CS 1550

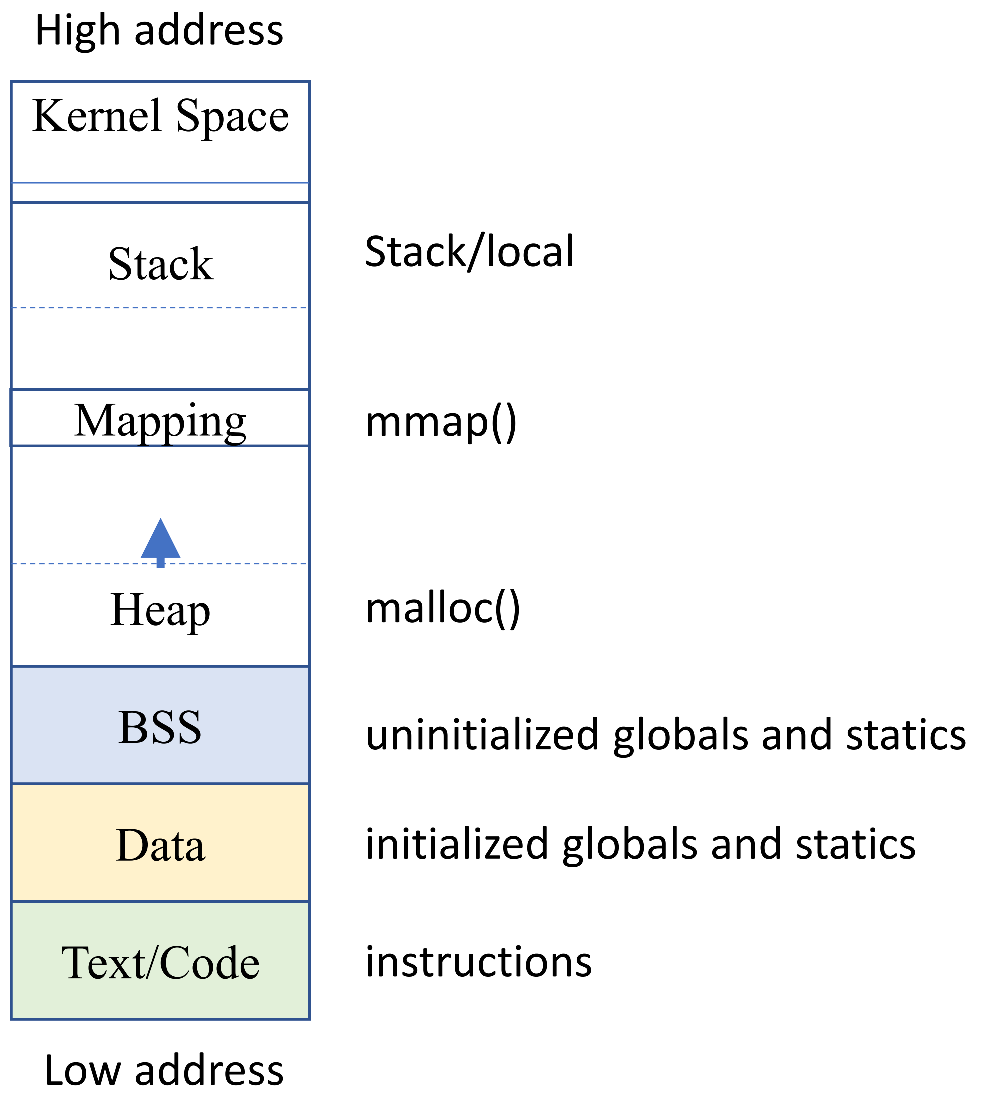
Week 11 – Lab 4

Teaching Assistant

Xiaoyu(Veronica) Liang

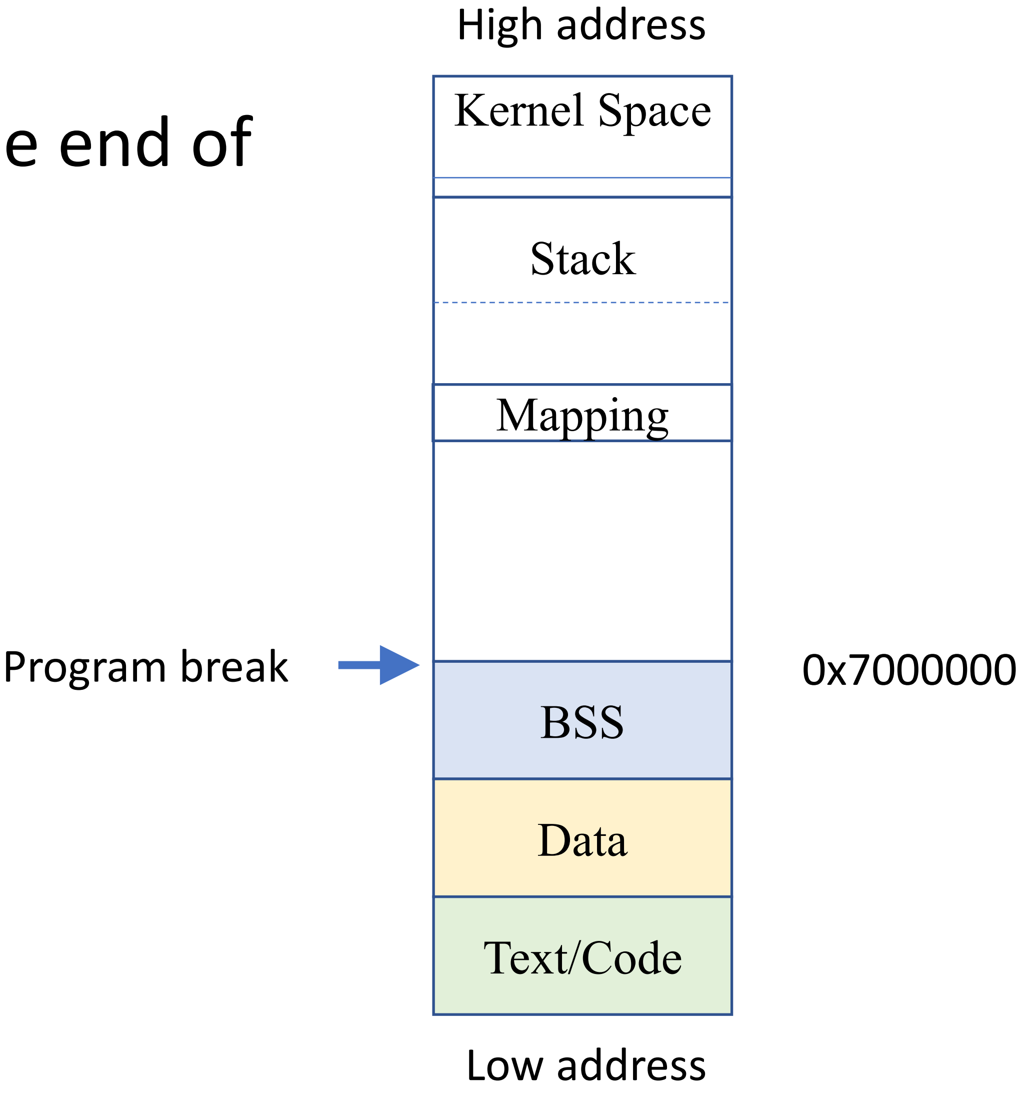
Memory layout

```
int  t = 0; // Data
int m;      // BSS
…
int main() {
        …
        int  i;             // Stack
        static  int  j;     // BSS

        // ptr: Stack
        // 4B pointed by ptr: Heap
        char * ptr = (char*)malloc(4);

         // mptr: Stack
        // 4K pointed by mptr: memory Mapping
        char * mptr = (char*)mmap(…,4096,…);
        …
}
```



