2020년 2학기

Assignment #6

과목 : 컴퓨터SW시스템개론

담당 교수 : 김종

학과 : 컴퓨터공학과

학번 : 20190439

이름 : 오승훈

povis id : sho0927

|  |
| --- |
| < 명예서약(Honor code) >  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem : Malloc Lab**

1. 문제의 개요

이번 과제에서는 Heap 영역에 메모리를 동적 할당을 하는 방법에 대한 구현을 목표로 한다. 구현해야 할 함수는 다음과 같이 총 4가지로 구성이 되어 있다.

1. int mm\_init (void);

가장 먼저 함수가 call이 되어 heap 영역을 어떻게 잡을 것인지에 대해 알려주는 함수이다. 맨 앞과 뒤에 header와 footer를 넣어줘서 seg\_list 관리에 도움을 주도록 한다.

1. void \*mm\_malloc(size\_t size);

필요한 블록 사이즈가 얼마인지를 결정하고, free list들 중 어느 블록에 값이 들어갈 것인지에 대해서 결정한 후, 맞는 블록이 존재하면 그 블록에 넣어주고, coalescing 과정을 거치게 되고, 블록이 존재하지 않으면 heap 영역을 늘려서 블록을 넣어주도록 한다.

1. void mm\_free(void \*ptr);

동적 할당을 통해서 생성된 블록의 경우 할당을 풀어주고, seg\_list에 블록 크기가 맞도록 list에 넣어주도록 한다.

1. void \*mm\_realloc(void \*ptr, size\_t size);

realloc 함수의 경우 ptr이 NULL인 경우 mm\_malloc과 동일한 동작을 하도록 구현을 시키면 되고, size가 0인 경우 free와 동일한 동작을 구현해주면 된다. Ptr과 size 모두 NULL과 0이 아닌 경우에는 원래 ptr의 size를 변경해주고, return을 시키면 된다.

1. 실제 구현 방법
2. macro들에 관한 설명
3. ALIGNMENT : allocation의 영역에 대한 할당 alignment를 나타내주는 macro이다.
4. ALIGN(size) : 실제 사이즈를 기반으로 ALIGN이 되었을 때 사이즈가 몇이면 되는지를 알려주는 macro이다.
5. WSIZE word\_size인 4를 나타낸다.
6. DSIZE double word\_size인 8를 나타낸다.
7. CHUNKSIZE (1<<12) heap 영역을 해당 byte 만큼 늘리기 위해서 사용하는 macro이다.
8. 밑의 INIT, REALLOC 역시 동일하다.
9. MAX\_SEG 는 segregated list의 최대 크기를 20개로 설정해뒀기 때문에 그것을 알려주는 macro이다.
10. MAX, MIN 둘 중 큰 값과, 작은 값을 결정해서 알려주는 매크로이다.
11. PACK size와 allocation bit을 함께 header나 footer에 넣어주는 매크로이다.
12. GET, PUT, PUT\_NOTAG 등은 현재 주소에 읽고 쓰는 것을 하기 위한 용도의 매크로이다.
13. GET\_SIZE, GET\_ALLOC, GET\_TAG, SET\_RATAG, REMOVE\_RATAG는 현재 블록의 사이즈 값을 얻어오고, 할당 되어있는지를 알려주는 bit 값을 얻어오고, TAG인 REALLOCATION이 되어있는지를 알려주는 bit, RATAG를 1로 설정해주고, RATAG를 없애주는 매크로들이다.
14. HDRP, FTRP, 현재 블록의 header와 footer의 주소를 돌려주는 매크로이다.
15. NEXT\_BLKP, PREV\_BLKP 물리적으로 봤을 때 다음 블록의 주소와, 이전 블록의 주소를 돌려주는 매크로이다.
16. GET\_PREV, GET\_NEXT는 free block들 중 현재 블록을 기준으로 이전 블록과 다음 블록의 입력 주소를 돌려주는 매크로이다.
17. GET\_PREV\_BLK, GET\_NEXT\_BLK는 free block들 중 현재 블록을 기준으로 이전 블록과 다음 블록의 address를 제공해준다.
18. 함수 구현
19. mm\_init(void)

mm\_init 함수의 경우 가장 먼저 해야할 것은 \*\* pointer로 형성이 되어 있는 seglist를 seglist를 관리할 수 있도록 동적할당을 먼저 해줘야한다. 그를 위해서 mem\_sbrk(sizeof(int\*) \* MAX\_SEG)를 사용하여 int\* 의 형식을 가지는 20개의 메모리에 영역을 잡아준다. 그와 동시에 seglist에 대한 초기화를 시켜주고, heap 영역을 사용하기 위해서 heap영역을 맨 앞과 뒤에 각각 padding을 넣어줄 것인데 맨 앞 자리에 word size만큼의 header와 footer를 잡아주고, 맨 뒤에도 footer를 넣어 이상한 영역에 할당이 되지 않도록, 그리고 heap 영역이 맨 앞부터, 맨 뒤의 footer까지이다라는 것을 알려주기 위한 용도로 할당을 해둔다. 그리고 heap 영역을 extend\_heap 함수를 이용하여 늘려주면 된다.

