    1494  0.000016 94386 ./traces/perl.rep
 * yes     68%     118  0.000001 93179 ./traces/hostname.rep
 * yes     65%   11913  0.000103115651 ./traces/xterm.rep
 * yes     23%    5694  0.000064 88723 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000069 84858 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000076 87679 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000063 85902 ./traces/expr-bal.rep
 * yes      0%   14400  0.000143100760 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000064 75161 ./traces/random-bal.rep
 * yes     55%    6000  0.000056107450 ./traces/binary-bal.rep
10         41%   62295  0.000654 95214

Perf index = 26 (util) + 40 (thru) = 66/100
```

Malloc Lab – Naive 빌드 및 점수 확인

- mm-naive.c 분석
  - mm_init()함수가 구현되어 있지 않다.
  - free()함수가 구현되어 있지 않다.
    - malloc을 위한 free를 구현해야 하나, free 함수가 구현되어 있지 않기 때문에 성능 면에서 좋지 않은 결과를 가진다.
  - 성능: 66/100

CNU Embedded System Laboratory

Malloc Lab – Naive GDB 이용방법

1) make를 다시 한 후, gdb mdriver를 실행한다.

1) GDB 사용법 참고 https://bpsecblog.wordpress.com/gdb_memory/

```
[c0000000000@eslab malloclab-handout]$ gdb mdriver
GNU gdb (GDB) Fedora 7.7.1-18.fc20
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from mdriver...done.
(gdb)
```

3) mm_malloc에 ‘break’를 걸고 실행(run)한다.

```
(gdb) b mm_malloc
Breakpoint 1 at 0x804a67c: file mm.c, line 65.
(gdb) r
Starting program: /home/sys03/c000000000/malloclab-handout/mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Breakpoint 1, mm_malloc (size=size@entry=32) at mm.c:65
65          int newsize = ALIGN(size + SIZE_T_SIZE);
(gdb)
```

- 위와 같이 naive의 mm_malloc에 break가 걸리는 것을 확인 할 수 있음.

CNU Embedded System Laboratory

Malloc Lab – Naive GDB 이용방법

4) size는 32인 것을 확인 할 수 있고, 'list' 명령을 통해 66번째 줄에서 mem_sbrk 함수를 통해 할당 후 메모리 주소를 받는 것을 알 수 있다.

```
Breakpoint 1, mm_malloc (size=size@entry=32) at mm.c:65
65          int newsize = ALIGN(size + SIZE_T_SIZE);
(gdb) list
60       * malloc - Allocate a block by incrementing the brk pointer.
61       *      Always allocate a block whose size is a multiple of the alignment.
62       */
63      void *malloc(size_t size)
64      {
65        int newsize = ALIGN(size + SIZE_T_SIZE);
66        unsigned char *p = mem_sbrk(newsize);
67        //dbg_printf("malloc %u => %p\n", size, p);
68
69        if ((long)p < 0)
(gdb)
```

CNU Embedded System Laboratory

Malloc Lab – Naive GDB 이용방법

5) ‘display’를 통해 size와 p를 확인해 볼 수 있다.

```
(gdb) display size
1: size = 32
(gdb) display p
2: p = (unsigned char *) 0x28 <error: Cannot access memory at address 0x28>
(gdb) n
66          unsigned char *p = mem_sbrk(newsize);
2: p = (unsigned char *) 0x28 <error: Cannot access memory at address 0x28>
1: size = 32
(gdb) n
69          if ((long)p < 0)
2: p = (unsigned char *) 0x805e5a0 <heap> ""
1: size = 32
(gdb)
```

- 초기 메모리 주소가 ‘0x6176a0’이라는 것을 확인 할 수 있음.

6) ‘continue’를 사용해서 계속 실행하면, 다음 할당 사이즈를 알 수 있다.

```
(gdb) c
Continuing.