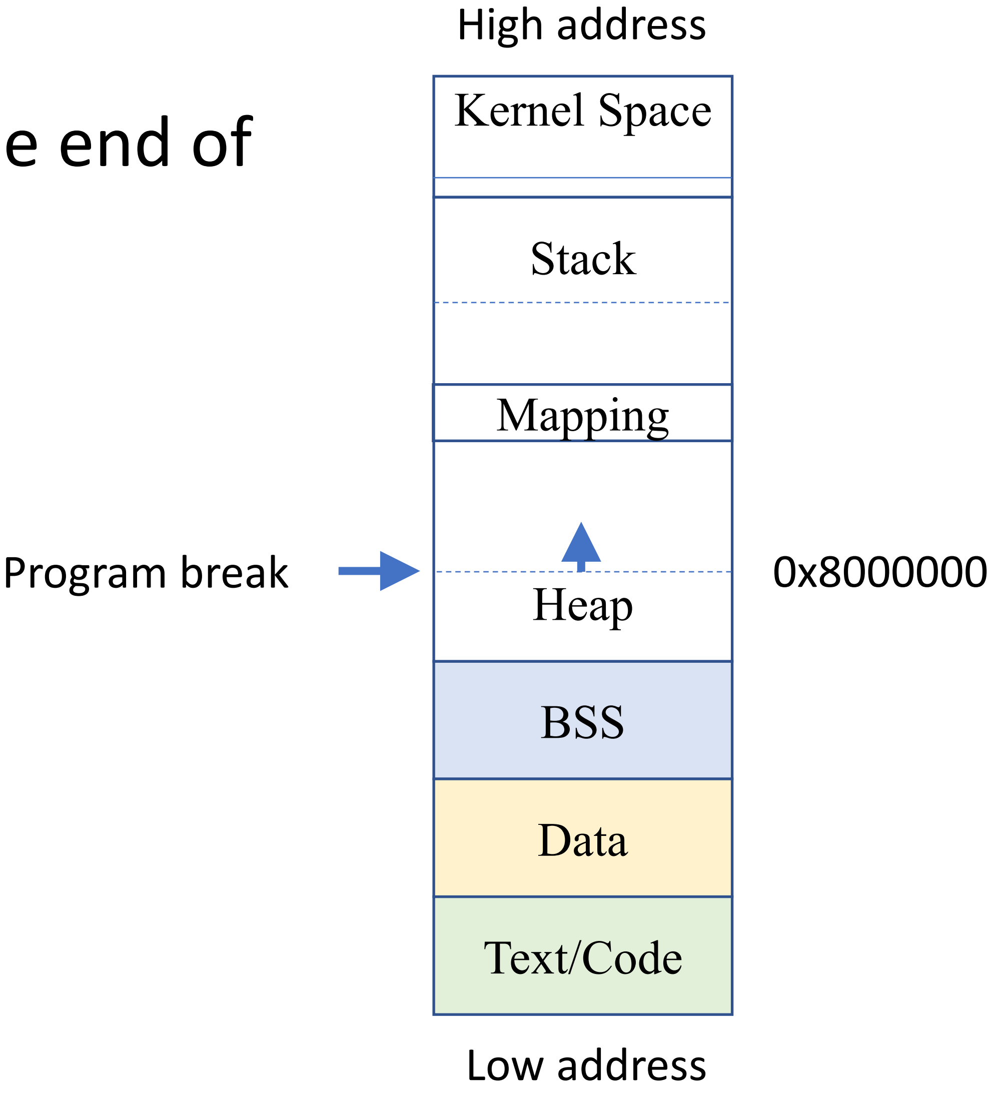
Program Break

- Program break marks the end of the uninitialized data



Program Break

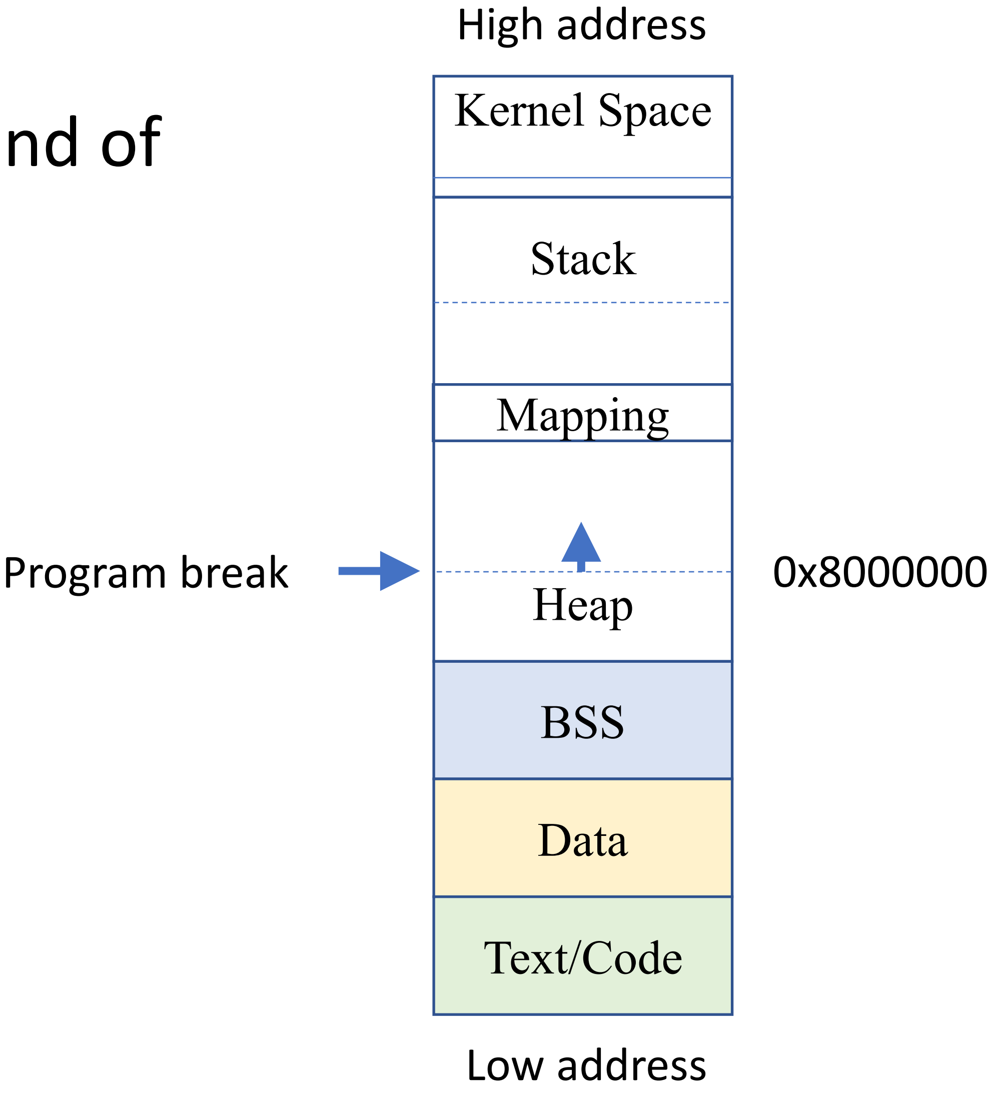
- Program break marks the end of the uninitialized data



