OS'25 Project

PART V: **TESTING II**

(INDIVIDUAL MODULES**)**

Agenda

- **CODE UPDATE**
- **PART VI: TESTING OF INDIVIDUAL MODULES**
  1. Fault Handler II
  2. User Heap
  3. Shared Memory
  4. CPU Scheduling
  5. Kernel Protection
  6. Fault Handler III

CAUTION

During your solution, any SHARED data MUST be PROTECTED by critical section via LOCKS

ACTION

CORRECTNESS by DESIGN

Be LOGIC-DRIVEN... Not TEST-DRIVEN

TESTING…

IMPORTANT NOTE

Before testing ANY module, you MUST DO **enable** the kernel heap:

- In '**inc/memlayout.h**': set **`USE_KHEAP`** to **1**

Code Updates

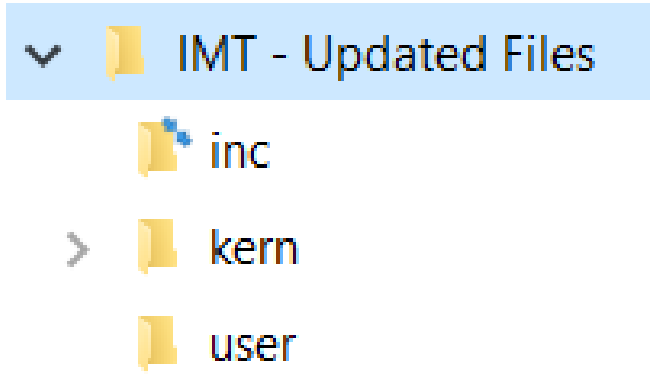
INDIVIDUAL MODULES TEST

New Files

1. **SELECT ALL** in the given "**IMT - Updated Files**" folder,

2. **COPY** & **PASTE** (**REPLACE ALL**) in **FOS_CODES/FOS_PROJECT_2025_TEMPLATE/**

NOTE: If any of these files are already edited by you, make sure to apply the edits in the new files



Code Fix

In **page_fault_handler()** function inside **kern/trap/fault_handler.c** apply the following fix:

```
else if (isPageReplacmentAlgorithmOPTIMAL())
{
    //TODO: [PROJECT'25.IM#1] FAULT HANDLER II - #3 Clock Replacement
    //Your code is here
    //Comment the following line
    panic("page_fault_handler().REPLACEMENT is not implemented yet...!!");
}
```

```
else if (isPageReplacmentAlgorithmCLOCK())
{
    //TODO: [PROJECT'25.IM#1] FAULT HANDLER II - #3 Clock Replacement
    //Your code is here
    //Comment the following line
    panic("page_fault_handler().REPLACEMENT is not implemented yet...!!");
}
```

Code Usage

To iterate on any list, use the **UPDATED SAFE VERSION** of LIST FOREACH as follows:

```
LIST_FOREACH_SAFE (Type_inside_list* iterator, Linked_List* list, Type_inside_list)
```

Parameters:

`iterator`: pointer to the current element in the list

`list`: pointer to the linked list to loop on its elements

`Type_inside_list`: name of the struct that is used inside the list (i.e. type of each element)

Example:

```
struct ELEMENTDataType *element;
LIST_FOREACH_SAFE(element, &(ActiveList), ELEMENTDataType)
{
        //write your code.
}
```

TEST: Fault Handler II

INDIVIDUAL MODULE#1 – TEST

TEST: Fault Handler II – **OPTIMAL**

- The time limit of each individual test: **max of 5 sec / each**
- **MAKE SURE** to switch the replacement policy before start testing: **`FOS> optimal`**

#	Test Functionality	Test
1	***tst_page_replacement_optimal_1.c (toptimal1):*** check working set, final reference stream and allocations in memory after applying the **optimal** algorithm with **small** working set.	➢ **FOS>** run toptimal1 11
2	***tst_page_replacement_optimal_2.c (toptimal2):*** check working set, final reference stream and allocations in memory after applying the **optimal** algorithm with **large** working set.	➢ **FOS>** run toptimal2 3000
3	***tst_page_replacement_optimal_3.c (toptimal3):*** check calculated number of page faults after applying the **optimal** algorithm with **small** working set.	➢ **FOS>** run toptimal3 11

