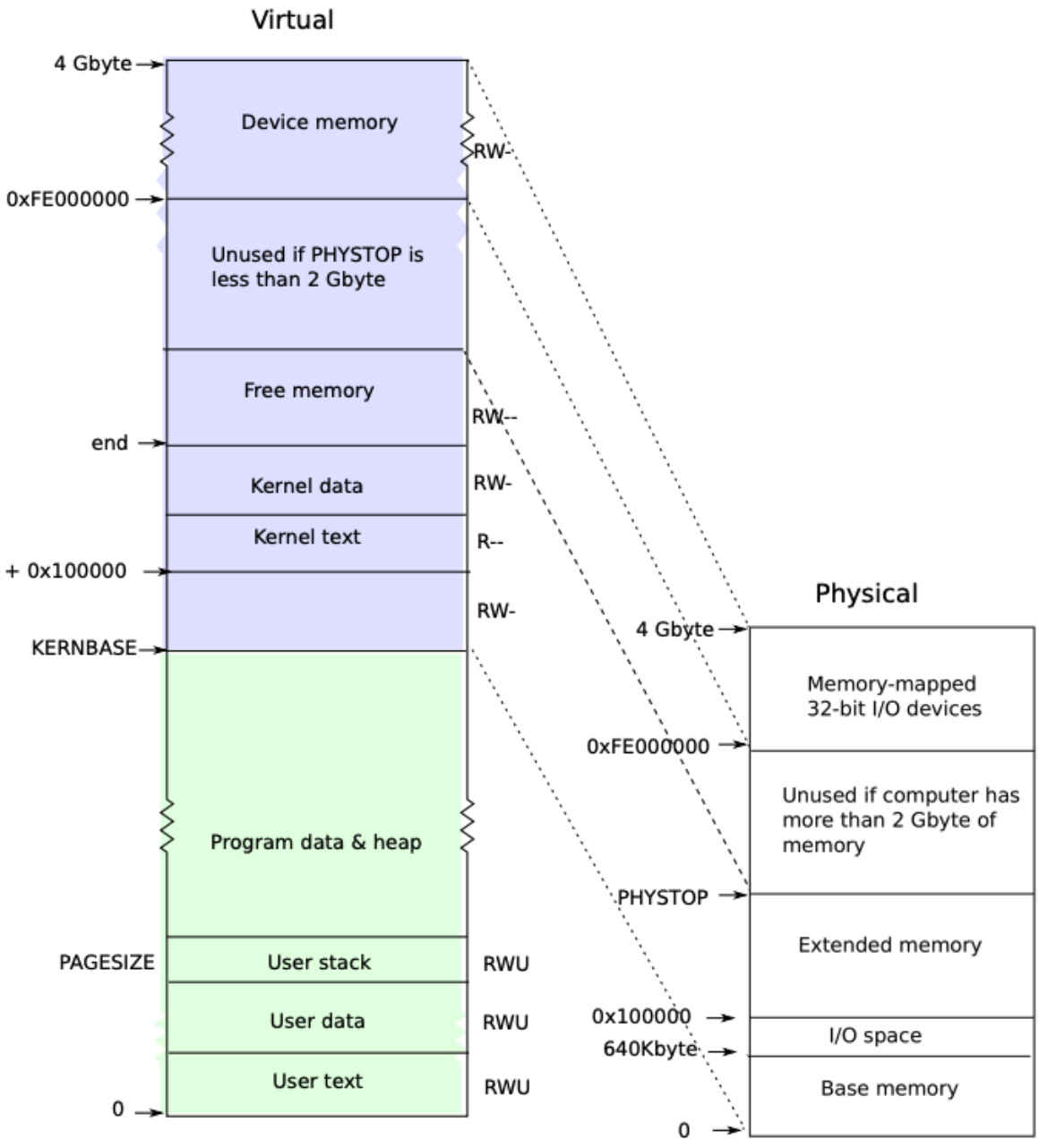
This project might seem intimidating at first, however, the key is to break it down into multiple easily solvable portions and chain them together!

