Faculty of Computer and Information Sciences

Ain Shams University

Third Year – First Semester

2023 - 2024

Operating Systems

FOS KERNEL PROJECT

Milestone 2
APPENDICES

Contents

APPENDICES

APPENDIX I: ENTRY MANIPULATION in TABLES and DIRECTORY

Location in Code

`/kern/mem/paging_helpers.h`

`/kern/mem/paging_helpers.c`

Permissions in Page Table

Set Page Permission

Function declaration:

```
inline void pt_set_page_permissions(struct Env* ptr_env, uint32 virtual_address, uint32
permissions_to_set, uint32 permissions_to_clear)
```

Description:

Sets the permissions given by "**`permissions_to_set`**" to "1" in the page table entry of the given page (virtual address), and **Clears** the permissions given by "**`permissions_to_clear`**". The environment used is the one given by "`ptr_env`"

Parameters:

`ptr_env`: pointer to environment that you should work on

`virtual_address`: any virtual address of the page

`permissions_to_set`: page permissions to be set to 1

`permissions_to_clear`: page permissions to be set to 0

Examples:

1. to set page PERM_WRITEABLE bit to 1 and set PERM_PRESENT to 0

```
pt_set_page_permissions(environment, virtual_address,
PERM_WRITEABLE, PERM_PRESENT);
```

2. to set PERM_MODIFIED to 0

```
pt_set_page_permissions(environment, virtual_address, 0,
PERM_MODIFIED);
```

Get Page Permission

Function declaration:

```
inline uint32 pt_get_page_permissions(struct Env* ptr_env, uint32 virtual_address )
```

Description:

Returns all permissions bits for the given page (virtual address) in the given environment page directory (ptr_pgdir)

Parameters:

`ptr_env`: pointer to environment that you should work on `virtual_address`: any virtual address of the page

Return value:

Unsigned integer containing all permissions bits for the given page

Example:

To check if a page is modified:

```
uint32 page_permissions = pt_get_page_permissions(environment, virtual_address);
if(page_permissions & PERM_MODIFIED)
{
        . . .
}
```

Clear Page Table Entry

Function declaration:

```
inline void pt_clear_page_table_entry(struct Env* ptr_env, uint32 virtual_address)
```

Description:

Set the entry of the given page inside the page table to **NULL**. This indicates that the page is no longer exists in the memory.

Parameters:

`ptr_env`: pointer to environment that you should work on

`virtual_address`: any virtual address inside the page

Permissions in Page Directory

Clear Page Dir Entry

Function declaration:

```
inline void pd_clear_page_dir_entry(struct Env* ptr_env, uint32 virtual_address)
```

Description:

Set the entry of the page table inside the page directory to **NULL**. This indicates that the page table, which contains the given virtual address, becomes no longer exists in the whole system (memory and page file).

Parameters:

`ptr_env`: pointer to environment that you should work on

`virtual_address`: any virtual address inside the range that is covered by the page table

Check if a Table is Used

Function declaration:

```
inline uint32 pd_is_table_used(Env* ptr_environment, uint32 virtual_address)
```

Description:

Returns a value indicating whether the table at "`virtual_address`" was used by the processor

Parameters:

`ptr_environment`: pointer to environment

`virtual_address`: any virtual address inside the table

Return value:

0: if the table at “`virtual_address`” is not used (accessed) by the processor

1: if the table at “`virtual_address`” is used (accessed) by the processor

Example:

```
if(pd_is_table_used(faulted_env, virtual_address))
{
	…
}
```

Set a Table to be Unused

Function declaration:

```
inline void pd_set_table_unused(Env* ptr_environment, uint32 virtual_address)
```

Description:

Clears the “Used Bit” of the table at `virtual_address` in the given directory

Parameters:

`ptr_environment`: pointer to environment

`virtual_address`: any virtual address inside the table

APPENDIX II: PAGE FILE HELPER FUNCTIONS

Location in Code

/kern/disk/pagefile_manager.h

/kern/disk/pagefile_manager.c

Pages Functions

Add a new environment page to the page file

Function declaration:

```
int pf_add_empty_env_page( struct Env* ptr_env, uint32 virtual_address, uint8 
initializeByZero);
```

Description:

Add a new environment page with the given virtual address to the page file and initialize it by zeros. Used during the initial loading of a process (inside `env_create`)

Parameters:

`ptr_env`: pointer to the environment that you want to add the page for it.

