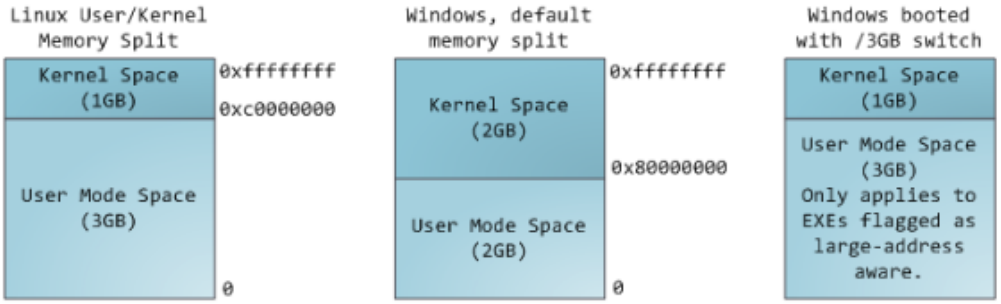
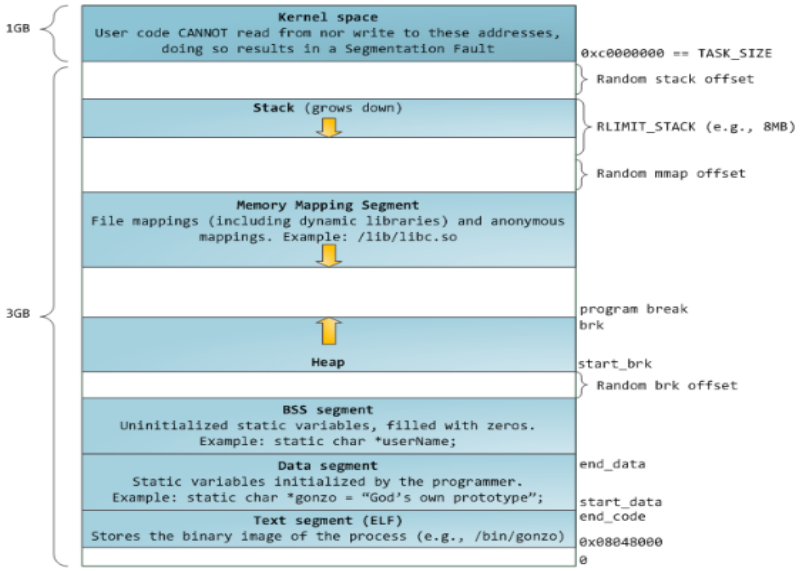
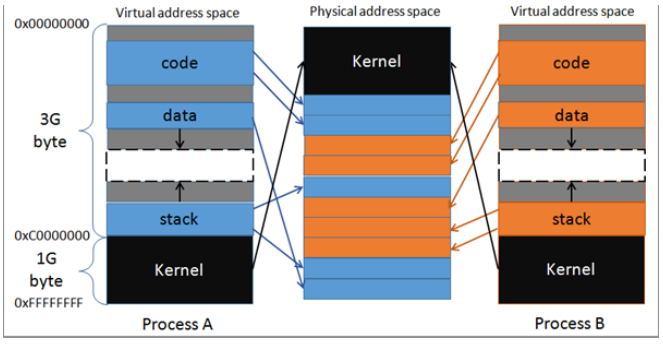
전형적인 운영체제 가상 메모리 layout



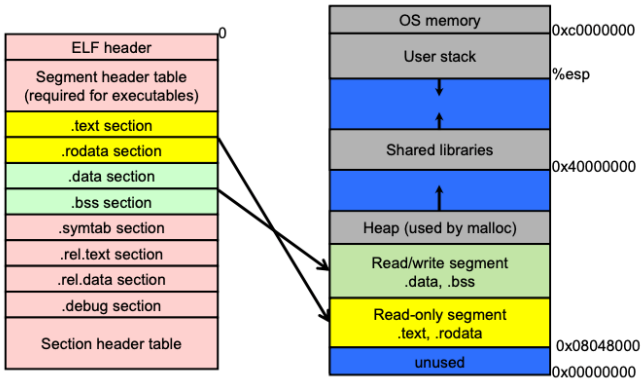
< 가상 메모리 주소 공간 layout >



< IA-32 기반 리눅스의 가상 메모리 주소 공간 >



< 프로세스와 가상 메모리 주소 공간 >



< ELF 포맷과 가상 메모리 주소 공간 >

- IA-32에서 수행되는 바이너리 실행파일의 경우 0x8048000,
- 32bit SPARC v8에서 수행되는 바이너리 실행파일의 경우 0x10000
- IA-64에서 수행되는 바이너리 실행파일의 경우 0x100000000

○ 바이너리 실행 프로그램 파일이 이미지

- Text 세그먼트
  - ✓ 코드 세그먼트
  - ✓ 프로그램 실행 가능 명령어 포함
  - ✓ Overflow로 덮어 쓰지 않게 하기 위해 힙이나 스택 아래 둠
  - ✓ 공유가 가능한 부분 있음
  - ✓ 읽기/실행 영역
- Data 세그먼트
  - ✓ 초기화된 데이터 세그먼트
  - ✓ 프로그래머가 초기화한 전역 변수와 정적 변수가 포함
  - ✓ 일반적으로 변수의 값은 런타임에 변경 될 수 있기 때문에 Data 세그먼트는 읽기 전용이 아님
  - ✓ 초기화된 읽기전용 영역(\RoData)과 초기화된 읽기/쓰기 영역으로 분류 가능
    - 초기화된 읽기/쓰기 영역에 포함되는 예

      char s[] = "hello world";

      int debug = 1;

      static int i = 10 ;
    - 초기화된 읽기전용 영역 (즉, RoData)에 포함되는 예

      const char* str = "hello world";