TEST: Fault Handler II – **CLOCK**

- The time limit of each individual test: **max of 5 sec / each**
- **MAKE SURE** to switch the replacement policy before start testing: **`FOS> clock`**

#	Test Functionality	Test
1	***tst_page_replacement_alloc.c (tpr1):*** tests allocation in memory and page file after page replacement.	➢ **FOS>** clock ➢ **FOS>** run tpr1 11
2	***tst_page_replacement_stack.c (tpr2):*** tests page replacement of **stack** (i.e. new pages) (creating, modifying and reading them)	➢ **FOS>** clock ➢ **FOS>** run tpr2 6
3	***tst_page_replacement_clock_1.c (tclock1):*** tests page replacement by **Clock** algorithm (it checks working set before & after replacements)	➢ **FOS>** clock ➢ **FOS>** run tclock1 11
4	***tst_page_replacement_clock_2.c (tclock2):*** tests page replacement by **Clock** algorithm (check maintaining the correct FIFO order, during the **placement,** after removing some pages from the working set)	➢ **FOS>** clock ➢ **FOS>** run tclock2 11

TEST: Fault Handler II

Module	Function	Diff.	Testing	Notes
OPTIMAL Replac.	Ref Stream	L2	`FOS> run toptimal1 11`	
			`FOS> run toptimal2 3000`	
	Trace Num of Faults	L3	`FOS> run toptimal3 11`	Depend on Ref Stream
CLOCK Replac.	Allocation in RAM & Disk	L2	`FOS> run tpr1 11`	
			`FOS> run tpr2 6`	
	Algorithm itself		`FOS> run tclock1 11`	Handled in replacement
			`FOS> run tclock2 11`	Handled in placement

"Congratulations!! test [TEST NAME] is completed successfully"

To ensure the test success, a congratulations message like this **MUST appear without any ERROR messages or PANICs**.

TEST: User Heap

INDIVIDUAL MODULE#2 – TEST

TEST: User Heap

- The time limit of each individual test: **max of 20 sec / each**
- **MAKE SURE** to switch the strategy before start testing: **`FOS> uhcustomfit`**

#	Test Functionality	Test
1	***tst_malloc_1.c (tm1):*** tests **malloc()** & **allocate_user_mem()** in **PAGE ALLOCATOR**. It validates: 1. return addresses from the malloc() 2. NOTHING is allocated in page file or memory 3. memory access (read & write) of the allocated spaces (placement of fault handler should work) 4. number of allocated frames and the WS entries after each memory access	➢ **FOS>** run tm1 3000
2	***tst_free_1.c (tf1):*** tests **free()** & **free_user_mem()** in **PAGE ALLOCATOR**. It validates: 1. number of freed frames by free_user_mem() 2. Removing the allocated pages from working set (if any) 3. memory access (read & write) of the removed spaces (should not be allowed)	➢ **FOS>** run tf1 3000
3	***tst_custom_fit_1.c (tcf1):*** tests the **CUSTOM FIT strategy** in **PAGE ALLOCATOR**. Tests both granted and non-granted requests. (It depends on `free`).	➢ **FOS>** run tcf1 3000

TEST: User Heap

Module	Function	Diff.	Testing	Notes
USER HEAP	malloc()	L3	`FOS> run tm1 3000`	
	allocate_user_mem()	L1		
	free()	L2	`FOS> run tf1 3000`	Depend on malloc
	free_user_mem()	L2		
	Custom Fit Cases		`FOS> run tcf1 3000`	Depend on malloc & free

"test [TEST NAME] completed. Evaluation = …%"

To ensure the test success, a congratulations message like this **MUST appear without any ERROR messages or PANICs**.