The Syscall sbrk

- Sbrk adds a size to the end of cur_brk

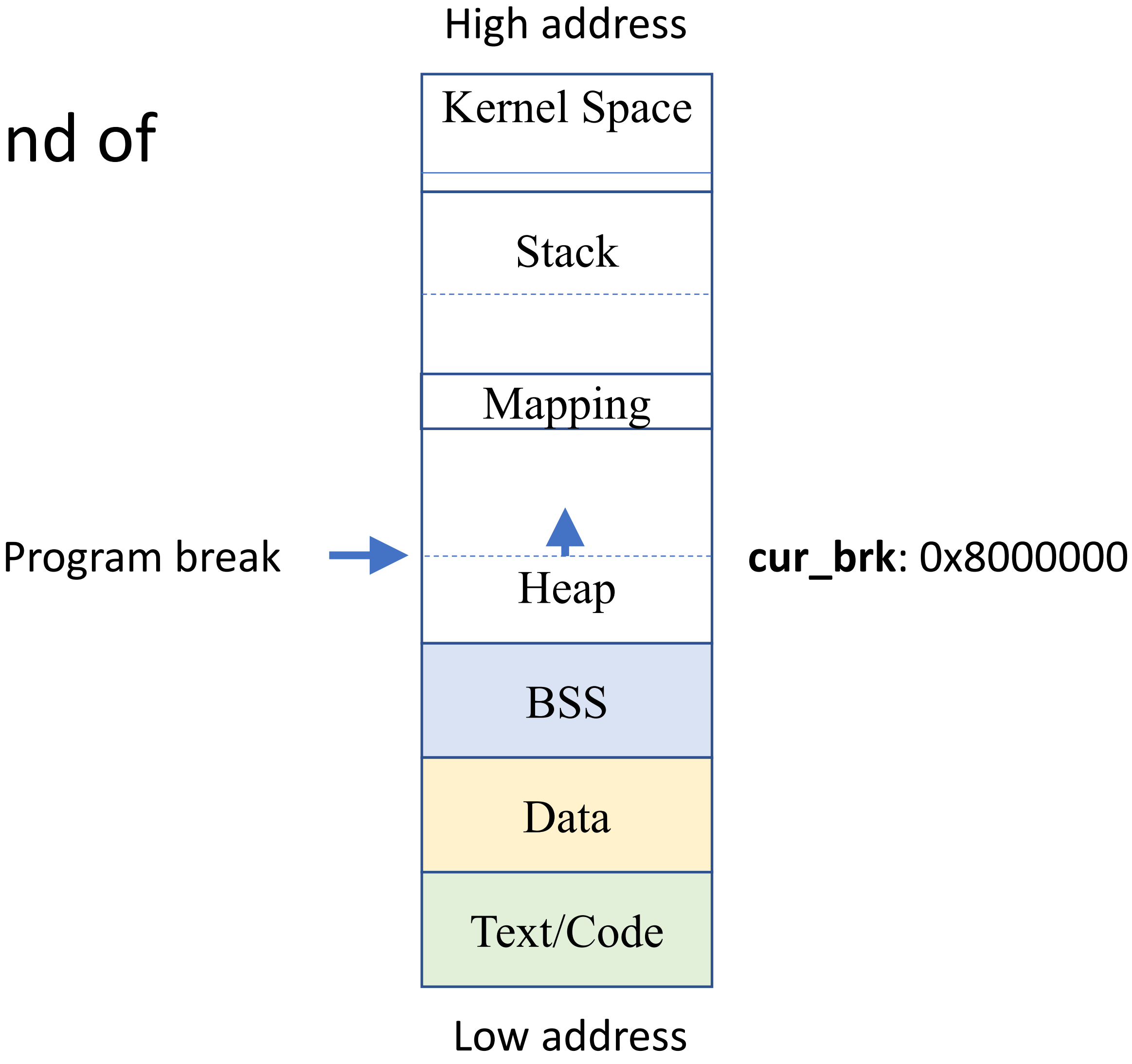
```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);

void *new_brk = sbrk(0);
```



