Faculty of Computer and Information Sciences

Ain Shams University

Third Year – First Semester

2023 - 2024

Operating Systems

FOS KERNEL PROJECT

Milestone 3
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`

LRU Working Set Structure

Each environment has an **Active List & Second Chance List** that are initialized at the env_create(). These lists should hold pointers of type **`struct WorkingSetElement`** containing info about the currently loaded pages in RAM.

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

It is defined inside the environment structure "`struct Env`" located in "inc/environment_definitions.h".

Max size of each list is also defined inside the `Env` & already set in during the env_create().

```
struct Env
{
    …….
    struct WS_List ActiveList;      //LRU Approx: ActiveList that should work as FCFS
    struct WS_List SecondList;      //LRU Approx: SecondList that should work as LRU
    int ActiveListSize ;            //LRU Approx: Max allowed size of ActiveList
    int SecondListSize ;            //LRU Approx: Max allowed size of SecondList
    int page_WS_max_size;           //Max WS Size = ActiveListSize + SecondListSize
…….
```

Figure 1: Definitions of the LRU Lists & their Sizes inside `struct Env`

FIFO 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 RAM.

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

It 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 2: Definitions of the working set list & its size inside `struct Env`

Working Set Functions

Print Working Set

Function declaration:

`inline void env_page_ws_print(struct Env* e)`

Description:

CASE1: If LRU List Approx. Replacement

- Print the content of the **Active List** & **Second List**.

CASE2: Else, (any other replacement)

- Print the page **Working Set List** 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

Function declaration:

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

Description:

CASE1: If LRU List Approx. Replacement

- Search for the given virtual address inside the **Active List** & **Second List** of **"e"** and, if found:
  - **removes** its entry from the corresponding list & **update** the lists accordingly.
  - **Unmap** its page from memory.

CASE2: Else, (any other replacement)

- Search for the given virtual address inside the **Working Set List** of **"e"** and, if found:
  - **removes** its entry from the list.

Parameters:

`e`: pointer to an environment

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

APPENDIX IV: SCHEDULER STRUCTURE & HELPER FUNCTIONS

Location in Code

kern/cpu/sched.h

kern/cpu/kclock.h

kern/cpu/sched_helpers.h

kern/cpu/sched_helpers.c

Data Structures

1. Number of ready queues
2. Array of ready queues: to be created and initialized later during the initialization
3. Array of quantums in millisecond: to be created and initialized later during the initialization

```
struct Env_Queue *env_ready_queues; // Ready queue(s)
uint8 *quantums ;                   // Quantum(s) in ms
uint8 num_of_ready_queues ;        // Number of ready queue(s)
```

Helper Functions

Set quantum of the CPU

Function declaration:

```
void kclock_set_quantum (uint8 quantum_in_ms);
```

Description:

Set the CPU quantum by the given quantum

Parameters:

quantum in ms

Timer Ticks

Function declaration:

```
int timer_ticks();
```

Description:

Get the current number of ticks since the beginning of the run

Initialize Queue

Description:

Initialize a new queue by setting to NULL (ZERO) its head, tail and size.

Function declaration:

```
void init_queue(struct Env_Queue* queue);
```

Parameters:

`queue`: pointer (i.e. address) to the queue to be initialized.

Example: initialize a newly created queue

```
struct Env_Queue myQueue ;

init_queue(&myQueue);
```

Get Queue Size

Description:

Get the current number of elements inside the queue.

Function declaration:

```
int queue_size(struct Env_Queue* queue);
```

Parameters:

`queue`: pointer (i.e. address) to the queue to get its size.

Example:

```
struct Env_Queue myQueue ;

...

int size = queue_size(&myQueue);
```

Enqueue Environment

Description:

Add the given environment into the head of the given queue.

Function declaration:

```
void enqueue(struct Env_Queue* queue, struct Env* env);
```

Parameters:

`queue`: pointer (i.e. address) to the queue to insert on it.

`env`: pointer to the environment to be inserted.

Example: add current environment to myQueue

```
struct Env_Queue myQueue ;

...

enqueue(&myQueue, curenv);
```

Dequeue Environment

Description:

Get and remove the environment from the tail of the given queue.

Function declaration:

```
struct Env* dequeue(struct Env_Queue* queue);
```

Parameters:

`queue`: pointer (i.e. address) to the queue.

Return value:

pointer to the environment on the tail of the queue (after removing it from the queue).

Example:

```
struct Env* env;

...

env = dequeue(&myQueue);
```

Remove Environment from Queue

Description:

Remove a given environment from the queue.

Function declaration:

```
void remove_from_queue(struct Env_Queue* queue, struct Env* env);
```

Parameters:

`queue`: pointer (i.e. address) to the queue.

`env`: pointer to the environment to be removed.

Find Environment in the Queue

Description:

Search for an environment with the given ID in the given queue.

Function declaration:

```
struct Env* find_env_in_queue(struct Env_Queue* queue, uint32 envID);
```

Parameters:

`queue`: pointer (i.e. address) to the queue.

`envID`: environment ID to search for.

Return value:

If found: pointer to the environment with the given ID.

Else: null.

Example: find environment with ID = 1024

```
struct Env* env;
env = find_env_in_queue(&myQueue, 1024);
```

Insert Environment to the NEW Queue

Function declaration:

```
void sched_insert_new(struct Env* env);
```

Description:

Enqueue the given environment to the new queue in order to indicate that it's loaded now.

