Project 3b: Virtual Memory

Preliminaries

> Fill in your name and email address.

Ivory E. Si 2000012957@stu.pku.edu.cn

> If you have any preliminary comments on your submission, notes for the TAs, please give them here.

> Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.

Stack Growth

ALGORITHMS

> A1: Explain your heuristic for deciding whether a page fault for an invalid virtual address should cause the stack to be extended into the page that faulted.

Update stack pointer in system call or page fault validation check.

Check the difference between new stack pointer and the old position. It may be a really large array. The limitation is set to STACK_MAX_GAP (0x1000000).

And check whether the fault virtual address is above the stack pointer or less than stack pointer in a rational range(say STACK_ESP 0x200).

Memory Mapped Files

DATA STRUCTURES

> B1: Copy here the declaration of each new or changed struct or struct member, global or static variable, typedef, or enumeration. Identify the purpose of each in 25 words or less.

```
typedef struct {
    mapid_t mapid; /* file descriptor for process */
    struct file *f; /* file pointer in file system */
    struct list_elem elem; /* element in file_list per process */
    void *start, *end; /* virtual address range */
}mmap_descriptor;
```

```
struct thread
{
    ...
    struct list mmap_list; /* to maintain mmap descriptor */
    ...
};
```

ALGORITHMS

> B2: Describe how memory mapped files integrate into your virtual
> memory subsystem. Explain how the page fault and eviction
> processes differ between swap pages and other pages.

31-9	8,7	6	5	4	3	2	1	0
SECTOR_NUMBER/MAPID	RESERVE	0	PAGE_FILE	PAGE_SWAP	PAGE_LOAD	PTE_U	PTE_W	PTE_P
depend	any	0	0/1	0/1	1	keep	keep	0

We change the page table entry for virtual address in the mmap range. We set **PAGE_FILE = 1** and **MAPID** for each thread here.

When we load page from file, the use mmap_descriptor to find the offset in the file of this page.

When we write the page back to disk, we check the PTE_D bit. If we don't write anything in memory, we do not need to change the content of the file.

> B3: Explain how you determine whether a new file mapping overlaps
> any existing segment.

When we open a new file in mmap system call, we traverse all the mmap_descriptor in the thread's mmap_list to check whether there is a overlap between new memory range and any memory segment that already exists.

RATIONALE

> B4: Mappings created with "mmap" have similar semantics to those of
> data demand-paged from executables, except that "mmap" mappings are
> written back to their original files, not to swap. This implies
> that much of their implementation can be shared. Explain why your
> implementation either does or does not share much of the code for
> the two situations.

The implementation of demand-paged from executables are not shared with mmap. The mmap mechanism is implemented in a new way.

Because the in memory-mapping file, the content of file needs to be changed after swapping page out of memory. But in execution , the original executable file should keep the same. It is better to treat executable files specially to keep system consistency.