Memory Mapping

The mmap() system call provides mapping in the virtual address space of the calling process that maps the files or devices into memory. This is of two types −

File mapping or File-backed mapping − This mapping maps the area of the process' virtual memory to the files. This means reading or writing to those areas of memory causes the file to be read or written. This is the default mapping type.

Anonymous mapping − This mapping maps the area of the process' virtual memory without backed by any file. The contents are initialized to zero. This mapping is similar to dynamic memory allocation (malloc()) and is used in some malloc() implementations for certain allocations.

The memory in one process mapping may be shared with mappings in other processes. This can be done in two ways −

- When two processes map the same region of a file, they share the same pages of physical memory.
- If a child process is created, it inherits the parent's mappings and these mappings refer to the same pages of physical memory as that of the parent. Upon any change of data in the child process, different pages would be created for the child process.

When two or more processes share the same pages, each process can see the changes of the page contents made by other processes depending on the mapping type. The mapping type can be either private or shared −

Private Mapping (MAP_PRIVATE) − Modifications to the contents of this mapping are not visible to other processes and the mapping is not carried to the underlying file.

Shared Mapping (MAP_SHARED) − Modifications to the contents of this mapping are visible to other processes and mapping is carried to the underlying file.

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

The above system call returns the starting address of the mapping on success or MAP_FAILED on error.

The virtual address addr, can be either user specified or generated by the kernel (upon passing addr as NULL). The field length indicated requires the size of mapping in bytes. The field prot indicates memory protection values such as PROT_NONE, PROT_READ, PROT_WRITE, PROT_EXEC meant for regions that may not be accessed, read, write or executed respectively. This value can be single (PROT_NONE) or can be ORd with any of the three flags (last 3). The field flags indicate mapping type either or MAP_PRIVATE or MAP_SHARED. The field 'fd' indicates the file descriptor identifying the file to be mapped and the field 'offset' implies the starting point of the file, if need to map the entire file, offset should be zero.

```
#include <sys/mman.h>

int munmap(void *addr, size_t length);
```

The above system call returns 0 on success or -1 on error.

The system call munmap, performs the unmapping of the already memory mapped region. The fields addr indicates the starting address of the mapping and the length indicates the size in bytes of the mapping to be unmapped. Usually, the mapping and unmapping would be for the entire mapped regions. If this has to be different, then it should be either shrinked or cut in two parts. If the addr doesn’t have any mappings this call would have no effect and the call returns 0 (success).