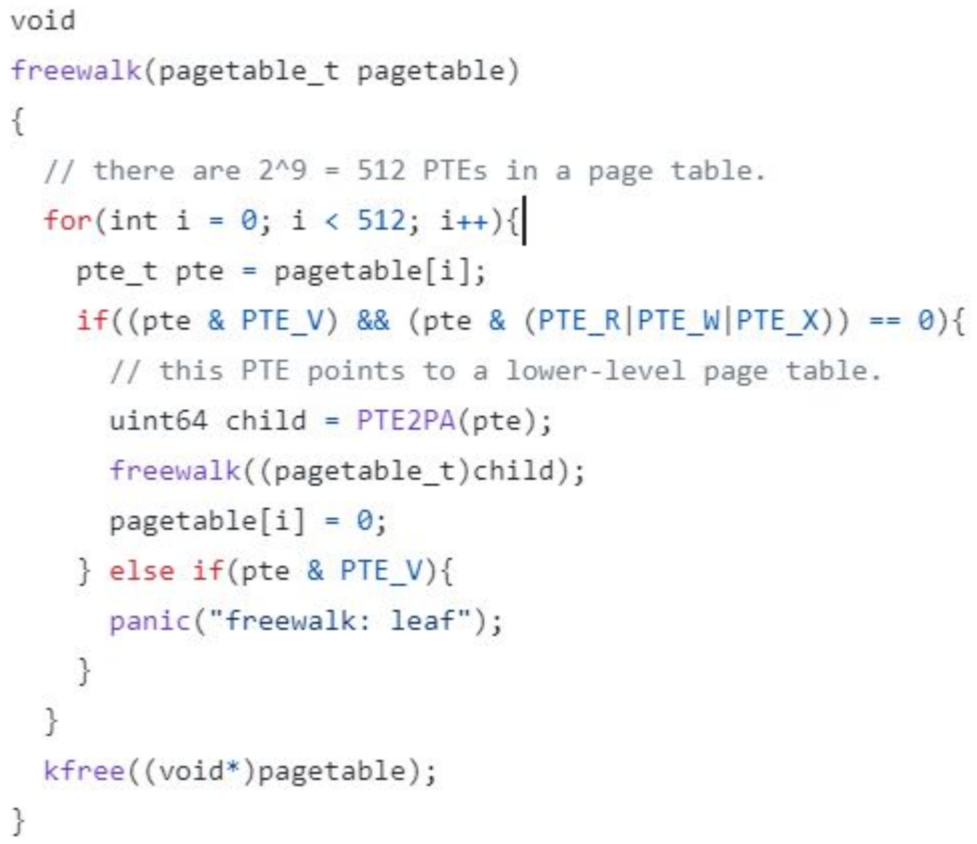
Memory Management in XV-6

Important Files and functions:

Vm.c

- Description: responsible for implementing virtual memory management.
- The vm.c file is crucial for handling page tables, page faults, and memory allocation.
- Please refer to the code to understand what functions are implemented here and how those work.
- **Take a close look into the function freewalk(pagetable_t pagetable) to understand how to get access to page table entries recursively**.



```c
void
freewalk(pagetable_t pagetable)
{
  // there are 2^9 = 512 PTEs in a page table.
  for(int i = 0; i < 512; i++){
    pte_t pte = pagetable[i];
    if((pte & PTE_V) && (pte & (PTE_R|PTE_W|PTE_X)) == 0){
      // this PTE points to a lower-level page table.
      uint64 child = PTE2PA(pte);
      freewalk((pagetable_t)child);
      pagetable[i] = 0;
    } else if(pte & PTE_V){
      panic("freewalk: leaf");
    }
  }
  kfree((void*)pagetable);
}
```

Freewalk
The function takes a single argument pagetable, which is a pointer to a page table data structure. This page table is assumed to be organized hierarchically, meaning it can have sub-level page tables nested within it.

The function begins with a for loop that iterates over each entry in the page table. The loop runs for 512 iterations since there are 2^9 = 512 Page Table Entries (PTEs) in a page table.

Inside the loop, it retrieves the value of the PTE at the current index i and stores it in a variable pte.

It then checks two conditions using bitwise operations:

a. (pte & PTE_V): This checks whether the Valid (V) bit of the PTE is set. If it's set, it indicates that the PTE is valid and points to a physical page frame or another level of the page table.

b. (pte & (PTE_R | PTE_W | PTE_X)) == 0: This checks whether the PTE has read (R), write (W), and execute (X) permission bits all unset. If they are unset, it implies that the PTE is not used for actual data storage but instead points to a lower-level page table.