Breakpoint 1, mm_malloc (size=size@entry=32) at mm.c:65
65          int newsize = ALIGN(size + SIZE_T_SIZE);
2: p = (unsigned char *) 0xa8625b0 "./traces/malloc.rep"
1: size = 32
(gdb)
```

CNU Embedded System Laboratory

Malloc Lab – Naive GDB 이용방법

7) ‘next’를 통해 실행파일을 실행하여 값을 확인해보면, heap으로부터 40만큼 증가 된 것을 확인 할 수 있다.

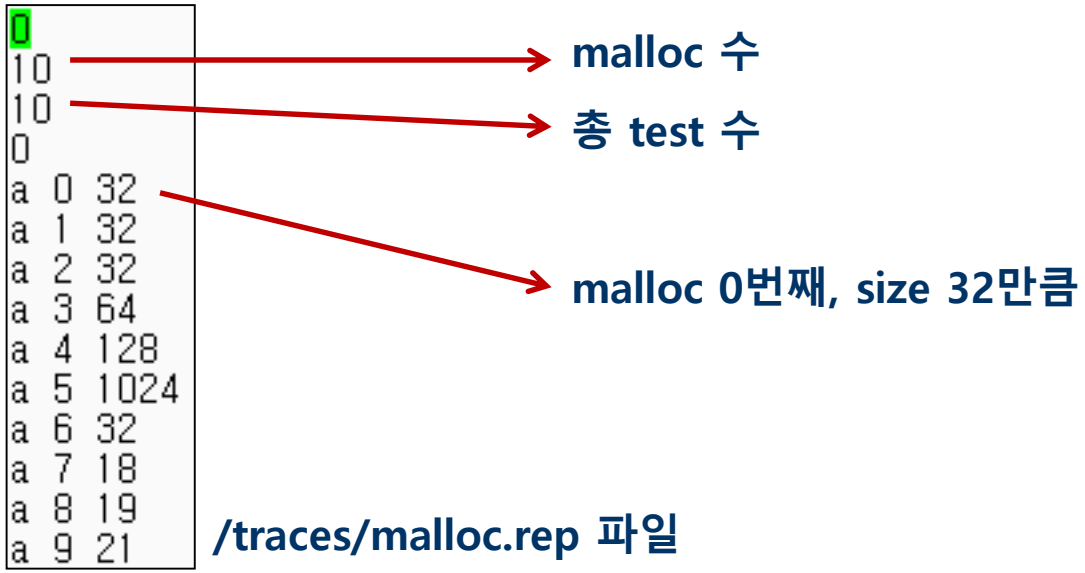
- 40만큼 증가한 이유는, 할당하려는 사이즈 32에서 header 크기인 8을 더한 크기만큼 할당해야 하기 때문이다.

```
69          if ((long)p < 0)
2: p = (unsigned char *) 0x805e5a0 <heap> ""
1: size = 32
(gdb) c
Continuing.