Memory Layout

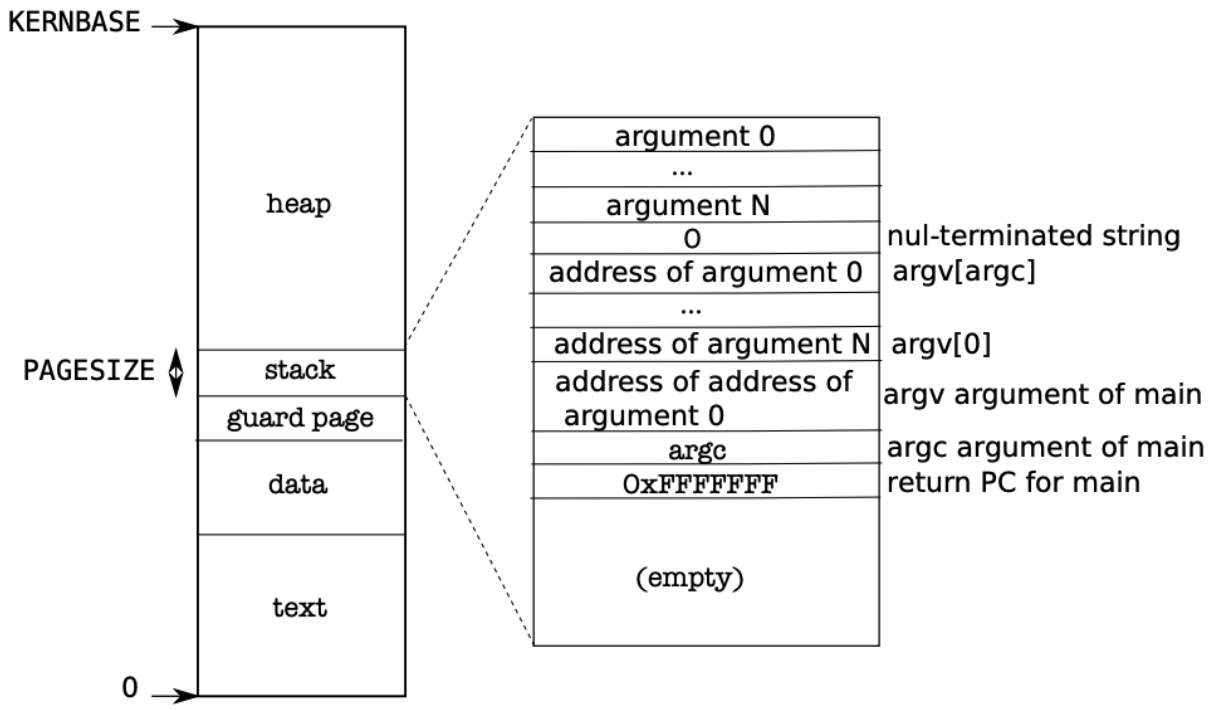


xv6 is a 32-bit OS, so its virtual address is of size 4GB (2^32 bytes). The bottom half [0, KERNBASE] is in userspace and the top half is in kernel space. For example, if you try to dereference a pointer 0xFE000000, you will get a page fault because it is not a valid userspace address and you get a permission denial. This memory layout also implies, for all the processes,

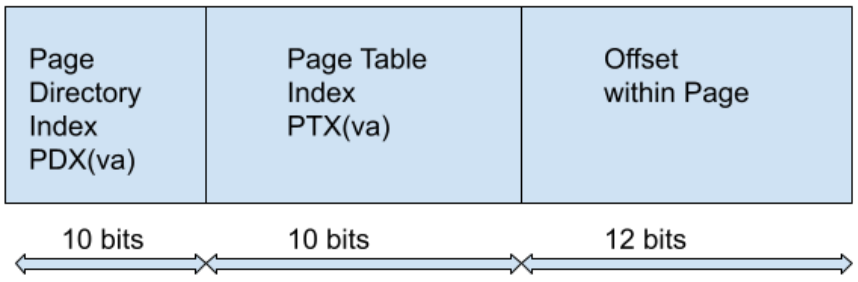
their page table entries for memory over KERNBASE is the same (because they share the same kernel).

All the physical memory (pa) is directly mapped into the kernel space kva so that the kernel could easily access every byte of physical memory.

A detailed view of user space memory layout (the green part of the picture above):



Structure of virtual memory:



See mmu.h to learn about the structure of virtual memory and how to calculate different portions of page table calculations - PDX(virtual_address), PTX(virtual_address) and offset within the

page. If you're not familiar with virtual address calculation, you can watch virtual memory lectures given on the canvas page.