Environment status becomes NEW.

Parameters:

`env`: pointer to the environment to be inserted.

Remove Environment from NEW Queue

Function declaration:

```
void sched_remove_new(struct Env* env);
```

Description:

Remove the given environment from the new queue.

Environment status becomes UNKNOWN.

Parameters:

`env`: pointer to the environment to be removed.

Insert a NEW Environment to the FIRST READY Queue

Function declaration:

```
void sched_insert_ready0(struct Env* env);
```

Description:

Enqueue the given environment to the FIRST ready queue, so, it'll be scheduled by the CPU.

Environment status becomes READY.

Parameters:

`env`: pointer to the environment to be inserted.

Remove Environment from the READY Queue(s)

Function declaration:

```
void sched_remove_ready(struct Env* env);
```

Description:

Search for and remove the given environment from the ready queue(s), so, it'll be NOT scheduled anymore by the CPU.

Environment status becomes UNKNOWN.

Parameters:

`env`: pointer to the environment to be removed.

Insert Environment to the EXIT Queue

Function declaration:

```
void sched_insert_exit(struct Env* env);
```

Description:

Enqueue the given environment to the exit queue to indicate that it's finished now.

Environment status becomes EXIT.

Parameters:

`env`: pointer to the environment to be inserted.

Remove Environment from EXIT Queue

Function declaration:

```
void sched_remove_exit(struct Env* env);
```

Description:

Remove the given environment from the exit queue.

Environment status becomes UNKNOWN.

Parameters:

`env`: pointer to the environment to be removed.

APPENDIX V: MEMORY MANAGEMENT FUNCTIONS

Basic Functions

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

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 VI: COMMAND PROMPT

Location in Code

kern/cmd/commands.h

kern/cmd/commands.c

Run Process (for LRU Lists or Others)

Name: **run** <*prog_name*> <*page_WS_size*> [<*LRU_second_list_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

LRU_second_list_size: specify the max size of the **Second Chance List** for this program ***[OPTIONAL]***

Description:

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

Load Process (for LRU Lists or Others)

Name: **load** <*prog_name*> <*page_WS_size*> [<*LRU_second_list_size*>]

Arguments:

prog_name: name of user program to load it into the virtual memory (should be identical to name field in UserProgramInfo array).

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

LRU_second_list_size: specify the max size of the **Second Chance List** for this program ***[OPTIONAL]***

Description:

JUST Load the given program into the virtual memory (RAM & Page File) but **don't run** it.

Kill Process

Name: **kill <env ID>**

Arguments:

Env ID: ID of the environment to be killed (i.e. freeing it).

Description:

Kill the given environment by calling env_free.

Run All Loaded Processes

Name: **runall**

Description:

Run all programs that are previously loaded by **"ld"** command using Round Robin scheduling algorithm.

Print All Processes

Name: **`printall`**

Description:

Print all programs' names that are currently exist in new, ready and exit queues.

Kill All Processes

Name: **`killall`**

Description:

Kill all programs that are currently loaded in the system (new, ready and exit queues. (by calling env_free).

Print Current Scheduler Method

Name: **`sched?`**

Description:

Print the current scheduler method with its quantum(s) (RR or BSD).

Change the Scheduler to BSD

Name: **`schedBSD`** `<number of levels> <quantum>`

Description:

Change the scheduler to BSD with the given number of levels and the quantum (in ms).

Print Current Replacement Policy (fifo, LRU, …)

Name: **`rep?`**

Description:

Print the current page replacement algorithm (CLOCK, LRU, FIFO…).

Change Replacement Policy (fifo, LRU, …)

Name: **`lru 2`** *(***`fifo, clock, modifiedclock`**…*)*

Description:

Set the current page replacement algorithm to CLOCK (LRU list approx, FIFO,…).

Print Current User Heap Strategy (NEXT FIT, BUDDY, BEST FIT, …)

Name: **`uheap?`**

Description:

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

Change User Heap 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 Strategy (NEXT FIT, BEST FIT, …)

Name: **`kheap?`**

Description:

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

Change 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, …).

APPENDIX VII: FIXED POINT OPERATIONS

Location in Code

`inc/fixed_point.h`

Functions

- Let **x** and **y** be **fixed-point p.q** numbers, let **n** be an **integer,** and **f** is **1 << q**

Convert n to fixed point:	n × f	`fix_int(int n)`
Convert x to integer (rounding toward zero):	x / f	`fix_trunc(fixed_point_t x)`
Convert x to integer (rounding to nearest):	(x + f/2) / f if x >= 0, (x - f/2) / f if x <= 0.	`fix_round(fixed_point_t x)`
Add x and y:	x + y	`fix_add(fixed_point_t x, fixed_point_t y)`
Subtract y from x:	x – y	`fix_sub(fixed_point_t x, fixed_point_t y)`
Add x and n:	x + n × f	
Subtract n from x:	x - n × f	
Multiply x by y:	((int64) x) × y / f	`fix_mul(fixed_point_t x, fixed_point_t y)`
Multiply x by n:	x × n	`fix_scale(fixed_point_t x, int n)`
Divide x by y:	((int64) x) × f / y	`fix_mul(fixed_point_t x, fixed_point_t y)`
Divide x by n:	x / n	`fix_unscale(fixed_point_t x, int n)`