The Syscall sbrk

- Sbrk adds a size to the end of cur_brk

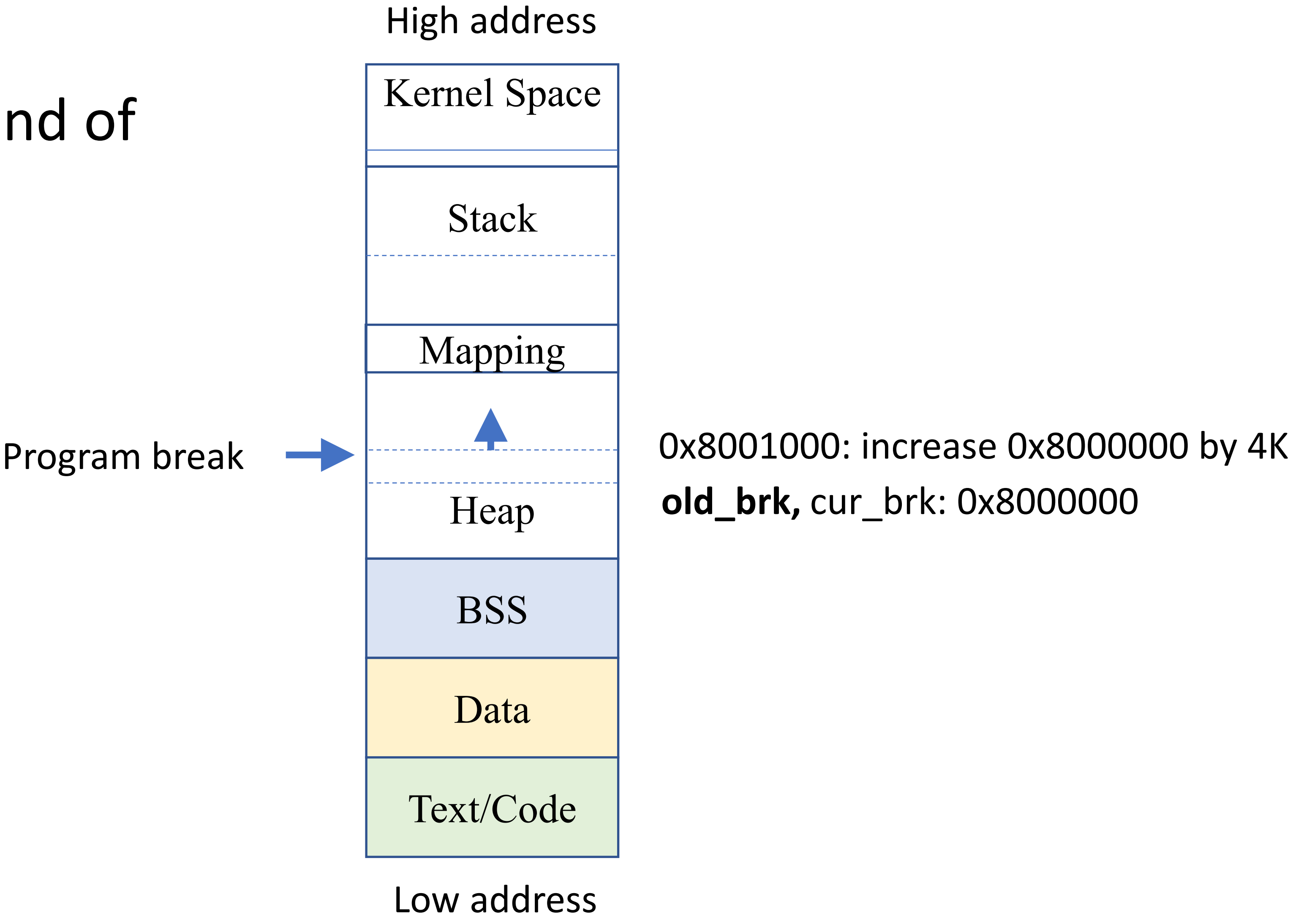
```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);

void *new_brk = sbrk(0);
```



