OS'22: Assignment 3

Contents

NOTE: MAKE SURE that you followed the instructions carefully. Good Luck isA ☺

Important Instructions

Please read instructions carefully, any mistake or error may result in assignment rejection from the automated grading system:

1. Make sure to start solving assignment3 using the **assignment3.startup** code from:
   Make a copy of your previous code, then copy and replace the **five files** from the above link into your "**/kern**".

2. During your solution, make sure that you *DON'T CHANGE* any other file rather than command_prompt.c/.h

3. Your code *MUST* be written inside the given function for each question (as specified below).
   ANY violation of this rule, or change in function names, or deletion of template functions may result in assignment rejection by the automated grading system.

4. Make sure that your code *PASS* the **automatic tests** in the following **TWO MODES**:

   1. Running ALL test cases together (default test)
   2. Run the test case of each question alone by commenting all other tests and leave one test only, and repeat this for each question. (Tests calls inside "kern\tests.c" in function "TestAssignment3()"

5. Assignments *MUST* be delivered using this online sheet:
   https://docs.google.com/forms/d/e/1FAIpQLSeVJ4gQxrWuwIKb6y3bXlEqtupAOvy2SC5cN_2n4Aoth5xrMg/viewform?usp=sf_link

Delivery Method & Dates

- **Assignment & Bonus:** through the above online form
  - **Early delivery (main questions):** **Fri** of 7th week (8-4-2022 23:59)
  - **Final delivery (main questions + bonus):** **Fri** of 8th week (15-4-2022 23:59)

Questions

Add the following commands to the Kernel:

Find physical address of the given virtual one

Name:

```
fpa <virtual address in HEX>
```

Description:

- This command should find the **EXACT** physical address that corresponds to the given virtual address.
- If the corresponding page exists, then return the **EXACT** physical address (i.e. including the offset). Else, return -1.

Example:

FOS> **fpa** 0xF0001000
Should return 4096.

FOS> **fpa** 0xF0100000
Should return 1,048,576.

FOS> **fpa** 0x100
Should return -1.

FOS> **fpa** 0xF0000005
Should return 5.

FOS> **fpa** 0xF0100555
Should return 1,049,941.

FOS> **cvp** 0x0 0x100000
This command **connects** the virtual address 0 to physical address 1 MB. Now page at address 0 is connected to physical frame

FOS> **fpa** 0x100
Should return 1,048,832.

FOS> **dvp** 0xF0F00000
This command **dis**connects the virtual address F0F00000 from its corresponding physical. Now page at address F0F00000 is **NOT** connected to physical frame

FOS> **fpa** 0xF0F00F00
Should return -1.

Function:

Your code MUST be written inside the following function:

```
int FindPhysicalAddress(char** arguments)
```

- arguments[1]: virtual address of the page in HEXADECIMAL

Return:

- If the page exists, return the **EXACT** physical address (i.e. including the offset).
- Else, return -1

Share range with permissions

Name:

```
srp <va1 in HEX> <va2 in HEX> <size in MB> <r/w>
```

Arguments:

< va1>: the start virtual address of the range to be shared

<va2>: the start virtual address that will see <virtual address 1> physical frame

<size in MB>: size of the sharing range (in Mega Bytes)

<r/w>: sharing permission: 'r' for read-only permission, 'w' for read/write permission

Description:

- This command shares the physical frames of the range **[**<va1>, <va1> + size**)** with the range **[**<va2>, <va2> + size**)**.
- It should set the permissions of the second range **[**<va2>, <va2> + size**)** by the given one
  - <r>: read only
  - <w>: read/write

Example:

FOS> srp F0000000 40000000 256 w

➔ share the entire FOS mappings (256 MB) from KERNEL_BASE to address 1 GB with WRITE permission

FOS> wum F0000000 A ➔ write 'A' at address KERNEL_BASE

FOS> rum F0000000 ➔ read value at KERNEL_BASE

value at address F0000000 = A

FOS> rum 40000000 ➔ read value at 1 GB

value at address **40000000** = A ➔ same value as KERNEL_BASE (**shared!**)

FOS> wum 41000000 C ➔ write 'C' at address 16 MB after 1 GB

FOS> rum F1000000 ➔ read value at 16 MB after KERNEL_BASE

value at address f1000000 = C ➔ same value as the one at 16 MB after 1 GB

FOS> srp F0000000 80000000 128 r

➔ share the half the FOS mappings (128 MB) from KERNEL_BASE to address 2 GB with READ ONLY permission