Breakpoint 1, mm_malloc (size=size@entry=32) at mm.c:65
65          int newsize = ALIGN(size + SIZE_T_SIZE);
2: p = (unsigned char *) 0xa8625b0 "./traces/malloc.rep"
1: size = 32
(gdb) n
66          unsigned char *p = mem_sbrk(newsize);
2: p = (unsigned char *) 0xa8625b0 "./traces/malloc.rep"
1: size = 32
(gdb) n
69          if ((long)p < 0)
2: p = (unsigned char *) 0x805e5c8 <heap+40> ""
1: size = 32
(gdb)
```

CNU Embedded System Laboratory

Malloc Lab – Naive GDB 이용방법

8) mdriver의 test case들을 저장해놓은 traces 디렉터리에서 malloc.rep와 같은 파일들을 열람하여 테스트하는 사항들을 확인한다.



- Test case와 앞서 설명한 GDB를 이용한 방식을 사용하여 메모리가 정확하게 할당되는지 확인.
- implicit, explicit 또한 동일하게 gdb를 이용하여 확인할 수 있음.

implicit

간접 리스트
모든 블록을 연결한다

CNU Embedded System Laboratory

1. 초기화 mm_init()

- 우선 초기블록을 만든다

- 사용할 공간 (하나의 커다란 프리블록)을 만든다 extend_heap()

CNU Embedded System Laboratory

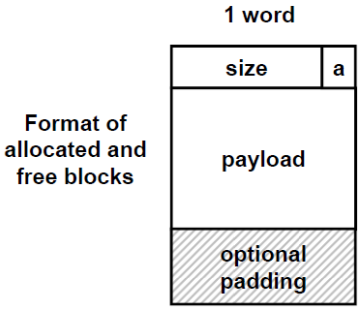
* 블록

