# Q2 2023-2024 SOA Project

The professor Baka Baka likes the potential of our Zeos system and wants to implement some kind of video game. He would like to design a Pacman(™) like game in which different objects (ghosts) are displayed on the screen and you control your player with the keyboard.  But, he realizes that Zeos currently lacks the required support to do it effectively and wants you to improve it.

## Videogame architecture

There will be two running processes to manage the videogame. The main process of the videogame is responsible for reading the keyboard. Another process will manage the movement of enemies and other elements, the placement of the spaceship and finally draw the resulting frame on the screen. To transfer information between one process and the other you will use a shared memory region.

Notice that in ZeOS, a screen can be viewed as a matrix of 80 columns by 25 rows. Review the implementation of printc to understand how the video memory is accessed.

Finally, some statistics about the performance of the videogame will be desirable. In the topmost line of the screen, you must show the frames per second.

## Keyboard support

Therefore we have to add a new system call:

```
int read(char* b, int maxchars);
```

It allows a user process to obtain *maxchars* keys pressed and store it in ´b´. This is a **non-blocking** function, it will return the number of characters that have been read (0 if none). The keyboard device support implementation has to store the keystrokes in a circular buffer.

## Screen support

Current screen support just writes characters sequentially. Add a new system call:

```
int gotoxy(int x, int y)
```

That changes the current position of the cursor to the x$^{th}$ column and y$^{th}$ row of the screen.

Add a new system call:

```
    int set_color(int fg, int bg)
```

This system call changes the current foreground and background colors of the text, and affects the following write operations.

## Shared Memory support

In order to share memory between the processes a shared memory support is required. We want a mechanism to share a maximum of 10 physical pages, numbered 0 to 9. Each process must be able to map one (or more) of these pages into its address space. The following system calls must be implemented:

```
    void* shmat(int id, void* addr)
```

Attach the shared region 'ID' to the logical address space of the current process at page aligned address 'addr'. If this is not possible, find an empty region. If 'addr' is NULL, find an empty address. If 'addr' is not page aligned, return an error.

```
    int shmdt(void* addr)
```

Remove the shared region at logical address 'addr' from current address space.

```
    int shmrm(int id)
```

Mark the shared page 'ID' to be cleared (rewrite all its content to zero) as soon as the last process removes the last mapping.

## Process optimization (COW)

The current method to create processes has quite a lot of overhead, as it needs to reserve and copy a lot of frames that may not be used at all. Therefore we want to improve the creation of processes by using a Copy On Write (COW) mechanism, so the frames will be reserved and copied only in the case that someone tries to write them.
Therefore, change the fork system call to just copy the page table from the parent, and mark all user pages with write permissions as read-only (these will be the Copy-on-write pages). Modify the physical frames bitmap to track how many processes share each page. Change also the page-fault exception handler to detect if the fault is due to a write access to a COW page, and then do the missing reservation, mapping and copying from the original fork.

## Milestones

1. (1 point) Keyboard interrupt stores keys in a circular buffer.
2. (1 point) Functional *read* feature.
3. (1 point) Functional *gotoxy* and *set_color* features.
4. (1 point) Functional *shmat* feature.
5. (2 points) Functional *shmdt* and *shmrm* features.
6. (1 point) Performance statistics.
7. (2 points) Functional COW support.
8. (1 point) Functional game using different implemented features.