Assignment Report: Assignment 3

Student ID:121090406

In this report, when citing a reference, such as the xv6-book, feel free to use footnotes[1].

Your report should follow the template with the following section structure.

No page limitation

1 Introduction [3']

In the introduction section, you are required to include the following:

1. Provide a simple overview of how the system works when facing an mmap request.

   Request Initiation: The process initiates an mmap request by calling the mmap function, specifying the file descriptor of the file to map, the desired address range in its virtual memory (optional), the length of the mapping, desired protection levels (like read, write), and flags determining the nature of the map (shared or private).

   Virtual Address Space Allocation: The operating system allocates a region in the process's virtual address space. This allocation is typically done lazily, meaning the actual physical memory or file data is not loaded yet.

   Page Table Update: The operating system updates the process's page table to reflect the mapping. Each page in the requested range is marked as mapped but not present. The page table entries will point to the corresponding offsets in the file.

   Lazy Loading on Access: When the process accesses a part of the mapped area, it triggers a page fault (since the data isn't actually loaded into memory yet). The operating system then loads the required part of the file into physical memory. This is known as demand paging.

   Data Access: Once the relevant data is loaded into memory, the process can access it just like regular memory. Read and write operations to the mapped area are translated into file I/O operations.

   Unmapping: When the process no longer needs the mapping, it calls munmap. The operating system then invalidates the memory region, updates the page tables, and, depending on the type of mapping and flags, may write back changes to the file.

2. Briefly introduce what you have accomplished in this assignment.

   Implement struct vma according to parameters of mmap, and implement function sys_mmap, sys_munmap, fage fault handling and improve fork and exit to be compatible with mmap.

3. Describe any difficulties you encountered during this assignment. (If none, simply state that your progress was smooth.)

   1. I do not know what paremeters struct vma should have. So I check the function void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset), so I figure out what the parameters mean and add them to struct vma.

   2. I do not know how to read data from file. I search the problem in google, I get readi which can help me out. But I get an error "undefined", and I then read assignment description, I find mapfile can solve the error. Also, the function can print "off", which makes the result the same as shown in assignment description. **Assignment description is very very import!!**

[1] Reference: https://pdos.csail.mit.edu/6.828/2005/readings/pdp11-40.pdf

2 Implementation [5']

In this section, you should present your design. For each **TODO** task, explain your implementation in a separate subsection.

It is suggested to include progress flowcharts (at most one for each subsection should suffice), like Figure 6, with text explanations.

You do not need to provide figures for **mmap** and **VMA,** as **VMA** is just a structure definition, and we have already provided the **mmap** flowchart in the instructions.

We recommend providing a succinct flowchart with a clear description if you have concerns about plagiarism detection. This flowchart should be concise and effectively illustrate your implementation without the need to copy your code directly into the report.

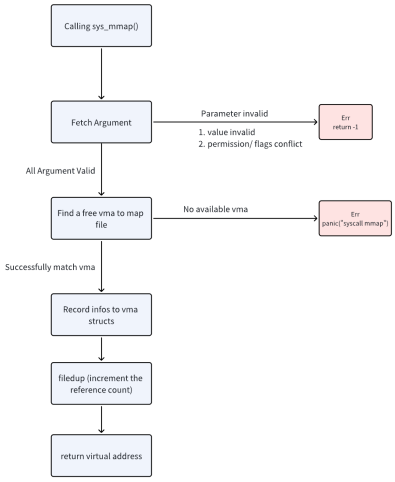

Figure 1: Do NOT use this figure directly.

2.1 VMA

struct vma {

uint64 addr;// starting address of the vma

int len;// length of the vma, it should be a multiple of page size.

int prot;// protection level

int flags;// share or private or so on

int fd;// file descriptor

int offset;// in this assignment, it is 0.

struct file *f;// file mapped to the vma

};

I implement this struct according to

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

addr is the starting address of the vma.

len is the length of the vma, and it should be a multiple of page size.

prot is the protection level of the vma, such as the right to read and the right to write.

flags controls whether some processes can share the vma.

offset is 0, in this assignment. the offset is based on addr.

f is the file mapped to the vma.