The Syscall sbrk

- Sbrk adds a size to the end of cur_brk

```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);

void *new_brk = sbrk(0);
```



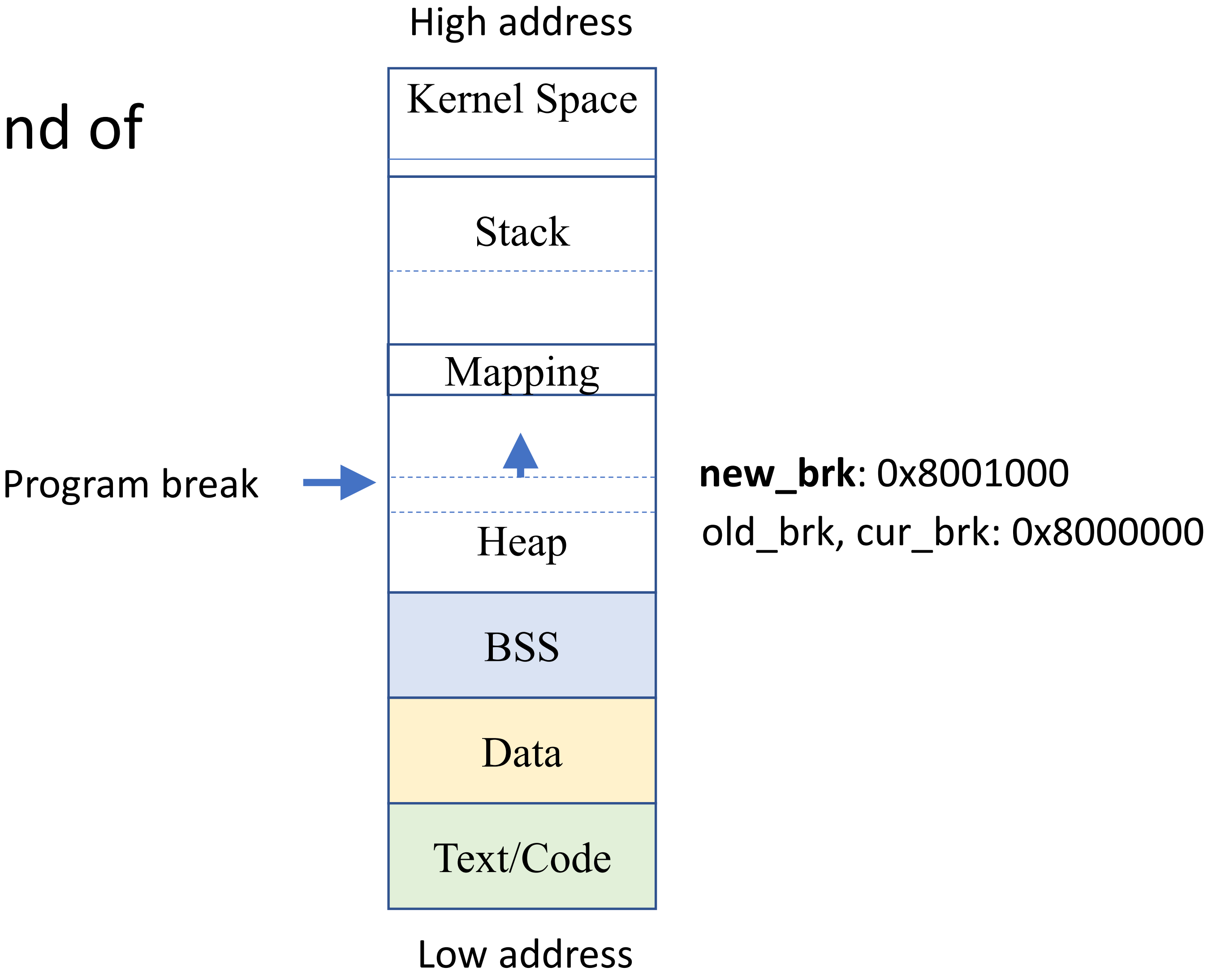
0x8001000: increase 0x8000000 by 4K

old_brk, cur_brk: 0x8000000

The Syscall sbrk

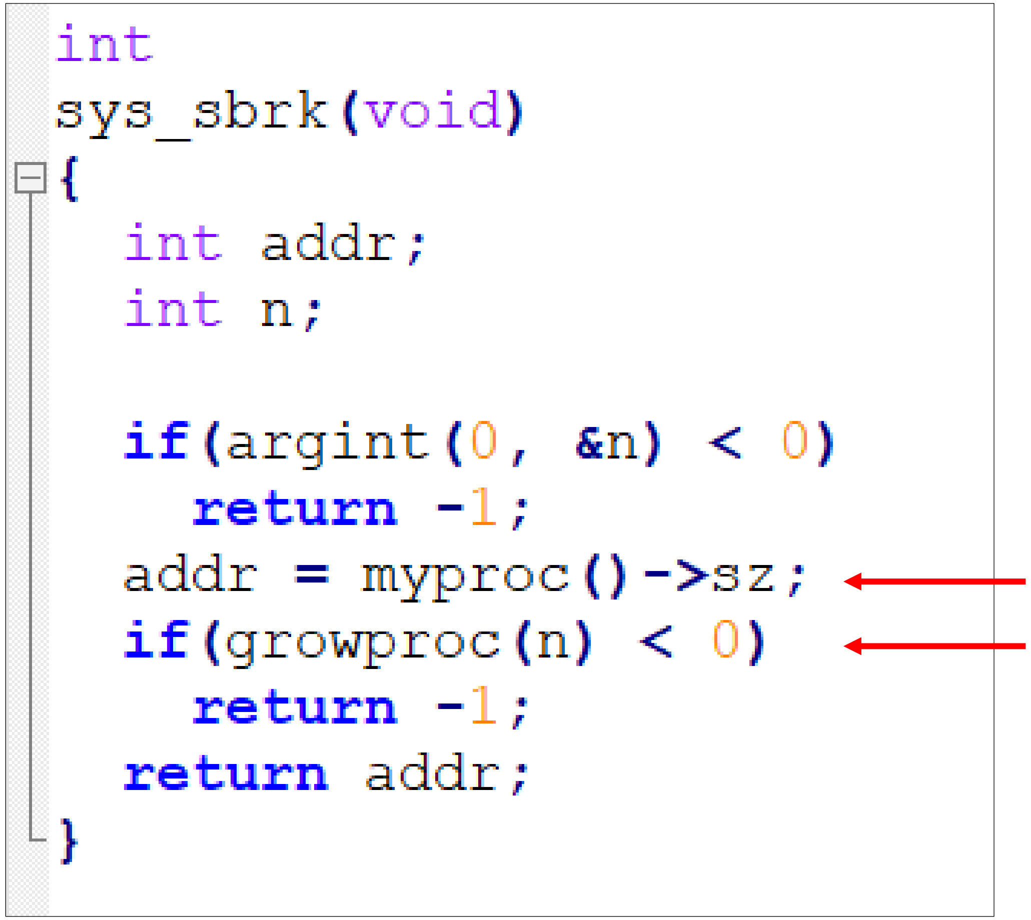
- Sbrk adds a size to the end of cur_brk