free alloc

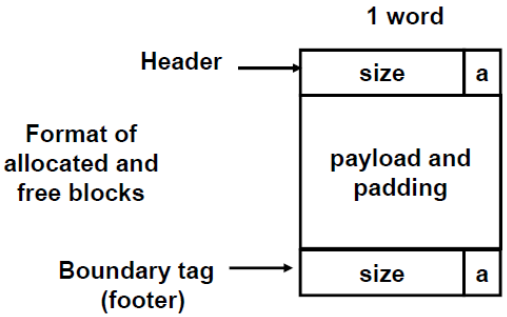
- 프리블록 또는 할당된 블록은 아래와 같이 구성된다.

  → 둘 중 하나를 선택하여 구현한다

단방향 연결



양방향 연결



CNU Embedded System Laboratory

2. Malloc & free

- **할당 시**Malloc() **프리블록을 검색하여** find_fit() **공간을 할당한다.** place()

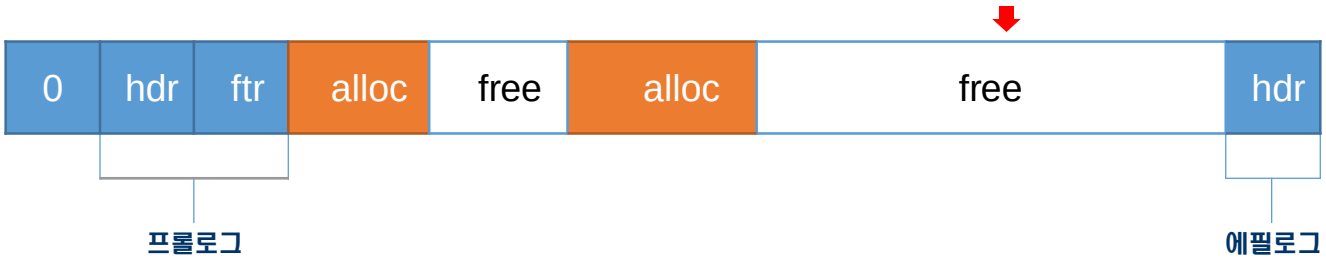
왼쪽에 나타난 그림은
* 첫 번째 블록 할당
* 두 번째 블록 할당
* 세 번째 블록 할당
* 두 번째 블록 free

- **프리블록 검색은** first, next, best **중 선택한다!**

CNU Embedded System Laboratory

2. Malloc & free

- 만약 공간이 부족하다면 사용 공간을 늘리고 extend_heap()

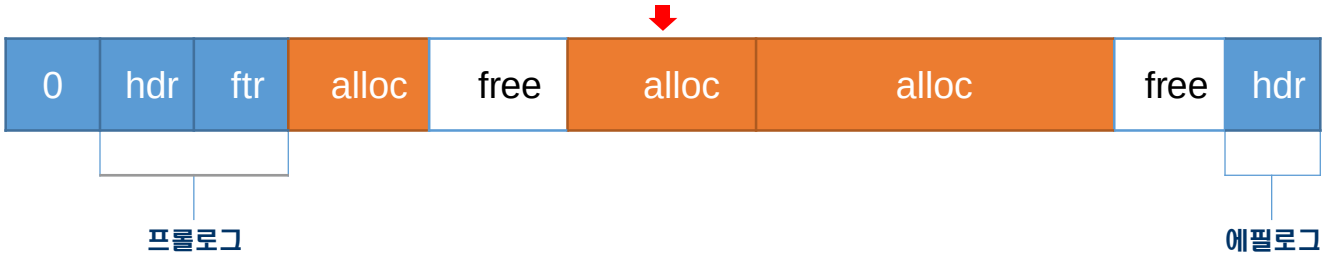


- 그 후 할당한다.

CNU Embedded System Laboratory

2. Malloc & free

- Free **할 때에는 앞 뒤 프리블록이 있는지 확인한 후**

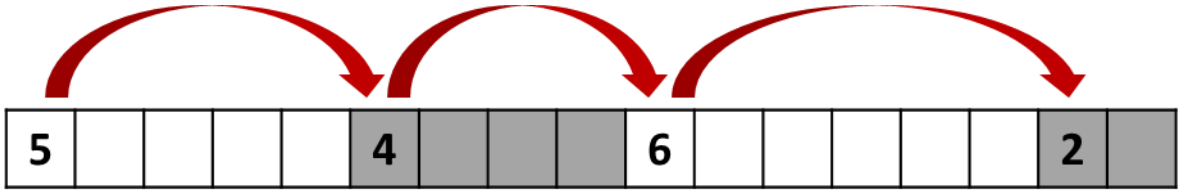


- **있으면 합쳐준다.** Coalesce()

CNU Embedded System Laboratory

Malloc Lab – Implicit

- Implicit list 방식은 각 블록의 크기를 이용해 연결된 모든 블록들을 탐색한다.



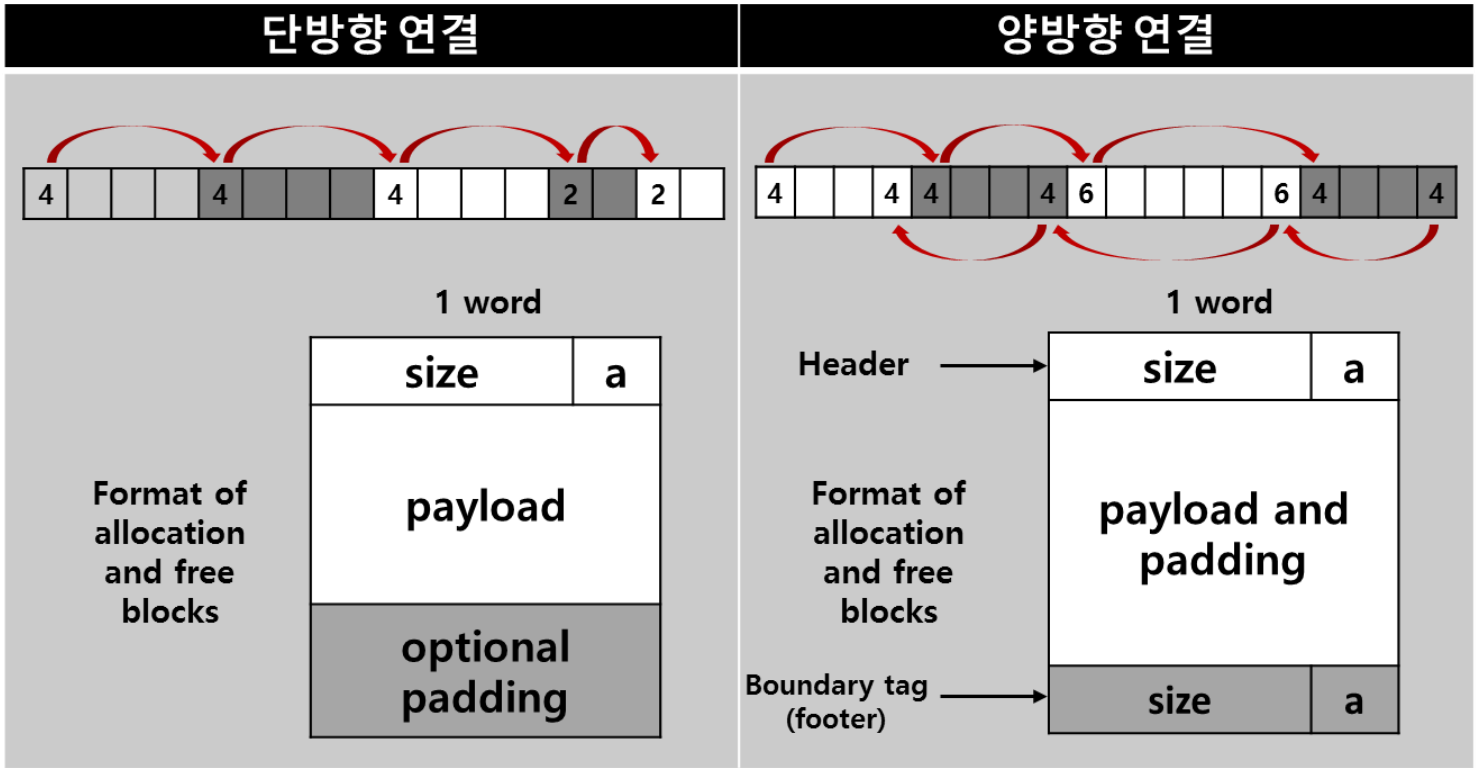
- 하지만, 모든 블록을 탐색해야 하므로 낮은 throughput을 보인다.

