Paris Sfikouris - 2671387
Nikolas Ioannou - 2671383

Lab7: Under Pressure Report

Firstly we appended to our page info struct:

- **struct list lru_node** - The LRU list node.
- **struct task *task** - The task that uses the page.
- **void *fault_addr** - The faulting address.
- **uint8_t r_bit** - The second chance bit.

Also we created a **struct list lru_list** which will hold the pages that are candidates for a swap out along with an **int lru_len** variable which will keep the count of pages in list indicating when there is a memory pressure.

For managing the swap in/out of pages we created a **struct sectors:**

```
struct sectors {
        void *pva;      - The faulting address. If NULL sector is available.
        uint64_t id;    - The index in the disk.
};
```

For the implementation we created the following procedures:

- **void sector_init(void)**
- **uint64_t sector_hashfunc(uint64_t)**
- **uint64_t get_free_sector_index(void*)**
- **void check_mem_pressure(void)**
- **void mem_pressure(void)**
- **int check_disk(void*)**
- **int read_sector(int, struct vma *)**
- **struct task *kernel_task_alloc()**

void sector_init(void):

Initializes the sectors to be used for the swap in/out. It calculates how many pages we need to store our sectors and allocates them while also initializing each sector.

uint64_t sector_hashfunc(uint64_t)

Calculates which sector the page is stored according to its faulting address.

uint64_t get_free_sector_index(void*)

Using the hash value from **sector_hashfunc** as index it returns it if it is free. If the sector indicated by index is used it will increment index until the first free sector where **pva == NULL** is found.(-1 otherwise)

void check_mem_pressure(void)

Using the **lru_len** variable it will check if the **lru_list** contains more than 70% of total pages(**npages**).

int check_disk(void*)

This function will check the disk if the page fault is for a page that was stored to the disk during a memory pressure. If so, using the hash value from **sector_hashfunc** as index it returns the sector if it points to the page needed. (-1 otherwise)

struct task *kernel_task_alloc()

It creates a TYPE_KERNEL_TASK task with **mem_pressure function** to be executed. Whenever **check_mem_pressure** indicates that there is a memory pressure the task is started to store pages to the disk.

void mem_pressure(void)

When memory pressure is detected this function is invoked. It swaps out 30% of all pages. It removes the page node from the lru list and then finds a new sector using the faulting address. After the sector is prepared it is written on the disk and the page is invalidated and the PTE is unmapped.

int read_sector(int, struct vma *)

After **check_disk** is invoked, if the page is found in the sectors hash table we then proceed with this function to retrieve it. Firstly we read the page with disk_read and then proceed to push it back to lru_list and then insert it to the task pml4 address space. Lastly we free the sector for later use.

The basic idea is that whenever the system detects memory pressure we assign the kernel task to a cpu to swap out pages from the lru_list. Our first attempt was to dedicate a certain cpu(the last) to handle the memory pressure by invoking the function.

In the main function we make a call to sector_init and initialize lru_list. The last cpu creates the kernel task and halts until memory pressure is detected.

Swap In:
When handling the page faults if the faulting page is not present before populating the region we check if it is stored on the disk and if so, we invoke read_sector where the swap in will happen.

Swap out:
We invoke check_mem_pressure whenever the system allocates pages (e.g do_populate , COW). If memory pressure is detected the last cpu is started and mem_pressure function is invoked where the swap out will happen.

Limitations:

The data structure used for sectors is a hash table with lists for dealing with collisions. This can be improved because in case of many collisions the lookup in hash tables begins to slow down.

There is a page fault occurring after testing our algorithm. It has to do with the vmas of the task. We could not find the bug causing the user fault and because of this the kernel does block while waiting for disk operations.

The main reason we did not manage to fix the bugs is because we did not have time (we both took also the CNS exam). We know it's not an excuse.