­­­­

**Faculty of Computer and Information Sciences**

**Ain Shams University**

**Third Year – First Semester**

**2024 - 2025**

**FOS KERNEL PROJECT**

**Milestone 2 APPENDICES**

**Operating Systems**

Contents

[APPENDIX I: ENTRY MANIPULATION in TABLES and DIRECTORY 3](#_Toc149505101)

[Location in Code 3](#_Toc149505102)

[Permissions in Page Table 3](#_Toc149505103)

[Set Page Permission 3](#_Toc149505104)

[Get Page Permission 3](#_Toc149505105)

[Clear Page Table Entry 4](#_Toc149505106)

[Permissions in Page Directory 4](#_Toc149505107)

[Clear Page Dir Entry 4](#_Toc149505108)

[Check if a Table is Used 4](#_Toc149505109)

[Set a Table to be Unused 5](#_Toc149505110)

[APPENDIX II: PAGE FILE HELPER FUNCTIONS 6](#_Toc149505111)

[Location in Code 6](#_Toc149505112)

[Pages Functions 6](#_Toc149505113)

[Add a new environment page to the page file 6](#_Toc149505114)

[Read an environment page from the page file to the main memory 6](#_Toc149505115)

[Update certain environment page in the page file by contents from the main memory 7](#_Toc149505116)

[Remove an existing environment page from the page file 7](#_Toc149505117)

[APPENDIX III: WORKING SET STRUCTURE & HELPER FUNCTIONS 8](#_Toc149505118)

[Location in Code 8](#_Toc149505119)

[Working Set Structure 8](#_Toc149505120)

[Working Set Functions 9](#_Toc149505121)

[Print Working Set 9](#_Toc149505122)

[Flush certain Virtual Address from Working Set 9](#_Toc149505123)

[APPENDIX IV: MEMORY MANAGEMENT FUNCTIONS 10](#_Toc149505124)

[Basic Functions 10](#_Toc149505125)

[Other Helpers Functions 10](#_Toc149505126)

[APPENDIX V: COMMAND PROMPT 11](#_Toc149505127)

[Location in Code 11](#_Toc149505128)

[Run process 11](#_Toc149505129)

[Print current user heap placement strategy (NEXT FIT, BUDDY, BEST FIT, …) 11](#_Toc149505130)

[Changing user heap placement strategy (NEXT FIT, BEST FIT, …) 11](#_Toc149505131)

[Print current kernel heap placement strategy (NEXT FIT, BEST FIT, …) 11](#_Toc149505132)

[Changing kernel heap placement strategy (NEXT FIT, BEST FIT, …) 11](#_Toc149505133)

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);

1. 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, …).