­­­­

**Faculty of Computer and Information Sciences**

**Ain Shams University**

**Third Year – First Semester**

**2024 - 2025**

**FOS KERNEL PROJECT**

**Milestone 3 APPENDICES**

**Operating Systems**

Contents

[APPENDIX I: ENTRY MANIPULATION in TABLES and DIRECTORY 4](#_Toc183807523)

[Location in Code 4](#_Toc183807524)

[Permissions in Page Table 4](#_Toc183807525)

[Set Page Permission 4](#_Toc183807526)

[Get Page Permission 4](#_Toc183807527)

[Clear Page Table Entry 5](#_Toc183807528)

[Permissions in Page Directory 5](#_Toc183807529)

[Clear Page Dir Entry 5](#_Toc183807530)

[Check if a Table is Used 5](#_Toc183807531)

[Set a Table to be Unused 6](#_Toc183807532)

[APPENDIX II: PAGE FILE HELPER FUNCTIONS 7](#_Toc183807533)

[Location in Code 7](#_Toc183807534)

[Pages Functions 7](#_Toc183807535)

[Add a new environment page to the page file 7](#_Toc183807536)

[Read an environment page from the page file to the main memory 7](#_Toc183807537)

[Update certain environment page in the page file by contents from the main memory 8](#_Toc183807538)

[Remove an existing environment page from the page file 8](#_Toc183807539)

[APPENDIX III: WORKING SET STRUCTURE & HELPER FUNCTIONS 9](#_Toc183807540)

[Location in Code 9](#_Toc183807541)

[Working Set Structure 9](#_Toc183807542)

[Working Set Functions 9](#_Toc183807543)

[Print Working Set 9](#_Toc183807544)

[Flush certain Virtual Address from Working Set 10](#_Toc183807545)

[APPENDIX IV: SCHEDULER STRUCTURE & HELPER FUNCTIONS 11](#_Toc183807546)

[Location in Code 11](#_Toc183807547)

[Data Structures 11](#_Toc183807548)

[Helper Functions 11](#_Toc183807549)

[Set quantum of the CPU 11](#_Toc183807550)

[Timer Ticks 11](#_Toc183807551)

[Initialize Queue 11](#_Toc183807552)

[Get Queue Size 12](#_Toc183807553)

[Enqueue Environment 12](#_Toc183807554)

[Dequeue Environment 12](#_Toc183807555)

[Remove Environment from Queue 13](#_Toc183807556)

[Find Environment in the Queue 13](#_Toc183807557)

[Insert Environment to the NEW Queue 13](#_Toc183807558)

[Remove Environment from NEW Queue 14](#_Toc183807559)

[Insert a NEW Environment to the FIRST READY Queue 14](#_Toc183807560)

[Insert a NEW Environment into the READY Queue according to its Priority 14](#_Toc183807561)

[Remove Environment from the READY Queue(s) 14](#_Toc183807562)

[Insert Environment to the EXIT Queue 15](#_Toc183807563)

[Remove Environment from EXIT Queue 15](#_Toc183807564)

[APPENDIX V: MEMORY MANAGEMENT FUNCTIONS 16](#_Toc183807565)

[Basic Functions 16](#_Toc183807566)

[Other Helpers Functions 16](#_Toc183807567)

[APPENDIX VI: COMMAND PROMPT 17](#_Toc183807568)

[Location in Code 17](#_Toc183807569)

[Run Process (for LRU Lists or Others) 17](#_Toc183807570)

[Load Process (for LRU Lists or Others) 17](#_Toc183807571)

[Set Process Priority 17](#_Toc183807572)

[Set Starvation Threshold 18](#_Toc183807573)

[Kill Process 18](#_Toc183807574)

[Run All Loaded Processes 18](#_Toc183807575)

[Print All Processes 18](#_Toc183807576)

[Kill All Processes 18](#_Toc183807577)

[Print Current Scheduler Method 18](#_Toc183807578)