`virtual_address`: the virtual address of the page to be added.

`initializeByZero`: indicate whether you want to initialize the new page by ZEROs or not.

Return value:

= 0: the page is added successfully to the page file.

= `E_NO_PAGE_FILE_SPACE`: the page file is full, can't add any more pages to it.

Example:

In dynamic allocation: let for example we want to dynamically allocate 1 page at the beginning of the heap (i.e. at address USER_HEAP_START) without initializing it, so we need to add this page to the page file as follows:

```
int ret = pf_add_empty_env_page(ptr_env, USER_HEAP_START, 0);

if (ret == E_NO_PAGE_FILE_SPACE)

      panic("ERROR: No enough virtual space on the page file");
```

Read an environment page from the page file to the main memory

Function declaration:

```
int pf_read_env_page(struct Env* ptr_env, void *virtual_address);
```

Description:

Read an existing environment page at the given virtual address from the page file.

Parameters:

ptr_env: pointer to the environment that you want to read its page from the page file.

virtual_address: the virtual address of the page to be read.

Return value:

= 0: the page is read successfully to the given virtual address of the given environment.

= `E_PAGE_NOT_EXIST_IN_PF`: the page doesn't exist on the page file (i.e. no one added it before to the page file).

Example:

In placement steps: let for example there is a page fault occur at certain virtual address, then, we want to read it from the page file and place it in the main memory at the faulted virtual address as follows:

```
int ret = pf_read_env_page(ptr_env, fault_va);

if (ret == E_PAGE_NOT_EXIST_IN_PF)

{      ...    }
```

Update certain environment page in the page file by contents from the main memory

Function declaration:

```
int pf_update_env_page(struct Env* ptr_env, uint32 virtual_address, struct
FrameInfo* modified_page_frame_info));
```

Description:

- **Updates** an existing page in the page file by the given frame in memory.
- If the page **does not exist** in page file & **belongs** to either **USER HEAP** or **STACK**, it **adds** it to the page file

Parameters:

ptr_env: pointer to the environment that you want to update its page on the page file.

virtual_address: the virtual address of the page to be updated.

modified_page_frame_info: the FrameInfo* related to this page.

Return value:

= 0: the page is updated successfully on the page file.

= E_NO_PAGE_FILE_SPACE: the page file is full, can't add any more pages to it.

Example:

```
struct FrameInfo *ptr_frame_info = get_frame_info(…);
int ret = pf_update_env_page(environment, virtual_address, ptr_frame_info);
```

Remove an existing environment page from the page file

Function declaration:

```
void pf_remove_env_page(struct Env* ptr_env, uint32 virtual_address);
```

Description:

Remove an existing environment page at the given virtual address from the page file.

Parameters:

ptr_env: pointer to the environment that you want to remove its page (or table) on the page file.

virtual_address: the virtual address of the page to be removed.

Example:

Let's assume for example we want to free 1 page at the beginning of the heap (i.e. at address USER_HEAP_START), so we need to remove this page from the page file as follows:

```
pf_remove_env_page(ptr_env, USER_HEAP_START);
```

APPENDIX III: WORKING SET STRUCTURE & HELPER FUNCTIONS

Location in Code

`inc/environment_definitions.h`

`kern/mem/working_set_manager.h`

`kern/mem/working_set_manager.c`

Working Set Structure

Each environment has a **working set list** (**`page_WS_list`**) that is initialized at the env_create()

This list should hold pointers of type **`struct WorkingSetElement`** containing info about the currently loaded pages in memory.

Each struct holds two important values about each page:

1. User virtual address of the page
2. Previous & Next pointers to be used by list

The working set list is defined inside the environment structure "`struct Env`" located in "inc/environment_definitions.h".

Its max size is set in "**`page_WS_max_size`**" during the env_create().