```
Results for mm malloc:
   valid  util    ops     secs      Kops  trace
   yes     94%      10  0.000000135153 ./traces/malloc.rep
   yes     77%      17  0.000000183662 ./traces/malloc-free.rep
   yes    100%      15  0.000000113837 ./traces/corners.rep
 * yes     71%    1494  0.000009160721 ./traces/perl.rep
 * yes     68%     118  0.000001169617 ./traces/hostname.rep
 * yes     65%   11913  0.000063188378 ./traces/xterm.rep
 * yes     23%    5694  0.000055104165 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000057103331 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000066100221 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000053101859 ./traces/expr-bal.rep
 * yes      0%   14400  0.000152 94840 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000057 84344 ./traces/random-bal.rep
 * yes     55%    6000  0.000043141040 ./traces/binary-bal.rep
10         41%   62295  0.000555112258

Perf index = 26 (util) + 40 (thru) = 66/100
```

CNU Embedded System Laboratory

Malloc Lab – Implicit 데이터 구조

- Implicit의 구현에 아래의 구조 중 선택해서 사용한다.

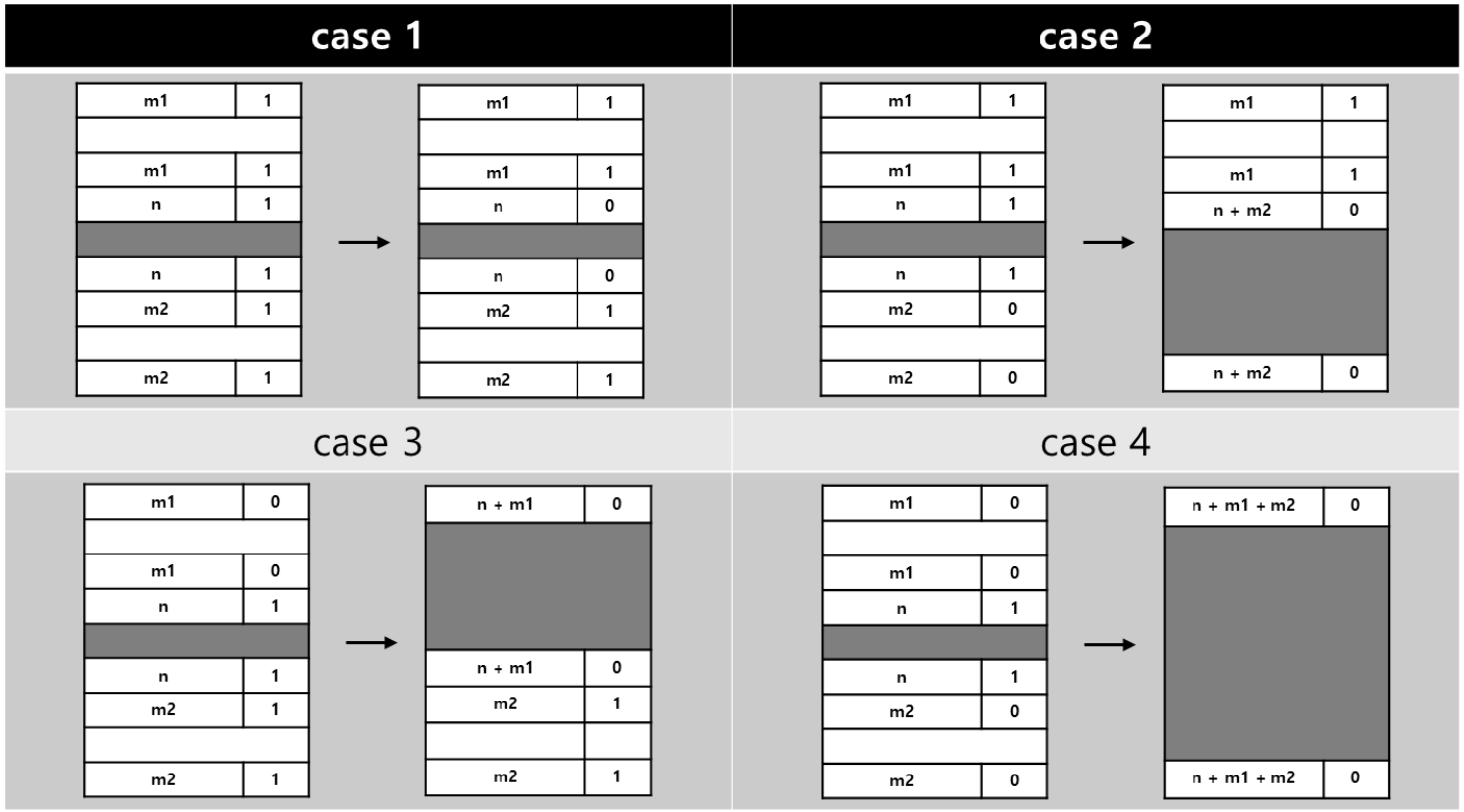


a = 1 할당된 블록 / a = 0 사용 가능한 블록
size = 블록의 크기
payload = 어플리케이션 데이터 (할당된 블록만 해당)

CNU Embedded System Laboratory

Malloc Lab – Implicit Coalescing

- 통합(Coalescing)
  - free block 상수 소요시간 연결방법



CNU Embedded System Laboratory

Malloc Lab – Implicit 구현 필요 함수

함 수	설 명
int mm_init(void)	heap을 초기화하는 함수 first block을 가리키는 포인터를 선언하여 구현
void *malloc(size_t size)	size크기의 블록을 heap에 할당
void *coalesce(void *bp)	인접한 free상태의 블록을 합쳐준다
static void place(void *bp, size_t asize)	bp 위치에 asize 크기의 메모리를 위치시켜줌
static void *extend_heap(size_t words)	요청 받은 크기의 빈 블록을 만들어 줌. 이전 block을 검사하여 case별로 free 시켜주도록 구현
static void *find_fit(size_t asize)	free block을 검색. First, Next, Best fit 알고리즘 중 하나를 선택해서 구현
void *realloc(void *oldptr, size_t size)	이미 할당되어 있는 메모리의 크기를 다시 할당
void free(void *bp)	할당된 메모리를 해제

CNU Embedded System Laboratory

Malloc Lab – Implicit 사용 매크로

매 크 로	설 명
#define WSIZE 4	word 크기 결정
#define DSIZE 8	double word 크기 결정
#define CHUNKSIZE (1 << 12)	초기 heap 크기를 설정 (bytes)(4096)
#define OVERHEAD 8	header + footer의 크기. 실제 데이터가 저장되는 공간이 아니므로 overhead가 된다.
#define MAX(x, y) ((x) > (y) ? (x) : (y))	x와 y 값 중에서 더 큰 값