      문자 포인터 변수 “hello world” 라는 문자열은 초기화된 읽기전용 영역에 저장

      그러나 이 문자열을 저장하기 위한 첫 번째 문자를 위한 문자 포인터 변수 str은 초기화된 읽기/쓰기 영역에 저장 (run time 시에 수정될 가능성 있기 때문)
- BSS 세그먼트
  - ✓ 초기화되지 않은 데이터 세그먼트 (Uninitialized data segment)
  - ✓ Block Started by Symbol"이라는 예전 어셈블러 오퍼레이터에서 유래
  - ✓ BSS 세그먼트의 데이터는 프로그램이 실행되기 전에 OS 커널에 의해 0으로 초기화
  - ✓ BSS 세그먼트는 데이터 세그먼트의 끝에서 시작하여 0으로 초기화되거나 명시적 초기화가 되지 않은 모든 전역 변수와 정적 변수를 포함

✓ static int i ;
- 읽기/쓰기 가능

- Stack 세그먼트
✓ 프로그램 스택
✓ 일반적으로 OS 커널 공간 바로 아래의 상위 메모리 주소에 위치한 LIFO 구조로 표준 x86 아키텍처에서는 하위 주소로 확장
✓ 일부 다른 아키텍처에서는 반대 방향으로 성장 가능
✓ 프로그램에서 함수 호출
✓ 한 함수 호출에 대해 푸시 된 값 집합은 스택 프레임으로 명명
✓ 모든 자동 변수 (함수의 범위에 국한, 함수의 입력으로 전달 된 실제 매개 변수 포함)
✓ caller의 리턴 주소

- Heap
✓ 동적 메모리 할당이 일반적으로 발생하는 세그먼트
✓ 실행 시에만 크기를 알 수 있고 프로그램 실행 전에 컴파일러가 정적으로 결정할 수 없는 변수에 프로그래머가 요청한 메모리를 할당하는 세그먼트
✓ 힙 영역은 BSS 세그먼트의 끝에서 시작하여 더 높은 메모리 주소로 위로 성장
✓ brk 및 sbrk 시스템 호출을 사용하여 크기를 조정
✓ 영역은 모든 공유 라이브러리와 프로세스에서 동적으로 로드된 모듈에 의해 공유
✓ stdlib.h

함수	기능
void *malloc(size_t size);	size 바이트의 메모리를 힙에서 할당하여 반환
void *calloc(size_t num, size_t size);	(num * size) 바이트의 메모리를 힙에서 할당하고 포인터값을 반환
void *realloc(void *ptr, size_t size);	ptr이 가리키는 메모리를 size 바이트만큼 힙에서 재할당하여 반환
void free(void *ptr);	ptr이 가리키는 메모리를 해제
	해제 전까지 계속 존재하므로 필요없으면 이 함수에 의해 해제

```
$ size /usr/bin/cc /bin/sh
   text    data     bss     dec     hex filename
  79606    1536     916   82058    1408a /usr/bin/cc
 619234   21120   18260  658614    a0cb6 /bin/sh
```

예제 1)

```
#include <stdio.h>
int main(void)
{
    return 0;
}
```

```
[oslab@os.ssu.ac.kr]$ gcc a.c -o a-memory
[oslab@os.ssu.ac.kr]$ size a-memory
text        data         bss         dec         hex     filename
960          248           8        1216         4c0     a-memory
```

예제 2)

```
#include <stdio.h>
int global; /* Uninitialized variable stored in bss */
int main(void)
{
    return 0;
}
```

```
[oslab@os.ssu.ac.kr]$ gcc a.c -o a-memory
[oslab@os.ssu.ac.kr]$ size a-memory
text        data        bss         dec         hex     filename
 960         248          12        1220         4c4    a-memory
```

예제 3)

```
#include <stdio.h>
int global; /* Uninitialized variable stored in bss */
int main(void)
{
    static int i; /* Uninitialized static variable stored in bss */
    return 0;
}
```

```
[oslab@os.ssu.ac.kr]$ gcc a.c -o a-memory
[oslab@os.ssu.ac.kr]$ size a-memory
text        data        bss         dec         hex     filename
 960         248          16        1224         4c8     a-memory
```

예제 4)

```
#include <stdio.h>
int global; /* Uninitialized variable stored in bss*/
int main(void)
{
    static int i = 100; /* Initialized static variable stored in DS*/
    return 0;
}
```

```
[oslab@os.ssu.ac.kr]$ size a-memory
text        data        bss         dec         hex     filename
960          252          12        1224         4C8    a-memory
```

예제 5)

```
#include <stdio.h>
int global = 10; /* initialized global variable stored in DS*/
int main(void)
{
    static int i = 100; /* Initialized static variable stored in DS*/
    return 0;
}
```

```
[oslab@os.ssu.ac.kr]$ size a-memory
text        data        bss         dec         hex     filename
960          256           8        1224         4c8    a-memory
```