1. extend\_heap(size\_t words)

이 함수의 경우 요청한 만큼의 size에 맞는 heap 영역을 늘려주는 함수인데 words를 먼저 ALIGN에 맞는 값을 MACRO를 이용하여 알아낸 후 mem\_sbrk 함수를 이용하여 영역을 확장을 시켜주면 된다. 확장이 되고 나면, 그 영역의 맨 앞과 맨 뒤에 현재는 할당 중이 아니므로 allocation이 되어 있지 않다는 bit과 size를 설정해준 뒤 그 블록을 seglist에 넣어 관리를 해주면 된다.

1. mm\_malloc(size\_t size)

먼저 할당을 요구하는 size를 ALIGN size에 맞는 값으로 설정을 해주고, find\_fit 함수를 이용하여 seg\_list 들 중 현재 원하는 block의 크기에 맞는 free block을 찾아서 그 block에 넣어주는 과정을 거친다. Find\_fit을 통해서 return된 포인터가 NULL이라면 현재 맞는 free block이 존재하지 않는다는 의미이기 때문에 heap 영역을 extend\_heap 함수를 이용하여 늘려주면 된다. NULL이 아니라면 정상적으로 free block을 찾은 것이다. 그 후 NULL이든 아니든 현재 block을 할당된 부분과 아닌 부분으로 split을 해주는 함수인 place 함수를 호출해주면 된다.

1. find\_fit (size\_t asize)

find\_fit의 경우 seglist들 중 first\_fit을 이용하여 구현을 하였고, seg\_list의 경우 index에 따라 0부터 19까지 1, 2~3, 4~7, 8~15, 16~31….와 같은 크기의 seglist를 관리하도록 구현을 하였다. 먼저 원하는 크기의 seg\_list size에 온 경우 그 seg\_list의 free block 중에 내가 원하는 크기의 block이 존재하는지를 확인을 한다. 현재 블록의 다음 노드들을 순차적으로 탐색하면서 원하는 크기가 있다면 그 node의 pointer 값을 return을 해서 가져가고, 아닌 경우 NULL을 반환하는 함수이다.

1. place(void\* bp, size\_t asize)

이 함수의 경우 현재 bp를 split을 해주는 함수인데 실제 블록의 크기는 우리가 원하는 size와 동일하다면 가장 좋겠지만 우리가 원하는 size보다 더 큰 경우들이 더 많이 존재할 것이다. 그러므로 우리가 원하는 size보다 더 큰 블록을 들고 왔을 경우 사용할 만큼만 블록을 사용하고 나머지 블록을 seg\_list 즉 free block list에 넣어줘야 다른 블록들이 효율적으로 allocation을 할 수 있을 것이다. 이 place를 구현하기 위해서 free 블록간의 연결을 먼저 없애주는 delete\_seg\_list를 이용하여 현재 seg\_list에 포함되어 있는 free 블록의 seg\_list간 연결을 끊어준다. 그 후 실제 블록 크기와 요구한 block size를 비교한다. 그 차이가 double word size의 두배보다 작다면 그 값은 split을 할 필요가 없으므로 요구한 size가 아닌 가져온 free block의 size로 block을 할당을 시켜주고, allocation bit을 넣어주고 return을 한다. 그 외의 경우는 두 가지 경우로 나눠서 봐야하는데 우리가 요구한 block의 size 값이 100을 넘는 경우와 100을 넘지 않는 경우 두 가지로 나눠서 봐야한다. Size가 100을 넘기는 경우 요구하는 size와의 차이만큼을 먼저 할당되지 않은 block으로 사용하여 seg\_list에 insert를 해줘야하고, 100이 넘지 않는 경우에는 먼저 요구한 size만큼 블록을 할당시킨 후 남은 size 만큼을 할당되지 않는 seg\_list에 insert를 해주는 과정을 거친 후 return을 해주면 된다.