TEST: Shared Memory

INDIVIDUAL MODULE#3 – TEST

TEST: Shared Memory

- Run each test independently in a **FRESH SEPARATE RUN**.
- The time limit of each individual test: **max of 15 sec / each**
- **MAKE SURE** to switch the strategy:

```
FOS> uhcustomfit
```

#	Function	Test
1	**`smalloc & createSharedObject`** ***tst_sharing_1.c (tshr1):*** It tests the creation of shared objects. It validates the returned addresses and the number of allocated frames. It also checks the memory access (read & write) of the created shared objects.	**FOS>** run tshr1 3000
2	**`smalloc & createSharedObject`** ***tst_sharing_3.c (tshr3):*** It tests handling the **special cases** of shared objects creation. Namely, creating objects with same name, creating large object that exceeds heap area and creating large number of objects that exceed the max allowed objects.	**FOS>** run tshr3 3000
3	**`smalloc, createSharedObject, sget & getSharedObject`** ***tst_sharing_2master.c (tshr2):*** It tests the request for sharing object. It validates the returned addresses & the number of allocated frames. It also checks the mem. access (read & write) of the retrieved shared objects with different read/write permissions.	**FOS>** run tshr2 3000
4	**`smalloc, createSharedObject, sget & getSharedObject, malloc & free`** ***tst_custom_fit_3.c (tcf3):*** tests the **custom fit strategy** by requesting **normal and shared** allocations that always fit in one of the free segments. All requests should be granted.	**FOS>** run tcf3 3000

TEST: Shared Memory

Module	Function	Diff.	Testing	Notes
SHARED MEMORY	alloc_share()	L1	**FOS>** run tshr1 3000 **FOS>** run tshr3 3000	
	smalloc()	L2		
	create_shared_object()	L2		
	sget()	L2	**FOS>** run tshr2 3000	Depend on smalloc
	get_shared_object()	L2		
	Custom Fit Cases		**FOS>** run tcf3 3000	Depend on smalloc, malloc & free

"test [TEST NAME] completed. Evaluation = …%"

To ensure the test success, a congratulations message like this **MUST appear without any ERROR messages or PANICs**.

TEST: CPU Scheduling

INDIVIDUAL MODULE#4 – TEST

"test [TEST NAME] completed. Evaluation = …%"
To ensure the test success, a congratulations message like this **MUST appear without any ERROR messages or PANICs**.

TEST: CPU Scheduling

- Run each test independently in a **FRESH SEPARATE RUN**. Loc.: `kern/Tests/test_scheduler.c`

```
test_priorityRR_0
test_priorityRR_1
test_priorityRR_2
```

#	Description	Tests…	Check…	Test Command	Time

1	PRIORITY EFFECT: 5 instances from (fib 35) with priority values (0,2,4,6,8) & WS 500. 1st four instances with fib of 35 & last instance fib 8 (priorities directly set by **kernel**)	• Initialize
		• Schedule Next Process
			check order of all instances	`FOS> schedPRIRR 10 40 2000`
				`FOS> tst priorityRR 0`
				`FOS> tst priorityRR 0`
					< 100sec

2	PRIORITY EFFECT: 4 instances from (fib 35). 3 with priority values (0, 4, 8) & one master with priority 4 that create 2 instances with priority 2 & 10. WS 500 All with fib of 35. (priority is changed by **system call**)	• Initialize
		• Schedule Next Process
		• Syscall: set pri.
			check order of all instances	`FOS> schedPRIRR 10 40 2000`
				`FOS> tst priorityRR 1`
				`FOS> tst priorityRR 1`
					< 100sec

3	PRIORITY PROMOTION: 8 instance from (fib 35) with priority values (0,2,4,6) [2 per each] + 8 instances from (fib_small 8) with priority values (0,1,2,3,4,5,6,7) [1 per each]. WS 500 (priorities directly set by **kernel**)	• Initialize
		• Schedule Next Process,
		• Timer Tick Handler (promotion)
			1. ALL 8 instances from fib_small should be finished first (in strict order)
			2. other 8 instances from fib, each 2 instances should be finished together according to the priority order
				`FOS> schedPRIRR 10 10 200`
				`FOS> tst priorityRR 2`
				`FOS> tst priorityRR 2`
					< 150sec

TEST: Kernel Protection

INDIVIDUAL MODULE#5 – TEST

"Congratulations!! test [TEST NAME] is completed successfully"
To ensure the test success, a congratulations message like this **MUST appear without any ERROR messages or PANICs**.

