CS450 Fall 2019
Programming Assignment 3
Jingwen Pickens
Anthony Pollizi

Design of Syscalls

Design of getsharedpage()

The syscall getsharedpage() is designed to take two arguments, a key and a number of pages the user requested. sys_getsharedpage() then calls sharedmempage() defined in vm.c. On success, getsharedpage() should return a virtual address to a physical shared page. We set the maximum value for number of requested pages. Value of the key is limited from 0 to 20. In this implementation of getsharedpage(), the first process that calls an unused key decide the amount of the physical shared page being allocated and subsequent calls to the same key will not allocate any new physical shared pages.

In vm.c, we defined four global variables to keep track of the key references, used keys, physical addresses associated with a page, and page counts, across multiple processes. Those variables will be initialized by sharedmeminit() which is also defined in vm.c. To do so we called sharedmeminit() at the end of xv6 main(). In proc.h, we defined three process-specific variables in the proc structure: an array of used keys, a 2D array storing virtual addresses associated with the keys, and an integer that stores the top of the next available virtual address space.

In sharedmempage(int key, int numPages, struct proc* proc), we first check if the argument key and numPage is valid. We then check whether it is the first time that the calling process is calling getsharedpage() by checking the used key array stored in its proc structure. If it is the first time this process calls getsharedpage(), the top of virtual address space is set to KERNBASE. There are two scenarios in which getsharedpage() will be called by a process: (1) process calls getsharedpage() with a key that is not being used by other processes; (2) process calls getsharedpage() with a key that is being used by other processes or by the calling process.

1) **Process calls getsharedpage() with a key that is not being used by other processes**

We allocate physical shared page by calling kalloc() and set physical page contents to zero by calling memset(). The physical address returned by kalloc() are stored in the global variable. We then translate the physical address to virtual address by calling P2V() and store the virtual address in the per-process 2D array. After that we store that virtual address as the top of the virtual address space in this process. After that, we map the virtual address to the physical address by calling mappages() to create the corresponding PTE. We repeat this procedure for every page requested. Finally, we update the reference count of the key and return the virtual address of the bottom of the last allocated shared page.

2) **Process calls getsharedpage() with a key that is being used**

   To reiterate, subsequent calls with the same key will not allocate new physical shared pages and calling process will have access to only the existing physical shared pages associated with that key.

   a) **If key is being used by other processes but not the calling process**

      We first check if numPage requested by calling process is greater than the number of pages associated with the key. If numPage is greater, we return the virtual address of the last physical shared page associated with the key. If numPage is less than or equal to the number of shared pages associated with key, we simply get the physical address associated with the key stored in

the global variable, translate that into virtual address with P2V and map that to the physical address by calling mappages() for each requested page. Finally, we increase the reference count of the key by 1 and return the virtual address of the bottom of the last shared page.

b) If key is being used by other processes and the calling process

In this case, we do not need the check the numPage being requested because without freeing the shared pages, any physical shared pages requested previously by the calling process will still be accessible in the calling process. Therefore, we simply get the last virtual address stored in the virtual address array in calling process's proc structure and return it.

getsharedpage() manual

NAME

getsharedpage - allocate a shared page

SYNOPSIS

```
#include "types.h"
#include "users.h"

Int getsharedpage(int key, int numPages);
```

DESCRIPTION

getsharedpage() returns the virtual address to the physical shared page associated with the value of the argument key. The number of physical shared pages will be allocated by this function is specified with the value of the argument numPages. The maximum value of numPage is 20. First call to getsharedpage() by a process using a key that is being used by other processes and a numPage greater than the number of existing physical shared pages associated with that key will only return the virtual address of the bottom of the last allocated physical shared page. A process should only call getsharedpage() once, calling it for the second time to request more pages will get same virtual address that was returned when the function is called the first time.

RETURN VALUE

On success, a valid virtual address that points to the bottom of the physical shared page is returned. On error, -1 is returned.

Design of freesharedpage()

The syscall getsharedpage() is designed to take the key as its argument. sys_freesharedpage() calls freesharedpage() which is defined in vm.c.

The function freesharedpage() first check if the calling process is using the key. If so, we decrease the reference count of that key by 1. Then for each shared page, we get the PTE corresponding to the stored virtual address associated with the key from the virtual address array in calling process's proc structure. After that, we check if the reference count is 0 at this point. If not, we simply set the PTE to 0. Otherwise, reference count of 0 implies that no process is using the key and we need to deallocate the physical shared pages corresponding to the key. To deallocate all physical shared pages associated with the key, we loop through all physical address associated with the key, which are stored in a global array, and pass them into kfree one at a time to deallocate those pages. We also added a checkpoint in the exit() syscall to check if a process has called freesharedpage() before existing, if it hasn't done so, freesharedpage() will be called in exit() for that process.

freesharedpage() manual

NAME

freesharedpage - allocate a shared page

SYNOPSIS

#include "types.h"
#include "users.h"

Int freesharedpage(int key);

DESCRIPTION

freesharedpage() remove the calling process from accessing the physical shared pages associated with the value of the argument key. When the number physical shared pages associated with that key is zero, those pages will be deallocated.

RETURN VALUE

freesharedpage() returns no value.

ERROR

If a process calls freesharedpage() with a key that has not been used when calling getsharedpage(), an error message "Calling process not associated with that key" will be printed.