FOS> rum 80000000	➔ read value at 2 GB
`value at address` **80000000** `= A`	➔ same value as KERNEL_BASE (**shared!**)
FOS> rum 81000000	➔ read value at 16 MB after 2 GB
`value at address` **81000000** `= C`	➔ same value as the one at KERNEL_BASE + 16 MB
FOS> wum 81800000 Z	➔ try writing 'Z' at address 24 MB after 2 GB (should restart as the sharing is done with read only)

Function:

Your code MUST be written inside the following function:

```
void ShareRangeWithPermissions(char** arguments)
```

- arguments[1]: start virtual address of the range to be shared (in HEX)
- arguments[2]: start virtual address of the second range (in HEX)
- arguments[3]: size of the sharing range (in MB)
- arguments[4]: <r/w>: 'r' for read-only permission, 'w' for read/write permission

Helper:

- You may need to use `PERM_xxxxx` constants
- There's a constant in the code called "PAGE_SIZE" which equal to 4KB

Find virtual address of the given frame number

Name:

`fv <Frame Number>`

Description:

- This command searches the ENTIRE virtual memory to find the virtual address of the page that is directly connected to the given `<frame number>`.
- If there're MULTIPLE pages connected to the given frame number, just return the virtual address of the FIRST one.
- If there's no page connected to the given frame number, return -1.

Example:

FOS> fv `1` ➔ return the KERNEL_BASE + 4 KB (0xF0001000)

FOS> cvp `0x00004000 0x1000` ➔ Connect virtual address at 16 KB to physical address 4 KB

[Now, both pages at KERNEL_BASE and at 16 KB are connected to frame #1 (@ 4 KB)]

FOS> fv 1 ➔ Now, it should return 16 KB (0x00004000).

[Since it's the first page that's connected to frame# 1]

FOS> fv 100 ➔ return the KERNEL_BASE + 400 KB (0xF0064000)

FOS> dvp 0xF0064000

This command **disconnects** the virtual address F0064000 from its corresponding physical (frame# 100).
Now page at address F0064000 is **NOT** connected to physical frame

FOS> fv 100 ➔ should return -1

Function:

Your code MUST be written inside the following function:

```
int FindVirtualOfFrameNum(char** arguments)
```

- arguments[1]: frame number

Return:

- If there's one or more pages connected to the given frame number, return the virtual address of the FIRST one.
- Else, return -1.

MAKE SURE that you followed the above instructions carefully. Good Luck isA ☺

BONUS

Connect page number to frame number with permissions

Name:

`cpf <page number> <frame number> <r/w> <s/u>`

Description:

- This command should connect the page with the given `<page number>` to the frame at the given `<frame number>` with the given permissions. After applying it, the page can be accessed with the given permissions.
- Given permissions are:
  - <r/w>: <r> for read only, <w> for read/write
  - <s/u>: <s> for supervisor, <u> for user
- The other permission bits should be set as follows:
  - PRESENT bit = 1
  - ACCESSED & DIRTY bits = 0
  - AVAIL bits = 111
- You should return the **page table ENTRY** of the given page after applying the connection.

Example:

FOS> rum 0x0

FOS should restart since the page at virtual address 0 is not connected to any physical address

FOS> **cpf** 0 768 w u

Now, *connect the page #0 to the frame #768 with WRITE and USER permissions.*

FOS> wum 0x0 A ➔ *should write 'A' at virtual address 0*

FOS> rum 0x0 ➔ *should read 'A'*

FOS> **cpf** 981000 768 w s

Connect the page #981000 to the same frame (#768) with WRITE and SUPERVISOR permissions.

FOS> rum 0xEF808000 ➔ *should read 'A'… **why?!***

FOS> wum 0xEF808000 B ➔ *should write 'B' at virtual address* EF808000

FOS> **cpf** 4 768 r u

Connect the page #4 to the same frame (#768) with read ONLY and USER permissions.

FOS> rum 0x00004000 ➔ *should read 'B'… **why?!***

FOS> wum 0x00004000 C ➔ *should restart FOS…**why?!***

Function:

Your code MUST be written inside the following function:

```
uint32 ConnectPageToFrame(char** arguments)
```

- arguments[1]: page number
- arguments[2]: frame number
- arguments[3]: <r/w>: 'r' for read-only permission, 'w' for read/write permission
- arguments[4]: <s/u>: 's' for supervisor permission, 'u' for user permission

Return: **page table** **ENTRY** of the given page after applying the connection

---

Good luck is A…