If both conditions are met, it means that the PTE is a valid pointer to a lower-level page table. In this case:

a. It extracts the physical address of the child page table by calling PTE2PA(pte). The PTE2PA function p extracts the physical address from the PTE.

b. It then recursively calls freewalk on the child page table, effectively descending to the next level of the page table hierarchy.

c. Finally, it sets the current PTE to 0, indicating that it's no longer valid. This is important to avoid double-freeing memory.

If the PTE is valid (i.e., (pte & PTE_V) is true), but it doesn't meet the condition in step 4b, it implies that the PTE is a leaf node in the page table hierarchy, and it should not be a valid leaf. In this case, it triggers a panic, which is an error condition indicating an unexpected situation.

The loop continues to the next PTE until all 512 PTEs have been processed.

After processing all the PTEs in the current page table, the function frees the memory associated with the current page table using kfree.

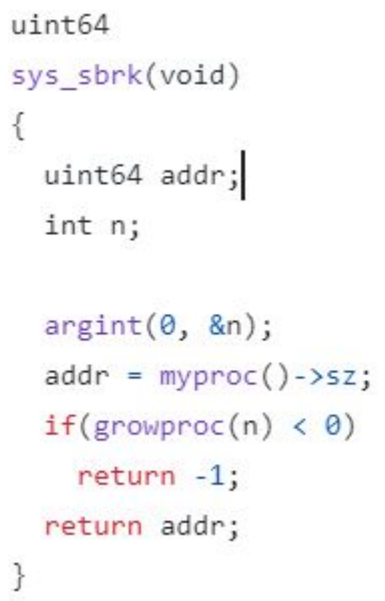
Important Files and functions:

Sbrk System call :

It is defined in sys_proc.c .

It is used to grow or shrink memory size allocated to the process.

Sys_sbrk -> growproc ()



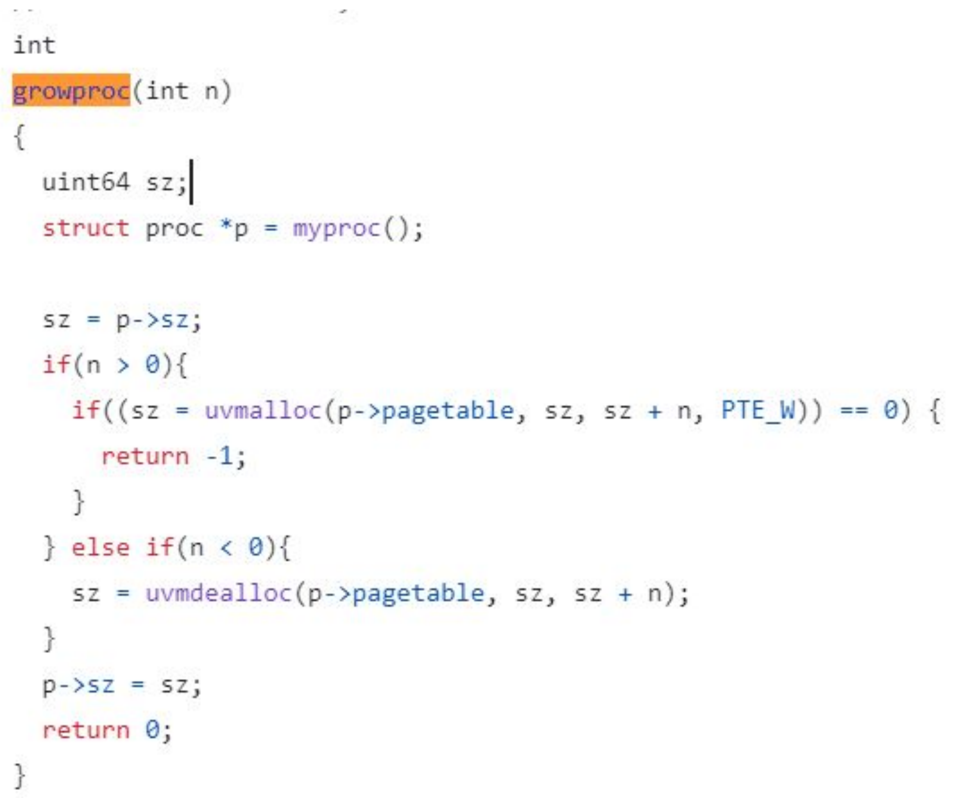
```
uint64
sys_sbrk(void)
{
  uint64 addr;
  int n;

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

Growproc (proc.c)

It invokes uvmalloc or uvdealloc.

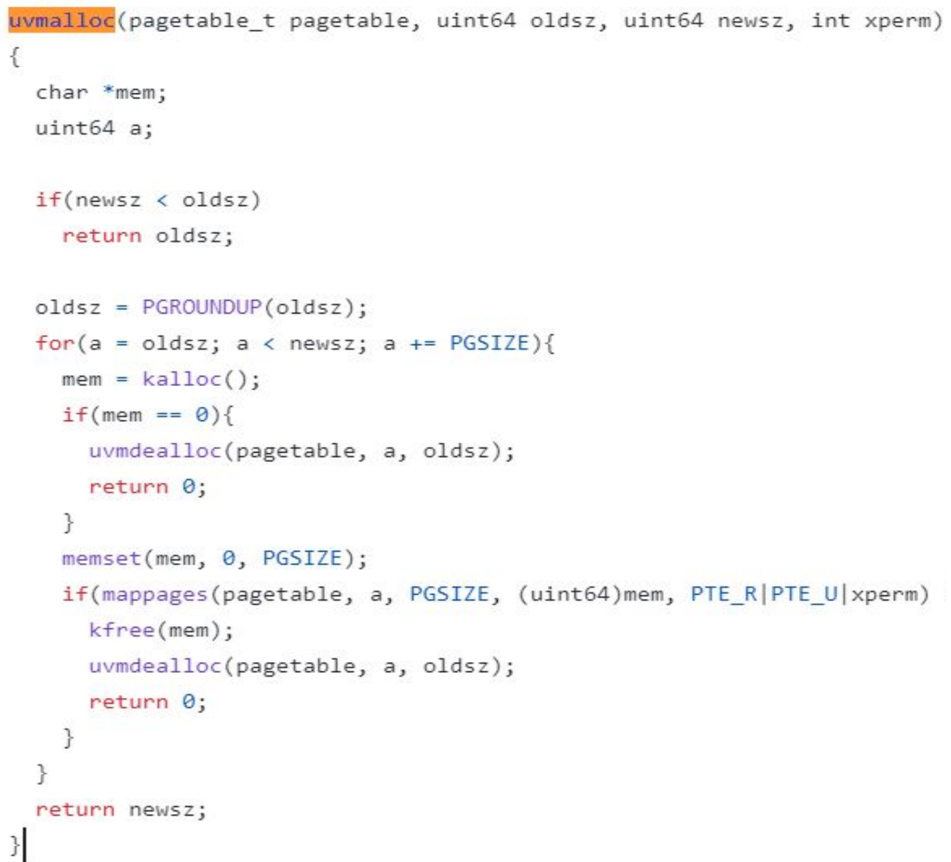
uvmalloc is responsible for mapping new pages to physical frames when the value of n is positive means we are requesting to increase the memory.

uvdealloc is responsible for freeing the memory and handle the paging when the value of n is negative. It means we are requesting to shrink the size of memory of process.