#define PACK(size, alloc) ((size) | (alloc))	PACK 매크로를 사용하여 size와 alloc의 값을 하나의 word로 묶음. 쉽게 header와 footer에 저장할 수 있음
#define GET(p) (*(unsigned int *)(p))	포인터 p가 가리키는 위치에서 word크기의 값을 읽는다
#define PUT(p, val) (*(unsigned int *)(p) = (val))	포인터 p가 가리키는 곳에 word크기의 val 값을 쓴다

CNU Embedded System Laboratory

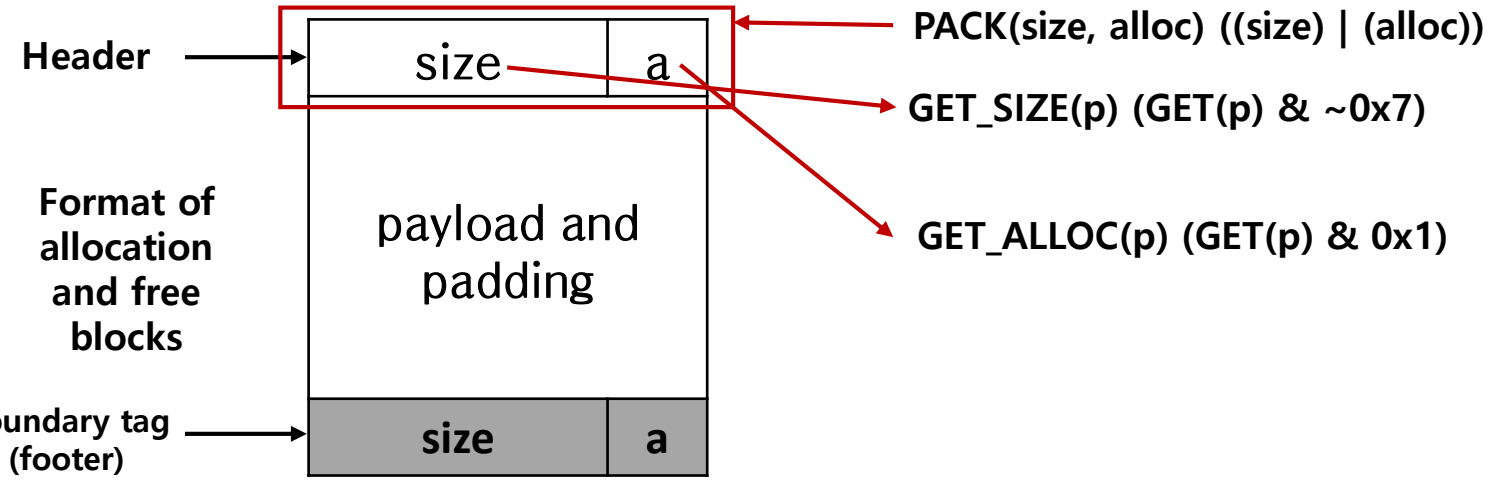
Malloc Lab – Implicit 사용 매크로

매 크 로	설 명
#define GET_SIZE(p) (GET(p) & ~0x7)	포인터 p가 가리키는 곳에서 한 word를 읽은 다음 하위 3bit을 버림 즉, Header에서 block size를 읽는 것과 같음
#define GET_ALLOC(p) (GET(p) & 0x1)	포인터 p가 가리키는 곳에서 한 word를 읽은 다음 하위 1bit을 읽음. block의 할당 여부 : 0 = No / 1 = Yes
#define HDRP(bp) ((char *)(bp) - WSIZE)	주어진 포인터 bp의 header의 주소를 계산
#define FTRP(bp) ((char*)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)	주어진 포인터 bp의 footer의 주소를 계산
#define NEXT_BLKP(bp) ((char*)(bp) + GET_SIZE((char *)(bp) -WSIZE))	주어진 포인터 bp를 이용하여 다음 block의 주소를 계산
#define PREV_BLKP(bp) ((char*)(bp) - GET_SIZE((char *)(bp) -DSIZE))	주어진 포인터 bp를 이용하여 이전 block의 주소를 계산

CNU Embedded System Laboratory

Malloc Lab – Implicit 매크로 사용 예제

- 매크로를 통해서 필요한 값을 쉽게 획득할 수 있다.



- HDRP(bp) ((char *)(bp) – WSIZE)
  FTRP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp)) - DISZE)
  GET(p) (*(size_t *)(p))
  - 사용 예
    - if(GET(HDRP(bp)) != GET(FTRP(bp)))
      - printf("ERROR : header dose not match footer\n");
    - 현재 블록 포인터 bp를 이용해서 Header와 Footer의 정보가 동일한지 검사

Malloc Lab – Implicit 매크로 사용 예제

- PUT(p, val) (*(size_t *)(p) = (val))
  - 사용 예
    - PUT(HDRP(bp), PACK(size, 0))
      - bp의 header에 block size와 alloc = 0 을 저장
      - 즉, 데이터를 지워주는 효과를 주고자 할 때 사용

- NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE(((char *)(bp) - WSIZE)))
  PREV_BLKP(bp) ((char *)(bp) - GET_SIZE(((char *)(bp) - DSIZE)))
  - 사용 예
    - size = GET_SIZE(HDRP(PREV_BLKP(bp)))
      - 이전 블록의 크기를 얻어온다. PREV_BLKP를 사용해서 bp의 이전 블록을 쉽게 알 수 있다.
    - PUT(FRTP(NEXT_BLKP(bp)), PACK(size, 0))
      - 다음 블록의 footer에 size와 alloc = 0 을 할당

CNU Embedded System Laboratory

Malloc Lab – Implicit의 구현 과정

1) 앞에서 제공한 매크로를 mm-implicit.c 파일에 작성한다.
2) first block의 포인터를 전역 변수로 선언한다.
   - static char *heap_listp = 0;
3) mm_init 함수를 아래와 같이 구현한다(교과서 참조).

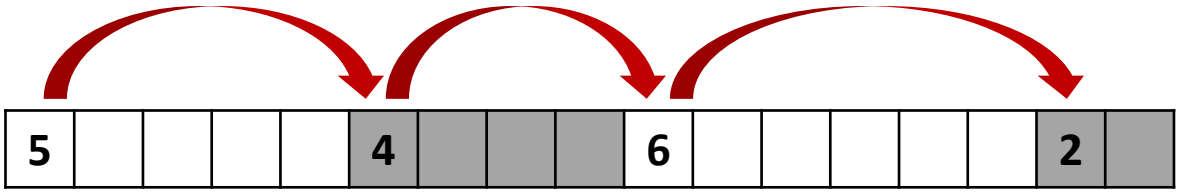
```
int mm_init(void) {