TEST: Kernel Protection

- Run each test independently in a **FRESH SEPARATE RUN**.
- The time limit of each individual test: **max of 30 sec / each**

Module	Function	Diff.	Testing	Notes	Test Files
CHANNEL	Sleep & Wakeup One	L2	`FOS> run tst_chan_one 500`	`# slaves = 7`	User/ tst_chan_one_master tst_chan_one_slave
	Sleep & Wakeup All	L1	`FOS> run tst_chan_all 500`	`# slaves = 7`	User/ tst_chan_all_master tst_chan_all_slave
SLEEP LOCK	Acquire & Release	L1	`FOS> run tst_sleeplock 500`	`# slaves = 7`	User/ tst_sleeplock_master tst_sleeplock_slave
KERN SEMAPHORE	Wait & Signal: dependencies & critical section	L1	`FOS> run tst_ksem1 500`	Depends on Channel	User/ tst_ksemaphore_1master tst_ksemaphore_1slave
		L1	`FOS> run tst_ksem2 500`	`# customers = 100` `Shop capacity = 30` Depends on Channel	User/ tst_ksemaphore_2master tst_ksemaphore_2slave

TEST: Fault Handler III

INDIVIDUAL MODULE#6 – TEST

TEST: Fault Handler III – **LRU**

- The time limit of each individual test: **max of 5 sec / each**
- **MAKE SURE** to switch the replacement policy before start testing: **`FOS> lru 1`**

#	Test Functionality	Test
1	***tst_page_replacement_alloc.c (tpr1):*** tests allocation in memory and page file after page replacement.	➢ **FOS>** lru 1 ➢ **FOS>** run tpr1 11
2	***tst_page_replacement_stack.c (tpr2):*** tests page replacement of **stack** (i.e. new pages) (creating, modifying and reading them)	➢ **FOS>** lru 1 ➢ **FOS>** run tpr2 6
3	***tst_page_replacement_lru.c (tlru):*** tests page replacement by **LRU** algorithm (it checks working set before & after replacements)	➢ **FOS>** lru 1 ➢ **FOS>** run tlru 11

TEST: Fault Handler III – **MOD. CLOCK**

- The time limit of each individual test: **max of 5 sec / each**
- **MAKE SURE** to switch the replacement policy before start testing: **`FOS> modclock`**

#	Test Functionality	Test
1	***tst_page_replacement_alloc.c (tpr1):*** tests allocation in memory and page file after page replacement.	➢ **FOS>** modclock ➢ **FOS>** run tpr1 11
2	***tst_page_replacement_stack.c (tpr2):*** tests page replacement of **stack** (i.e. new pages) (creating, modifying and reading them)	➢ **FOS>** modclock ➢ **FOS>** run tpr2 6
3	***tst_page_replacement_clock_1.c (tmodclk1):*** tests page replacement by **Modified Clock** algorithm (it checks working set before & after replacements)	➢ **FOS>** modclock ➢ **FOS>** run tmodclk1 11
4	***tst_page_replacement_clock_2.c (tmodclk2):*** tests page replacement by **Modified Clock** algorithm (check maintaining the correct FIFO order, during the **placement,** after removing some pages from the working set)	➢ **FOS>** modclock ➢ **FOS>** run tmodclk2 11

TEST: Fault Handler III

Module	Function	Diff.	Testing	Notes
LRU Replac.	Update Counters	L2	`FOS> run tpr1 11`	
			`FOS> run tpr2 6`	
	Algorithm itself	L1	`FOS> run tlru 11`	Depend on Update Counters
MODIFIED CLOCK Replac.	Allocation in RAM & Disk	L3	`FOS> run tpr1 11`	
			`FOS> run tpr2 6`	
	Algorithm itself		`FOS> run tmodclk1 11`	Handled in replacement
			`FOS> run tmodclk2 11`	Handled in placement

"Congratulations!! test [TEST NAME] is completed successfully"

To ensure the test success, a congratulations message like this **MUST appear without any ERROR messages or PANICs**.

GOOD LUCK isA

☺ Enjoy **developing** your **own OS** ☺