"**`page_last_WS_element`**" will point to

1. the next location in the WS after the last set one If list is full.
2. Null if the list is not full.

```
struct WorkingSetElement {
      uint32 virtual_address;  // the virtual address of the page
      LIST_ENTRY(WorkingSetElement) prev_next_info;   // list link pointers

};
struct Env {
      .
      .
      .
      //page working set management
      struct WS_List page_WS_list;
      unsigned int page_WS_max_size;
      // used for FIFO & clock algorithm, the next item (page) pointer
      uint32 page_last_WS_element;
};
```

Figure 1: Definitions of the working set & its index inside `struct Env`

Working Set Functions

Print Working Set

Function declaration:

```
inline void env_page_ws_print(struct Env* e)
```

Description:

Print the page working set together with the used, modified and buffered bits + time stamp. It also shows where the **`page_last_WS_element`** of the working set is point to.

Parameters:

`e`: pointer to an environment

Flush certain Virtual Address from Working Set

Description:

Search for the given virtual address inside the working set of **"e"** and, if found, removes its entry.

Function declaration:

```
inline void env_page_ws_invalidate(struct Env* e, uint32 virtual_address)
```

Parameters:

`e`: pointer to an environment

`virtual_address`: the virtual address to remove from working set

APPENDIX IV: MEMORY MANAGEMENT FUNCTIONS

Basic Functions

The basic **memory manager functions** that you may need to use are defined in "`kern/mem/memory_manager.c`" file:

Function Name	Description
`allocate_frame`	Used to allocate a free frame from the free frame list
`free_frame`	Used to free a frame by adding it to free frame list
`map_frame`	Used to map a single page with a given virtual address into a given allocated frame, simply by setting the directory and page table entries
`get_page_table`	Get a pointer to the page table if exist
`create_page_table`	Create a new page table by allocating a new page at the kernel heap, zeroing it and finally linking it with the directory
`unmap_frame`	Used to un-map a frame at the given virtual address, simply by clearing the page table entry
`get_frame_info`	Used to get both the page table and the frame of the given virtual address

Other Helpers Functions

There are some **helper functions** that we may need to use them in the rest of the course:

Function	Description	Defined in...
`PDX (uint32 virtual address)`	Gets the page directory index in the given virtual address (10 bits from 22 – 31).	`Inc/mmu.h`
`PTX (uint32 virtual address)`	Gets the page table index in the given virtual address (10 bits from 12 – 21).	`Inc/mmu.h`
`ROUNDUP (uint32 value, uint32 align)`	Rounds a given "value" to the nearest upper value that is divisible by "align".	`Inc/types.h`
`ROUNDDOWN (uint32 value, uint32 align)`	Rounds a given "value" to the nearest lower value that is divisible by "align".	`Inc/types.h`
`tlb_invalidate (uint32* page_directory, uint32 virtual address)`	Refresh the cache memory (TLB) to remove the given virtual address from it.	`Kern/mem/memory_manager.c`
`isKHeapPlacementStrategyFIRSTFIT() …]`	Check which strategy is currently selected using the given functions.	`Kern/mem/kheap.h`

APPENDIX V: COMMAND PROMPT

Location in Code

`kern/cmd/commands.h`

`kern/cmd/commands.c`

Run process

Name: `run <prog_name> <page_WS_size>`

Arguments:

prog_name: name of user program to load and run (should be identical to name field in UserProgramInfo array).

page_WS_size: specify the max size of the page WS for this program

Description:

Load the given program into the virtual memory (RAM & Page File) then run it.

Print current user heap placement strategy (NEXT FIT, BUDDY, BEST FIT, …)

Name: `uheap?`

Description:

Print the current USER heap placement strategy (NEXT FIT, BUDDY, BEST FIT, …).

Changing user heap placement strategy (NEXT FIT, BEST FIT, …)

Name: `uhnextfit` (`uhbestfit`, `uhfirstfit`, `uhworstfit`)

Description:

Set the current user heap placement strategy to NEXT FIT (BEST FIT, …).

Print current kernel heap placement strategy (NEXT FIT, BEST FIT, …)

Name: `kheap?`

Description:

Print the current KERNEL heap placement strategy (NEXT FIT, BEST FIT, …).

Changing kernel heap placement strategy (NEXT FIT, BEST FIT, …)

Name: `khnextfit (khbestfit`, `khfirstfit`)

Description:

Set the current KERNEL heap placement strategy to NEXT FIT (BEST FIT, …).