Debugging

This project will require solid debugging skills. Learning debugging in the beginning is recommended. Week 9 discussion - https://github.com/akshatgit/CS537-Discussion-sp22-2/tree/main/week-9 covers how to debug user applications and https://github.com/akshatgit/CS537-Discussion-sp22-2/tree/main/week-2 covers how to debug xv6 kernel.

Clock Algorithm

All pages are in one of three states:

- encrypted: present: 0, encrypted: 1, access: 0
- hot: present: 1, encrypted: 0, access: 1: the process is actively accessing this page
- cold: present: 1, encrypted: 0, access: 0

The possible state transitions include:

- When a page is created, it should be in the encrypted state.
- When an encrypted page is accessed, it turns to the hot state. This should trigger a page fault and the page fault handles this transition.
- When the clock hand moving over a hot page, it turns to the cold state.
- When the clock hand moving over a cold page, it gets evicted and turns to the encrypted state.
- When a cold page gets accessed, it turns hot. This has already been handled by the hardware MMU, so you don't need to do anything special.

Some tips:

- It is tricky to encrypt the page when it is allocated. It might be difficult to do it in functions like allocuvm because the kernel may access/modify these pages after calling allocuvm, but we cannot directly operate on encrypted data. These two functions (which are actually callers of allocuvm) might be better places to implement this:
    - exec.c:exec: it is the implementation for the syscall exec. exec will initialize memory and load data/code. Make sure you encrypt the pages after that.
    - proc.c:growproc: it is called only for growing user address space, typically when growing the heap by syscall sbrk.

- Since all the userspace pages are potential pages to encrypt, you may find some pages always stay in the hot page states: e.g. stack. Don't feel weird if you see them in your clock queue.

Starter code

Next step is to read starter code and understand what it does/provides:

1. Exposes interfaces of syscalls - mencrypt, getpgtable, dump_rawphymem and mdecrypt. Of Course that means all sys_* versions of above syscalls have already been implemented.
2. Dump_rawphymem is already implemented in starter code and can be used to dump memory
3. Getpgtable is also implemented in starter code except changes one might have to do while implementing the clock algorithm. **You are expected to add one check ni getpgtable() function.**
4. Each syscall is already defined in project spec, so we'll not go over them here.

Trap.c

This file contains code for page faults under T_PGFLT that performs decryption of the page. We induce fage faults, trap them in code provided inside trap.c

```
case T_PGFLT:
  //Food for thought: How can one distinguish between a regular page fault and a decryption request?
  cprintf("p4Debug : Page fault !\n");
  addr = (char*)rcr2();
  if (mdecrypt(addr))
  {
      panic("p4Debug: Memory fault");
      exit();
  };
  break;
```

Vm.c

This file include several important changes:

1. One might notice that everywhere PT_E and PT_P have been used together. This is because of the way we've defined stated changes. PT_E and PT_P are stored in page table flags and represent "page encrypted" and "page present" respectively. Now, if you may see the description of the clock algorithm(given above), EITHER page is present OR page is encrypted. Thus, we're checking for both bits because we only care about presence of page and page can be in either of above two states
2. Mencrypt does 3 things
    a. Checks page table for user space to kernel space address translation as form of error checking.
    b. Iterate on pages and do encryption.

    c. Once the page is encrypted, we'd want to perform a state switch from PTE_P to PTE_E.
3. Memdecrypt : Opposite of Mencrypt?
4. Getpgtable is simply getting all entries by calling walkpgdir. For each page table entry, it is filling all entries of pt_entry struct. Assuming, PTE_A denotes the access bit of the page table entry, we'd want to fill that bit as well while implementing the clock algorithm. Secondly, you'd want to see whether pages are in the clock's queue or not.
5. Can this file be the place where you can write a clock algorithm?

Clock Algo

You need to think about what steps the clock algorithm could take and which specific parts of exec and fork need to be modified for P4. Here are some hints:

Exec:
You need to create and clear out queue inside this function

Fork
You need to create and clear out the queue, then deep copy over parent's queue to this process here.

Copyuvm
This will perform a deep copy of the parent's queue while taking care of PT_P and PT_E. Think about what needs to be done for each bit.

Last question is where to run the clock algorithm or rather which function should invoke clock's movement and evict pages. Do I need to run it in both of the following places? Or just one is enough? Think!

1. mencrypt?
2. memdecrypt?

Moreover, since you're evicting pages, you may want to have some utility function which "evicts" a page for you. Think about whether you'd be evicting encrypted pages or decrypted pages?