        if((heap_listp = mem_sbrk(4 * WSIZE)) == NULL)   // 초기 empty heap 생성
                return -1;                               // heap_listp = 새로 생성되는 heap 영역의 시작 주소

        PUT(heap_listp, 0);                              // 정렬을 위해서 의미없는 값을 삽입
        PUT(heap_listp + WSIZE, PACK(OVERHEAD, 1));      // prologue header
        PUT(heap_listp + DSIZE, PACK(OVERHEAD, 1));      // prologue footer
        PUT(heap_listp + WSIZE + DIZE, PACK(0, 1));      // epilogue header
        heap_listp += DSIZE;

        if((extend_heap(CHUNKSIZE / WSIZE)) == NULL)     // CHUNKSIZE 바이트의 free block 만큼 empty heap을 확장
                                                         // 생성된 empty heap을 free block으로 확장
                return -1;                               // WSIZE로 align 되어있지 않으면 에러

        return 0;
}
```

- 맨 처음 heap을 생성하는 함수로, 메모리 공간(CHUNKSIZE)을 생성
- 메모리 공간(heap)에 다수의 블록을 생성하여 사용한다.
- 초기 생성하는 블록은 16바이트의 크기를 가짐
- heap_listp의 위치를 header와 footer 사이로 이동시킴
- CHUNKSIZE만큼 heap확장

CNU Embedded System Laboratory

Malloc Lab – Implicit의 구현 과정

4) implicit에 맞게 malloc 함수를 구현



5) free 함수는 아래와 같이 구현(교과서 참조)

```
void mm_free(void *bp){

        if (bp == 0) return;    // 잘못된 free 요청인 경우 함수를 종료한다. 이전 프로시져로 return
        size_t size = GET_SIZE(HDRP(bp));       // bp의 헤더에서 block size를 읽어온다

        // 실제로 데이터를 지우는 것이 아니라
        // header와 footer의 최하위 1 bit (1, 할당된 상태) 만을 수정

        PUT(HDRP(bp), PACK(size, 0));   // bp의 header에 block size와 alloc = 0 을 저장
        PUT(FTRP(bp), PACK(size, 0));   // bp의 footer에 block size와 alloc = 0 을 저장

        coalesce(bp);   // 주위에 빈 블록이 있을 시 병합한다

}
```

6) realloc함수는 naive와 동일하게 구현

CNU Embedded System Laboratory

Malloc Lab – Implicit의 구현 과정

7) void *coalesce(void *bp) 함수를 구현

- 빈 공간을 합쳐주는 역할을 하는 함수

```
static void *coalesce(void *bp){

        size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
        // 이전 블럭의 할당 여부 0 = NO, 1 = YES

        size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
        // 다음 블럭의 할당 여부 0 = NO, 1 = YES

        size_t size = GET_SIZE(HDRP(bp));
        // 현재 블럭의 크기

        /*
         * case 1 : 이전 블럭, 다음 블럭 최하위 bit가 둘다 1 인 경우 (할당)
         *          블럭 병합 없이 bp return
         */
        if(prev_alloc && next_alloc){
                return bp;
        }

        /*
         * case 2 : 이전 블럭 최하위 bit가 1이고 (할당), 다음 블럭 최하위 bit가 0 인 경우 (비할당)
         *          다음 블럭과 병합한 뒤 bp return
         */
        else if(prev_alloc && !next_alloc){
                size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
                PUT(HDRP(bp), PACK(size, 0));
                PUT(FTRP(bp), PACK(size, 0));
        }
}
```

CNU Embedded System Laboratory

Malloc Lab – Implicit의 구현 과정

7) void *coalesce(void *bp) 함수를 구현