1. insert\_seg\_list(void\* bp, size\_t asize)

seg\_list를 관리하기 위해 노드를 추가하는 함수이다. 이 함수에서는 추가하려는 블록의 크기에 맞는 seg\_list block이 몇 번째 index에 넣어주면 되는지를 확인할 후 같은 index에 존재하는 seg\_list들도 block의 크기가 작은 순으로 관리를 하면 우리가 first\_fit으로 구현을 해도 best\_fit과 같은 효과를 낼 수 있기 때문에 seg\_list를 순차적으로 탐색을 해가며 자신이 들어갈 block 공간을 찾아낸다. 이제 block이 들어갈 공간을 찾아내면 4개의 case가 존재하는데

case 1. 이전의 block이 NULL이 아니고, 다음 block은 NULL이 아닌 경우

case 2. 이전의 block이 NULL이 아니고, 다음 block은 NULL인 경우

case 3. 이전의 block이 NULL이고, 다음 block은 NULL이 아닌 경우

case 4. 이전의 block이 NULL이고, 다음 block도 NULL인 경우

이렇게 총 4가지의 경우가 존재하는데 모두 한 가지를 지키며 동작을 하게 되어 있다. 현재 블록의 전과 현재를 현재 블록에서는 previous block이 전 block인 것이고, 전 블록의 경우 next block이 현재 블록으로 인식하여 연결하고, 현재 블록과 다음 블록의 경우 현재 블록은 다음 블록을 다음 블록으로 연결 다음 블록은 현재 블록을 전 블록으로 연결해주는 과정을 거치면 된다. 그 과정에서 seg\_list에 연결을 해야하는 경우도 나오기 때문에 일부러 나눠서 구현을 해뒀다.

1. delete\_seg\_list(void\* bp)

가장 단순한 함수 중 하나이다. 현재 bp 블록의 크기에 맞는 seg\_list index를 찾아낸 후 현재 블록을 free를 해줄 것이므로, 현재 블록의 전과 다음 블록을 서로 연결해주는 과정을 거치면 된다. 그 과정에서 seg\_list에 직접적으로 연결을 해야하는 경우들이 생기기 때문에 seg\_list의 index값을 구한 것이다.

1. ret\_seg\_index(size\_t asize)

가장 단순한 함수 중 하나이다. 이 함수는 asize 값에 맞는 seg\_list index 값을 반환해주는 함수이다.

1. mm\_free(void \*ptr)

현재 ptr 값의 블록을 allocation tag를 없애주고, seg\_list에 다시 넣어주는 과정을 거치면 된다. 그리고 현재 넣어준 block이 physically 연결되어 있는 block이 free block이었는 경우 현재 우리가 지정해둔 값이 아닌 원래 free block의 크기 + 현재 블록의 크기가 되므로 coalescing 과정이 필요해진다. 그러므로 coalesce 함수를 불러서 현재 ptr의 크기를 조정해주는 과정을 거치면 된다.

1. coalesce(void\* bp)

coalesce 함수의 경우 현재 블록의 이전 블록과 다음 블록이 allocation인지 확인을 해주고, 총 4개의 경우로 나눠지게 되는데

Case1 : 이전 블록과 다음 블록 모두 할당이 되어 있는 경우

Case2 : 이전 블록은 할당이 되어있고, 다음 블록은 할당이 되어 있지 않는 경우

Case3 : 이전 블록은 할당이 되어 있지 않고, 다음 블록은 할당이 되어 있는 경우

Case4 : 이전과 다음 블록 모두 할당이 되어 있지 않는 경우

Case1의 경우 그냥 coalesce 할 것이 없으므로 그냥 return을 해주면 된다.

Case2의 경우 현재 블록과 다음 블록을 seg\_list에서 delete를 해준 뒤 현재 블록과 다음 블록의 크기만큼을 현재 블록의 header 그리고 다음 블록의 footer에 넣어주면서 할당을 나타내는 bit을 0으로 설정을 해주면 된다. 이때 return은 현재 블록을 return을 해주면 된다.

Case3의 경우 현재 블록과 이전 블록을 seg\_list에서 delete를 해준 뒤 이전 블록과 현재 블록의 크기만큼을 현재 블록의 footer 그리고 이전 블록의 header에 넣어주면서 할당을 나타내는 bit을 0으로 설정을 해주면 된다. 이때 return은 이전 블록의 주소를 return을 해줘야 seg\_list에 제대로 들어간 블록의 address를 반환해줄 수 있다.

Case4의 경우 현재 블록과 이전 블록 그리고 다음 블록 모두 seg\_list에서 delete를 해준 뒤 이전 블록, 다음 블록, 현재 블록의 크기를 모두 더해서 이전 블록의 header와 다음 블록의 footer에 그 값을 넣어주면서 할당을 나타내는 bit을 0으로 설정을 해주면 된다. 그리고 이때 역시 return은 이전 블록의 주소를 return을 해줘야 하므로 이전 블록의 address를 return 시켜준다.

