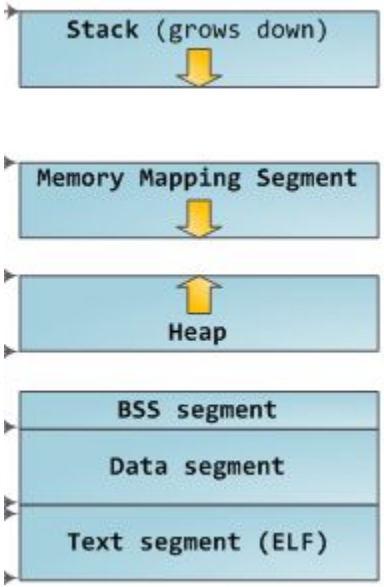
Memory Allocation

Syed Mushtaq Ahmed

Memory Layout of a Program



System calls which maniuplate heap

```
int brk(void *addr);

void *sbrk(intptr_t increment);
```

- **`brk()`** and **`sbrk()`** change the location of the program break which is essentially the heap size.
- **`brk()`** takes a direct address and **`sbrk()`** takes an offset.
- **`sbrk(0)`** gives the current top of the heap.

Naive implementation (problems?)

```c
#include <string.h>
#include <sys/types.h>
#include <unistd.h>


void *malloc(size_t size) {
  void *p = sbrk(0);
  void *request = sbrk(size);
  if (request == (void*) -1) {
    return NULL; // sbrk failed.
  } else {
    return p;
  }
}
```

Adding a metadata chunk

```
typedef struct s_block *t_block;

struct s_block {

    size_t size;
    t_block next;
    int free;

    char data[1];
}
```

A First-Fit Malloc

```
void * malloc ( size_t size ){
  _block b,last;
  size_t  s;
  s = align4 (size);
  if (base) {
    /* First find a block */
    last = base;
    b = find_block(&last, s);
    if (b) {
      /* can we split */
      if ((b->size - s) >= ( BLOCK_SIZE + 4))
      split_block (b,s);
      b->free =0;
    }
```

```
else {
      /* No fitting block , extend the heap */
      b = extend_heap (last ,s);
      if (!b)
            return (NULL);
    }

} else {
    /* first time */
    b = extend_heap (NULL ,s);
    if (!b)
    return (NULL );
    base = b;
  }
  return (b->data);
}
```

http://www.inf.udec.cl/~leo/Malloc_tutorial.pdf

Extend heap function

```
t_block extend_heap ( t_block last , size_t s){
        t_block b;

        b = sbrk (0);
        if ( sbrk ( BLOCK_SIZE + s) == (void *) -1)
                return (NULL );

        b->size = s;
        b->next = NULL;

        if (last)
                last ->next = b;

        b->free = 0;

        return (b);
}
```

Split function

```
void split_block ( t_block b, size_t s){
        t_block new;
        new = b->data + s;
        new ->size = b->size - s - BLOCK_SIZE ;
        new ->next = b->next;
        new ->free = 1;
        b->size = s;
        b->next = new;
}
```

Find block function

```
t_block find_block ( t_block *last , size_t size ){
        t_block b=base;
        while (b && !(b->free && b->size >= size )) {
                *last = b;
                b = b->next;
        }
        return (b);
}
```

Free

```
void free(void *p)
{
        t_block b;
        if ( valid_addr (p))
        {
                b = get_block (p);
                b->free = 1;
        }
}
```

```
void get_block(void *p){
        return p - BLOCK_SIZE;
}
```

```
int valid_addr(void *p) {

        if(base && p > base && p < sbrk(0) ) {
                        return 1;
         }

         return 0;
}
```

http://www.inf.udec.cl/~leo/Malloc_tutorial.pdf