Sharing Revisited: Shared Objects

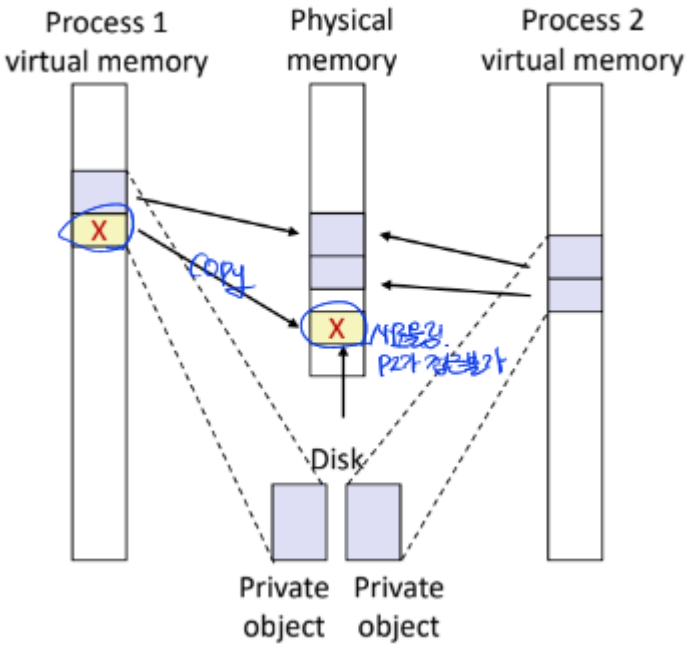
1. 프로세스 1, 2가 공유하는 자원 디스크로 존재
2. P1 또는 2가 공유자원에 접근, Disk에서 PM으로 올림
3. P1이 공유자원 변경하면 P2가 접근하는 공유자원도 변경됨 -> 각 프로세스가 한 공간을 참조하기 때문에 이러한 일 발생

공유 자원

공통 데이터는 공유하면서, 변화한 걸 다른 프로세스에게 보여주고 싶지 않다면?
-> 공유 자원에 변화를 주려고 할 때 copy해서 저장하기! (CoW)

Privacy: Copy-on-Write

1. 공유자원 존재하는 p1, p2 존재
2. 공유자원 read시 접근하는 위치 같음
3. p1이 공유자원 write 하려고 할 때 Physical Memory가 복사되며 변경된 부분은 더이상 공유자원이 아니게 된다. p1만의 것이 됨.



변화된 부분만 따로 PM에 dup되고, disk도 두개의 영역으로 나뉜다.

fork()

fork()하면 부모와 동일한 자식 프로세스 생성됨.
page table, 자원, 모두 공유하는 형태가 된다.
자식 프로세스에서 테이블이나 자원 변경하면 부모의 것은 변경되지 않는다.

CoW

- Insight
  공유자원은 좋으나 변경이 있을 때는 private copy가 필요하다.

- Idea
  page table에서 모든 변경 가능한 area를 CoW로 표시한다.

page table exists per-process!

The fork Functuin Revisited

- fork()
  - 새로운 process의 주소 할당
  - mm_struct, vm_area_struct, page-table 복사
  - page는 read-only로 설정 후 변경되면 CoW로 처리
  - vm_area_struct는 CoW로 처리한다.
    가상 메모리 영역으로, 공유자원 write 할 때만 Copy -> 효율적

mm_struct, vm_area_struct: process memory layout, vm_area_struct가 mm_struct의 멤버이다.

- 각 프로세스는 정확히 복사된 VM을 가진다
- 자식에서 read-only data에 write시 CoW로 처리한다

The execve Function Revisited

1. execve를 통해 현 프로세스에서 새로운 프로그램 로드 준비
2. vm_area_sturct, page table 할당 해제
3. 새로운 vm_area_struct, page table 할당

User-Level Memory Mapping

mmap (addr, len, prot, flag, fd, offset)

malloc이랑 비슷하게 메모리를 할당하는 방식이다.
맵핑된 영역의 시작주소를 return
주소에 null이 아닌 사용자 지정 주소 넣어도 되지만, 사용중이면 할당이 안되는 이슈가 존재하기 때문에 null로 선언해 자동으로 할당되게 하는 것이 낫다.

mmap vs malloc

mmap	**malloc**
System call	메모리 할당 인터페이스
shared lib같은 대용량	작은용량
Memory mapped region(stack, heap(x))	Heap

mmap 특징

1. Lazy loading
   VM에 만들어졌어도 접근 없으면 PM에 올리지 않음
2. Memory management
   munmap():
   할당 해제시 바로 OS에 반납

<->
free():
malloc은 pointer로만 free하기에 page가 없어지는 거지 바로 OS로 반납하지는 않음
PM은 page 단위이기에 (4kb) 4kb 이하는 free해도 바로 회수 하지 않으며, 4kb 전체가 free 되어야 반납된다.