```
int
growproc(int n)
{
  uint64 sz;
  struct proc *p = myproc();

  sz = p->sz;
  if(n > 0){
    if((sz = uvmalloc(p->pagetable, sz, sz + n, PTE_W)) == 0) {
      return -1;
    }
  } else if(n < 0){
    sz = uvmdealloc(p->pagetable, sz, sz + n);
  }
  p->sz = sz;
  return 0;
}
```

uvmalloc(vm.c)



```
uvmalloc(pagetable_t pagetable, uint64 oldsz, uint64 newsz, int xperm)
{
  char *mem;
  uint64 a;

  if(newsz < oldsz)
    return oldsz;

  oldsz = PGROUNDUP(oldsz);
  for(a = oldsz; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      uvmdealloc(pagetable, a, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pagetable, a, PGSIZE, (uint64)mem, PTE_R|PTE_U|xperm)
      kfree(mem);
      uvmdealloc(pagetable, a, oldsz);
      return 0;
    }
  }
  return newsz;
}
```

In the RISC-V version of the xv6 operating system, the `uvmalloc` and `uvdealloc` functions are used for memory allocation and deallocation within the user address space (hence the "uv" prefix, which stands for "user virtual"). These functions are important components of the memory management subsystem in xv6.

1. **uvmalloc**:
   `uvmalloc` is responsible for allocating memory within the user's address space. It is called when a user program requests memory allocation, typically through system calls like `sbrk` or when initializing a new user process.

   This function takes the requested size as a parameter and allocates memory pages to fulfill that request. It manages the user's page table to map the allocated physical pages into the user's virtual address space.

2. **Uvdealloc**:

- uvdealloc is used for deallocating memory within the user's address space. It is typically called when a user program explicitly releases memory using a system call like `sbrk`.

- This function takes the virtual address to be deallocated as a parameter. It is responsible for removing the corresponding page table entries and releasing the physical pages if they are no longer needed by the user program.

- uvdealloc is an essential function for managing the user's memory resources efficiently, preventing memory leaks and ensuring that memory is released when no longer in use.

Both `uvmalloc` and `uvdealloc` work closely with the page table and memory management hardware to provide user processes with virtual memory management capabilities.

We request students to go through the functions and see the functions called in uvmalloc and uvdealloc to see the actual working and process through which pages are allocated e.g. mappages kfree etc .