2.2 mmap

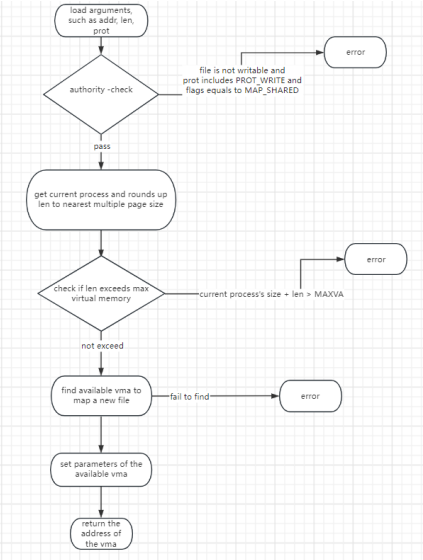

Figure 2: flowchart of sys_mmap.

First, check whether there are enough authority to execute mmap, eg. flags, prot and so on.

Second, get current process and check whether there is enough virtual memory(assure no to excees MAXVA).

Third, find available vma to map the file.

Fourth, set parameters and return the address of vma.

2.3 PageFault

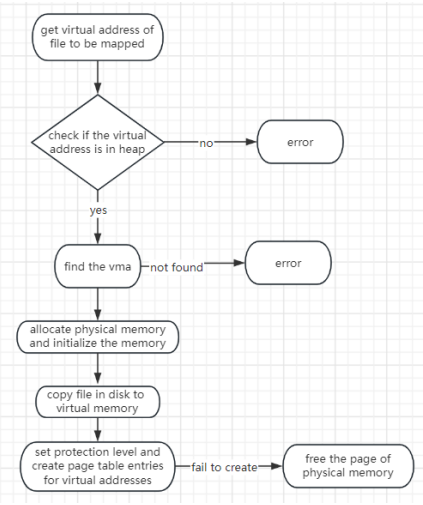

Figure 3: flowchart of PageFault.

First, get virtual memory of the file.

Second, check virtual address is in heap.

Third, find the vma and allocate physical memory.

Fourth, copy the file to virtual memory.

Fifth, set protection level and create page table entries.

2.4 munmap

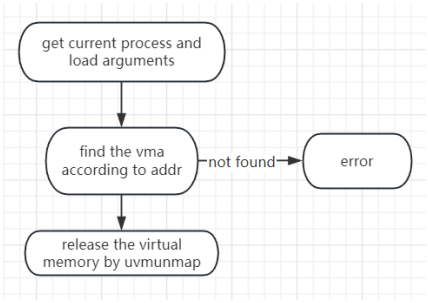

Figure 4: flowchart of munmap.

First, get current process and load arguments.

Second, find the vma according to addr.

Third, release the virtual memory by uvmunmap.

2.5 Bonus

```
//TODO
for(int i=0; i < VMASIZE; i++)
{
  memmove(&np->vma[i], &p->vma[i], sizeof(p->vma[i]));
  if(p->vma[i].f) filedup(p->vma[i].f);
}
//TODO end
```

Figure 5: code of fork.

Copy every vma from p (current process) to np (new process) in function proc.c/fork

```
for(int i=0; i<VMASIZE; i++)
{
  uvmunmap(p->pagetable, p->vma[i].addr, p->vma[i].len/PGSIZE, 1);
}
```

Figure 6: code of exit.

Unmap the virtual memory for every vma.

3 Test [2']

Briefly discuss which part of your implementation helped you pass each test. (Several sentences for each subsection should suffice.)

3.1 mmap f

This test checks if the code is able to run and load the correct content into memory. It is important to configure the vma and use mappages correctly.

3.2 mmap private

The key point is to allow writing to the mapped memory. It is important to have the correct permission set for mappages function.

3.3 mmap read-only

In this test, mmap should fail if it is set to be read/write and file is read-only. It is important to check the conditions in sys_mmap before doing anything else.

3.4 mmap read/write

In this test, the mapping should work for read/write permission for read/write opened file and should still work even after the file is closed. It is important to set the conditions correctly and use the correct function.

3.5 mmap dirty

In this test, unmapping should be able to write back to the file.

3.6 mmap two files

In this test, mmap should be able to map two files into the memory at the same time. Important to find correct vma for each mmap calls.