```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);

void *new_brk = sbrk(0);
```



sbrk on XV6

The ***sys_sbrk()*** in ***sysproc.c*** is the XV-6 implementation for sbrk.



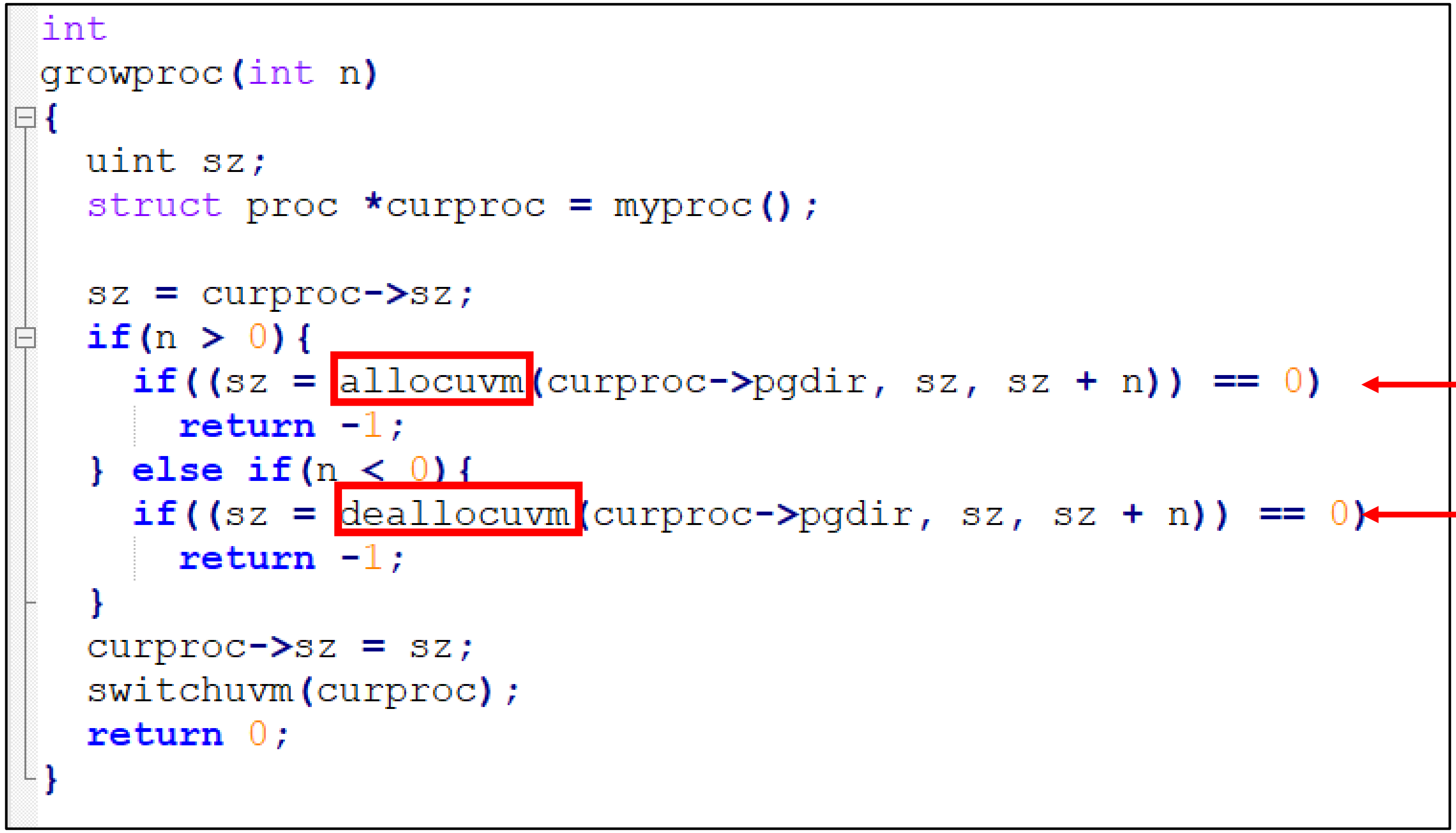
```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(growproc(n) < 0)
    return -1;
  return addr;
}
```

