Heap Allocation

Class Notes

Heap allocation:

- Each process has virtual address space reserved for heap allocations.
- Process memory descriptor contains a record of current start address of heap (**start_brk**) and current break address (**program_brk**).
- To allocate heap program **brk** should be moved to higher address. Linux provides an API called **brk** to alter heap break point.

```
BRK(2)                                                    Linux

NAME
       brk, sbrk - change data segment size

SYNOPSIS
       #include <unistd.h>

       int brk(void *addr);

       void *sbrk(intptr_t increment);
```

Memory Mapped Segment Allocation:

- Applications can allocate dynamic memory in memory mapped segment of the virtual address space.
- mmap can be used to map any of the following buffer regions to process address space.
  1) Anonymous buffer
  2) File buffer
  3) Device buffer (address region)

```
MMAP(2)                          Linux Programmer's Manual                          MMAP(2

NAME
       mmap, munmap - map or unmap files or devices into memory

SYNOPSIS
       #include <sys/mman.h>

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

- The **first** argument takes the start address of the offset. **Second** argument takes the size of the mapping, **third** argument takes the memory access attributes, **fourth** argument tells the size and scope of mapping, **fifth** argument takes the file descriptor mapped to the buffer and the **sixth** argument takes the start offset of the file region to be mapped.

Memory mapping vs. heap allocation:

- Allocation from **heap** segment is achieved by altering heap break point.
- Releasing memory from **heap** segment would require the break point to be decremented by 'n' bytes.
- When multiple allocations are carried out through **brk** and **sbrk,** second block would begin, right at address where the first block ends. It will not be possible to free the first block without releasing entire zone.
- **mmap** calls create a physical region mapping of a specified size into address space.
- Any **memory mapped** region can be released to system.

Note:

- Physical memory mapped to heap (memory mapped region) are maintained in the process **pcb** memory description table, using an **API** called **mincore** an application can probe physical mapping status.

```
MINCORE(2)                                                Linux Programmer's Manu

NAME
       mincore - determine whether pages are resident in memory

SYNOPSIS
       #include <unistd.h>
       #include <sys/mman.h>

       int mincore(void *addr, size_t length, unsigned char *vec);
```

Malloc implementation in glibc

- GNU C library provides a set of routines to optimize, tune and observe dynamic memory allocation of C library.

  mallopt, malloc_trim, malloc_usable_size
- Glibc malloc allocates approximate 128k bytes on the first malloc call from the application.
- The region is retained until the program termination and all dynamic memory requests would be processed from this block of 128k.
- Malloc uses mmap interface to serve requests beyond a fixed threshold (configurable). The default threshold value is set to 128k (i.e. up to 128 k size) and it uses heap and mmap interface beyond 128k

Applications running on Linux can choose to disable demand paging either on a region of the address space or the entire process address space using **mlock** and **mlockall**

```
MLOCK(2)                        Linux Programmer's Manual                       MLOCK(2)

NAME
       mlock, munlock, mlockall, munlockall - lock and unlock memory

SYNOPSIS
       #include <sys/mman.h>

       int mlock(const void *addr, size_t len);
       int munlock(const void *addr, size_t len);

       int mlockall(int flags);
       int munlockall(void);

DESCRIPTION
       mlock()  and  mlockall()  respectively lock part or all of the calling process's virtual address space into RAM, preventing that memory from
       being paged to the swap area.  munlock() and munlockall() perform the converse operation, respectively unlocking part or all of the  calling
       process's  virtual  address  space,  so that pages in the specified virtual address range may once more to be swapped out if required by the
       kernel memory manager.  Memory locking and unlocking are performed in units of whole pages.
```