이 모든 case들에서 return을 하기 전 insert\_seg\_list를 해줘야 seg\_list에 실제 free된 블록이 들어갈 수 있으므로 return을 해야하는 그 값들을 insert\_seg\_list에 넣어서 seg\_list에 넣어주고, 각각의 case에서 말한 pointer 값을 return을 시켜주면 된다.

1. mm\_realloc(void\* ptr, size\_t size)

이 realloc의 경우 일단 malloc read me를 최대한 참고를 하여 구현을 하였다. 먼저 ptr 값이 NULL인지를 확인을 해주고, NULL인 경우 mm\_malloc(size)를 해서 return을 시켜주면 된다.

그리고 size가 0인 경우에는 mm\_free(ptr)을 해준뒤 NULL 값을 return 해주면 된다.

이 외의 모든 경우들이 문제가 되는 상황인데 일단 mm\_malloc과 동일하게 먼저 ALIGN에 맞는 size 값을 구해준다. 그 후 현재 block보다 크기가 커지는 구현인지 아닌지를 구분을 하기 위해서 block\_buffer라는 변수에 현재 블록과 새로 만들 블록의 크기의 차이를 구한다. 이 값이 음수인 경우 현재 블록의 크기를 더 늘려야하는 구현이기 때문에 다음 블록(physically)이 할당이 되어 있지 않은가와 size가 0인지를 확인을 해준다. 그리고 만약 위의 조건을 만족한다면 현재 블록의 크기와 다음 블록의 크기를 더한 값에 우리가 새로 구현을 하려는 블록의 크기의 차이를 구해주면 되는데 이 값이 음수인 경우 heap을 늘려줘야 하는 상황이기 때문에 extend\_heap을 사용하여 heap 영역을 늘려주고, 현재 블록의 physically 다음 블록을 seg\_list에서 삭제를 시켜준 뒤 다시 새로운 사이즈와 remainder 값을 더한 값 즉 현재 블록과 다음 블록의 크기를 더한 값 만큼을 footer와 header에 기록을 하면서 allocation이 되었음을 알려주는 bit을 1로 설정을 해주면 된다.

그리고 위의 allocation이 1이고, block size가 0이 아닌 상황이 발생을 하면 이 경우에는 새로운 영역을 잡아서 할당을 해줘야 하는 상황이기 때문에 현재 size에서 DSIZE 만큼을 뺀 즉 ALIGN이 된 값 만큼을 빼주고, 다시 mm\_malloc에 넣어주면 된다. 그 후 현재 pointer에 저장되어있는 정보를 새로운 포인터에 복사를 해줘야하는데 MIN(size, new\_size) 만큼을 해주는 이유는 block의 크기가 커지는 경우 앞에서부터 채워야하고, block의 크기가 작아지는 경우 앞에서부터 작아지는 block의 크기만큼의 정보만을 복사를 해줘야하기 때문에 MIN size를 골라서 복사를 해줘야한다. 그리고 원래의 ptr값은 필요가 없어지므로 free를 시켜주면 된다. 그리고 새로운 ptr에는 realloc이 진행됐음을 알려주는 bit을 설정을 해주고, return을 시켜주면 된다.

현재 block size가 더 작아지는 것에 대해서는 고려를 안해줬는데 그 이유는 block size가 더 작아지는 경우에는 block의 크기를 조정을 할 이유가 없기 때문이다.

1. 토론

이번 과제는 오히려 책에 모든 내용들이 언급이 되어 있고, 수업 시간에 배운 내용을 바탕으로 구현을 할 수 있는 정도의 난이도였던 것 같았다. 하지만 어렵게 작용 했던 점은 macro들을 설정을 하는 것이 아무래도 익숙하지 않아서 macro를 설정을 하는 것과 우리가 linked list를 생각했을 때 보통은 pointer를 이용한 pointer\_next 등과 같은 특수한 값을 이용하여 구현을 하는데 이 경우에는 그러한 것이 아니라 macro를 이용하여 구현을 해뒀다는 점이 새롭게 다가온 점인 것 같다.

이번 과제는 조금 수월하게 진행이 되었는 것 같지만 사실 책에 코드가 나와있지 않았다면 힘들었을 수도 있을 것이라고 생각하고, segregate allocate list에 대해서 개념만 들었을 때는 이해가 잘 되지 않았지만 실제 구현을 해본 뒤 이렇게 활용을 할 수 있구나라는 것을 많이 깨달은 것 같고, segregate list도 관리를 하기 위해서 block의 크기 별로 블록을 넣어주는 방법이 first\_fit을 사용해도 best\_fit과 동일한 결과를 초래할 수 있다는 점을 잘 확인할 수 있는 과제가 되었다고 생각된다.