Get the current heapsize

Increase heapsize by n

growproc on XV6

The ***growproc()*** is in ***proc.c***:



```
int
growproc(int n)
{
  uint sz;
  struct proc *curproc = myproc();

  sz = curproc->sz;
  if(n > 0){
    if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  } else if(n < 0){
    if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  }
  curproc->sz = sz;
  switchuvm(curproc);
  return 0;
}
```

Allocates physical page, updates page table

Deallocation, updates page table, free physical page

Physical Memory Allocation

Given 4KB per page and allocating an array with size of 100 pages:

```
char * ptr = (char*) malloc (4096 * 100);
```

- This only allocates virtual memory: ptr to ptr+4096*100
- How about physical memory?

XV6: Immediately allocate all 100 physical page frames

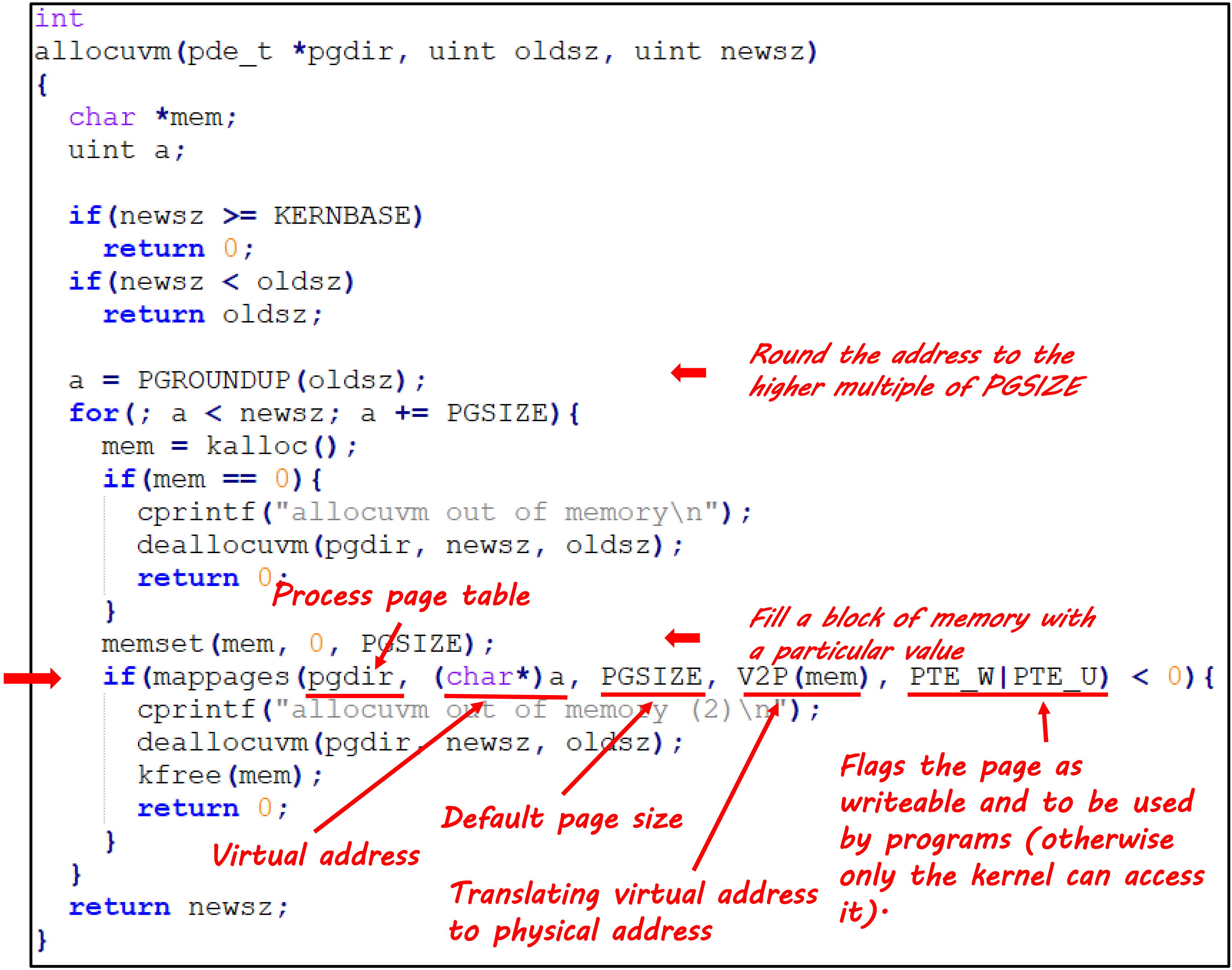
allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

mappages(pde_t *pgdir, void *va, unit size, unit pa, int perm)