```
/*
 * case 3 : 이전 블럭 최하위 bit가 0이고 (비할당), 다음 블럭 최하위 bit가 1인경우 (할당)
 *          이전 블럭과 병합한 뒤 새로운 bp return
 */
else if( ){

}

/*
 * case 4 : 이전 블럭 최하위 bit가 0이고 (비할당), 다음 블럭 최하위 bit가 0인경우 (비할당)
 *          이전 블럭, 현재 블럭, 다음 블럭을 모두 병합한 뒤 새로운 bp return
 */
else{

}

return bp;
// 병합된 블럭의 주소 bp return
```

❖ free block 상수 소요시간 4가지 경우를 이용해 연결방법을 구현

CNU Embedded System Laboratory

Malloc Lab – GDB를 이용한 implicit 디버깅

- 모든 코드를 작성하고 본인이 작성한 코드가 정확하게 메모리를 할당하고 해제하는지 GDB를 통해 확인한다.

1) 앞에서 설명한 gdb 수행 방법과 같이, 컴파일 하여 gdb를 통해 mdriver를 실행
2) mm_init 함수에 breakpoint를 설정하고 실행시켜 초기 설정이 제대로 이루어지고 있는지 확인
3) mm_init함수에서 heap_listp 사이즈를 증가시켜주는 부분에 breakpoint를 설정하고 c를 눌러 이동
4) x/4x heap_listp 명령을 통해 초기 prologue block과 epilogue block이 할당된 메모리의 상태를 확인

```
(gdb) x/4x heap_listp
0x805e5c0 <heap>:    0x00000000    0x00000009    0x00000009    0x00000001
```

padding | header | footer | epilogue

prologue

CNU Embedded System Laboratory

Malloc Lab – GDB를 이용한 implicit 디버깅

5) 이후 mm_malloc 함수에 breakpoint를 설정하고 메모리가 정확하게 할당되고 있는지 확인

```
Breakpoint 2, mm_malloc (size=size@entry=32) at mm.c:127
127     void *malloc (size_t size) {
```

6) 다시 한번 c를 눌러 진행을 하고 메모리 상태를 확인한다.

- x/16x heap_listp 명령을 통해 메모리가 할당하고자 하는 크기로 할당이 되고 있는지 확인

```
(gdb) x/16x heap_listp
0x805e5c8 <heap+8>:     0x00000009      0x00000029      0xff8fd5e2      0x5dd631b4
0x805e5d8 <heap+24>:    0x409d624d      0xa8b8a0fa      0xc56bf6d5      0x00918f55
0x805e5e8 <heap+40>:    0x2a4b7c80      0x93a6a4b0      0x00000029      0x00001fd8
0x805e5f8 <heap+56>:    0x00000000      0x00000000      0x00000000      0x00000000
```

header

footer

- 위 그림의 메모리 상태를 확인해보면 header와 footer가 있고, 그 사이에 32바이트(할당하고자 하는 메모리 크기)에 랜덤 값이 할당되어 있는 것을 확인 할 수 있음

CNU Embedded System Laboratory

Malloc Lab – GDB를 이용한 implicit 디버깅

- gdb에서 trace 파일 별로 실행하는 방법
  - (gdb) r -f ./trace/(테스트 할 rep 파일)

```
[b000000000@eslab malloclab-handout]$ gdb ./mdriver
GNU gdb (GDB) Fedora 7.7.1-18.fc20
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./mdriver...done.
(gdb) r -f ./traces/malloc.rep
Starting program: /home/sys02/b000000000/malloclab-handout/mdriver -f ./traces/malloc.rep
Measuring performance with a cycle counter.
Processor clock rate ~= 3392.3 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
   yes    94%      10  0.000000152123 ./traces/malloc.rep
 1         0%       0  0.000000     0

Perf index = 0 (util) + 0 (thru) = 0/100
[Inferior 1 (process 6330) exited normally]
(gdb)
```

과제 (다음주에 한번에 제출입니다.)

1. Malloc Lab
   I. Implicit방식의 malloc 구현

2. Malloc Lab 보고서
   I. mm-naive에 대한 설명
      1) naive에 사용된 **매크로 및 함수**에 대한 설명
   II. mm-implicit에 대한 설명
      1) implicit에 사용된 **매크로 및 구현 함수** 설명
   III. Naive와 Implicit 방식에 대한 ./mdriver 결과 첨부

3. 제출
   I. malloc-handout 디렉토리를 통째로 압축
      1) 파일명: [sys01]malloc_학번.tar.gz
   II. 결과 보고서를 작성
      1) 파일명: [sys01]malloc_학번.pdf
   III. I.과 II. 두개를 하나로 압축
      1) 파일명:[sys01]malloc_학번_이름.zip