[Change the Scheduler to Priority RR 19](#_Toc183807579)

[Print Current Replacement Policy (fifo, LRU, …) 19](#_Toc183807580)

[Change Replacement Policy (fifo, LRU, …) 19](#_Toc183807581)

[Print Current User Heap Strategy (NEXT FIT, BUDDY, BEST FIT, …) 19](#_Toc183807582)

[Change User Heap Strategy (NEXT FIT, BEST FIT, …) 19](#_Toc183807583)

[Print Current Kernel Heap Strategy (NEXT FIT, BEST FIT, …) 19](#_Toc183807584)

[Change Kernel Heap Placement Strategy (NEXT FIT, BEST FIT, …) 19](#_Toc183807585)

[APPENDIX VII: FIXED POINT OPERATIONS 20](#_Toc183807586)

[Location in Code 20](#_Toc183807587)

[Functions 20](#_Toc183807588)

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 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 1: 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**

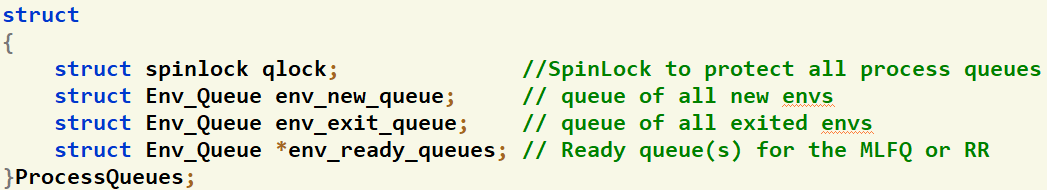
**kern/cpu/sched\_helpers.c**

## Data Structures

1. Number of ready queues A screenshot of a computer

   Description automatically generated
2. Array of quantums in millisecond: to be created and initialized later during the initialization A screenshot of a computer

   Description automatically generated
3. ProcessQueues struct:
   1. Queue of “New” processes
   2. Array of “Ready” queues: to be created and initialized later during the initialization
   3. Queue of “Exit” processes
   4. Protection lock “qlock”



## 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.

### Insert a NEW Environment into the READY Queue according to its Priority

#### Function declaration:

**void sched\_insert\_ready(struct Env\* env);**

#### Description:

Enqueue the given environment to the ready queue corresponding to its priority index, 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 |
| tlbflush() | Clear the entire content of the TLB | inc/x86.h |
| 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\_2nd\_list\_size>] [<priority>]

#### 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\_2nd\_list\_size: specify the max size of the **Second Chance List** for this program ***[OPTIONAL]***

priority: specify the process priority for the **Scheduler (default 0) *[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\_2nd\_list\_size>] [<priority>]

#### 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\_2nd\_list\_size: specify the max size of the **Second Chance List** for this program ***[OPTIONAL]***

priority: specify the process priority for the **Scheduler (default 0) *[OPTIONAL]***

#### Description:

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

### Set Process Priority

#### Name: setPri <envID> <priority>

#### Arguments:

envID: ID of the created environment (will be printed when execute load/run command).

priority: specify the process priority for the **Scheduler**

#### Description:

Set the priority of the given environment.

### Set Starvation Threshold

#### Name: setStarvThr <starvationThreshold>

#### Arguments:

starvationThreshold: new value of the starvation threshold (i.e. limit on # ticks before promoting the process priority.

#### Description:

Set the starvation threshold of priority-based scheduler.

### 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, BSD,…).

### Change the Scheduler to Priority RR

***Name:* schedPRIRR** <#priorities> <quantum> <starvThresh>

***Description:***

Change the scheduler to Priority RR with the given # priorities, quantum (in ms) and starvation threshold

### Print Current Replacement Policy (fifo, LRU, …)

#### Name: rep?

#### Description:

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

### Change Replacement Policy (fifo, LRU, …)

#### Name: nclock N (fifo, clock, lru, 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)** |