Creates translations from ***va*** (virtual address) to ***pa*** (physical address) in existing page table ***pgdir.*** Returns 0 if successful, -1 if not.



```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

Round the address to the higher multiple of PGSIZE

Process page table

Fill a block of memory with a particular value

Virtual address

Default page size

Translating virtual address to physical address

Flags the page as writeable and to be used by programs (otherwise only the kernel can access it).

Physical Memory Allocation

Given 4KB per page and allocating an array with size of 100 pages:

```
char * ptr = (char*) malloc (4096 * 100);
```

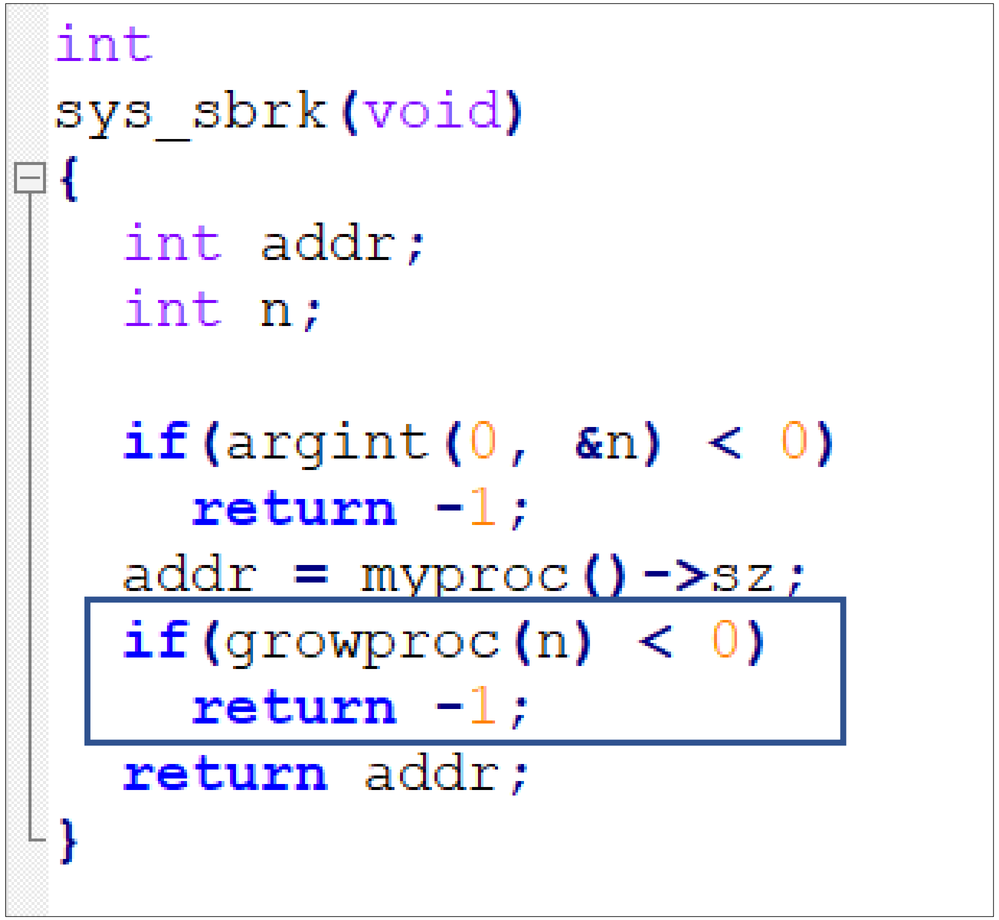
- This only allocates virtual memory: ptr to ptr+4096*100
- How about physical memory?

XV6: Immediately allocate all 100 physical page frames

Lab 4: allocate one physical page frame upon the 1st access on that page.
allocate one physical page frame when page fault happens.

Lab 4 – Part 1 Eliminate Allocation from sbrk()

- Just increment the process's size (proc->sz) by n and return the old size.
- Delete the call to growproc()



```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(growproc(n) < 0)
    return -1;
  return addr;
}
```

Comment out

Lab 4 – Part 2 Lazy Allocation

- Hint: find the virtual address that caused the page fault
  - In trap.c, find the cprintf arguments for “pid XX XX: trap XX err X on cpu X eip …”
- Hint: you can check whether a fault is a page fault by
  - By checking if ***tf->trapno*** is equal to **T_PGFLT**
- Hint: reference the logic of allocuvm() in vm.c
- Hint: use PGROUNDDOWN(va) to round the faulting virtual address down to a page
- Hint: break or return in order to avoid the cprintf and the proc->killed = 1
- Hint: call int mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
  - Delete the “static” in the declaration of mappages() in vm.c
  - Declare mappages() in trap.c

Lab 4

If all goes well, your lazy allocation code should result in “*echo hi*” working.

This is not a fully correct implementation. See the challenges in the lab description for a list of problems.

Don’t worry about these for this lab.

CS 1550 – Lab 4

- **Due**: Friday, November 15th, 2019 @11:59pm