Memory API

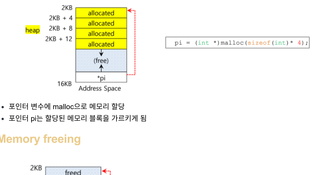
malloc()

- void* malloc(size_t size)
- heap에 있는 메모리 영역 할당
- malloc()으로 할당된 메모리 블록을 가르키는 포인터 반환

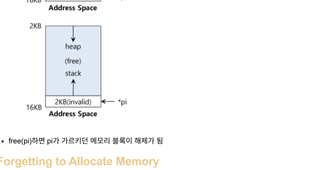
free()

- void free(void* ptr)
- malloc()으로 할당된 메모리 영역 해제
- malloc()으로 할당된 메모리 블록을 가르키는 포인터를 인자로 받음

Memory Allocating

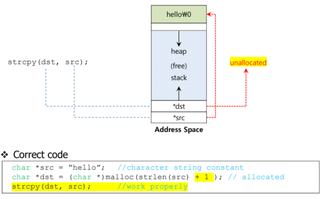


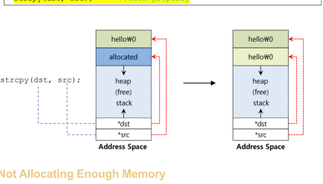
- 포인터 변수 생성.
- 지역변수이므로 스택에 저장



- 포인터 변수에 malloc으로 메모리 할당
- 포인터 pi는 할당된 메모리 블록을 가르키게 됨

Memory freeing





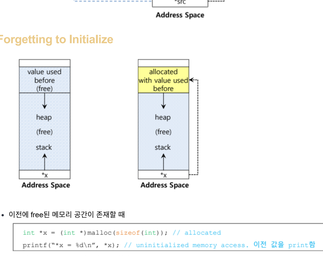
- free(pi)하면 pi가 가르키던 메모리 블록이 해제가 됨

Forgetting to Allocate Memory

❖ Incorrect code

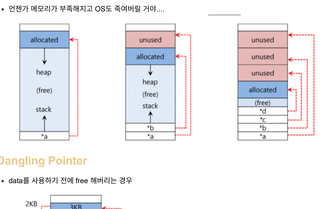
```
char *src = "hello"; //character string constant
char *dst;           //unallocated
strcpy(dst, src);    //segfault and die
```

gcc로 compile & run 하면, segmentation fault 발생



❖ Correct code

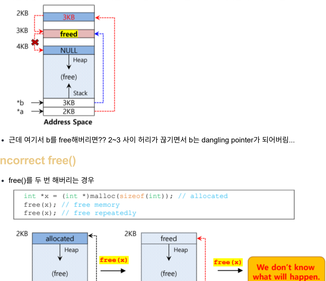
```
char *src = "hello";   //character string constant
char *dst = (char *)malloc(strlen(src) + 1); // allocated
strcpy(dat, src);      //work properly
```



Not Allocating Enough Memory

❖ Incorrect code, but work properly

```
char *src = "hello"; //character string constant
char *dst (char *)malloc(strlen(src)); // too small
strcpy(dst, src);    //work properly, but...
```



Forgetting to Initialize

- 이전에 free된 메모리 공간이 존재할 때

```
int *x = (int *)malloc(sizeof(int)); // allocated
printf("*x = %d\n", *x); // uninitialized memory access. 이전 값을 print함
```

- 포인터 x에 malloc으로 메모리 블록 할당
- 근데 그곳은 이전에 free 되었던 메모리 블록이었음
- 근데 포인터 x에 이상 값도 아직 안 넣어줌
- 그래서 printf하면 free 하기 전 넣어놨던 값이 그대로 출력됨
- free해도 data가 사라지는 게 아님

Memory Leak

- 프로그램이 free 하지 않고 계속 allocate하는 현상
- 언젠가 메모리가 부족해지고 OS도 죽어버릴 거야....

Dangling Pointer

- data를 사용하기 전에 free 해버리는 경우

- 다음과 같이 포인터 a는 원소가 3개인 리스트를 가르키고 b는 리스트의 허리를 가르키고 있음

- 근데 여기서 b를 free해버리면?? 2~3 사이 허리가 끊기면서 b는 dangling pointer가 되어버림...

Incorrect free()

- free()를 두 번 해버리는 경우

```
int *x = (int *)malloc(sizeof(int)); // allocated
free(x); // free memory
free(x); // free repeatedly
```


- 무슨 일이 일어날지 모른다...?

Other Memory API

calloc()

- void* calloc(size_t num, size_t size)
- num * size 만큼 메모리 할당
- 모든 영역을 0으로 초기화도 해줌

Memory example

realloc()

- void *realloc(void* ptr, size_t size)
- 메모리 블록의 크기 변경
- 여기서 포인터 ptr은 이미 할당된 메모리 블록이어도 됨
- size로 메모리 블록을 바꿔줌

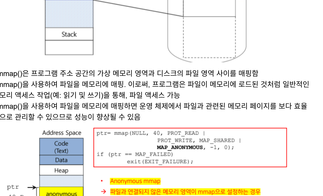
System calls

brk, sbrk

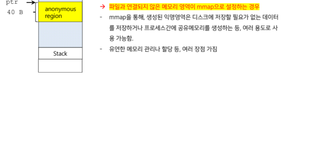
- int brk(void* addr)
- void* sbrk(intptr_t increment)
- address space에서 heap의 끝을 break라고 함
- brk와 sbrk는 system call임
- malloc은 brk를 사용함
  - brk를 사용해서 프로그램의 break를 확장시킴
  - brk는 주소를 인자로, sbrk는 increment를 인자로 받음
  - 프로그래머는 절대 brk나 sbrk를 직접 사용하면 안 됨

mmap

- void *mmap(void* ptr, size_t length, int prot, int flags, int fd, off_t offset)
- ptr가 가르키는 주소에 length만큼 메모리 영역 할당
- 만약 fd가 음수가 아니면 파일이 유효하다는 의미
- 유효한 파일이면 내용을 메모리 블록에 매핑
- 파일 시작 주소에서 offset 바이트 이후부터 시작



- mmap()은 프로그램 주소 공간의 가상 메모리 영역과 디스크의 파일 영역 사이를 매핑함
- mmap()을 사용하여 파일을 메모리에 매핑. 이로써 프로그램은 파일이 메모리에 로드된 것처럼 일반적인 메모리 액세스 작업(예: 읽기 및 쓰기)을 통해, 파일 액세스 가능
- mmap()을 사용하여 파일을 메모리에 매핑하면 운영 체제에서 파일과 관련된 메모리 페이지를 보다 효율적으로 관리할 수 있으므로 성능이 향상될 수 있음



- Anonymous mmap
- → 파일과 연결되지 않은 메모리 영역이 mmap으로 설정되는 경우
- mmap을 통해, 생성된 익명영역은 디스크에 저장할 필요가 없는 데이터를 저장하거나 프로세스간에 공유메모리를 생성하는 등, 여러 용도로 사용 가능함
- 유연한 메모리 관리나 할당 등, 여러 장점 가짐