Q1 2022-2023 SO2 Project

The respectable professor Baka Baka wants to implement a video game on our Zeos system. He would like to design an Operation Wolf(™) like game in which objects are displayed on the screen and you can control a crosshair in any direction with the keyboard to destroy them, but he realizes that Zeos currently does not have the necessary device access support (keyboard and screen).

Keyboard support

Therefore we have to add a new system call:

```
int get_key(char* c, int block);
```

It allows a user process to obtain one of the keys pressed and store it in ´c´. The ‘block’ parameter indicates the behavior of the call if there are no keys available: blocking (1) or non-blocking (0). In the blocking case, the process must block itself until someone presses a new key, at which point the process is unblocked, returning immediately the received key. In the non-blocking case, it will return an error immediately. If different processes execute this call, the keys must be served in strict sequential order (FIFO). The keyboard device support implementation has to store the keystrokes in a circular buffer.

Screen support

For simplicity we will support the text mode of the screen which consists of a matrix of 25 rows by 80 columns containing a character and its color (*char screen[25][80][2]*). For writing on the screen, ZeOS will call a user callback function every clock interrupt. The header of that user function is:

```
void screen_callback(char *screen_buffer);
```

The *screen_buffer* parameter is the address of a user buffer allocated by ZeOS in which the contents of the screen must be written. So, *screen_buffer* can be seen as a *char screen_buffer[25][80][2]*.

To program this user callback function, i.e., to make ZeOS know what function must be called in every clock interrupt, a new system call must be provided:

```
int set_screen_callback(void (*callback_function)(char*));
```

This system call will return 0 if no callback was previously defined and the address of the *callback_function* is correct. In any other case, it will return -1 setting errno to EINVAL.

Memory support

Due to the dynamic nature of a game, we will need to create different objects to represent the elements in the game. In particular, to avoid flickering, the game must implement a double buffering technique. This technique consists of having one buffer, the primary buffer, with the contents that must be displayed in the screen and one (or more buffers), secondary buffers, in which the new frames are currently being drawn. These secondary buffers become the primary buffer when its content is ready to be displayed. Once this happens, the old primary buffer becomes a secondary buffer.
Due to the this, we will need to allocate dynamically those buffers, therefore we will create specific system calls:

```
char* get_big()
int   free_big(char *s)
```

These calls allocate (and destroy) a memory region in the user process with 4096 bytes . These memory regions will be inherited by child processes.

Improve Memory management

Taking a look at the memory objects that we are creating (screens, enemies and crosshair) we observe that if we use a *get_big* for small elements we will waste a lot of physical resources. Therefore, implement a user level memory manager using the previous kernel-supported memory manager that minimizes the wasted space by allowing to allocate fixed size small objects (32 bytes).

```
char* get_small()
int   free_small(char *s)
```

These calls allocate (and destroy) a memory region in the user process with 32 bytes . These memory regions will be inherited by child processes.

Add support to mouse device

It would be nice to move the crosshair using the mouse device. Add the required support. You may find information on how to implement mouse support in the following link:

https://wiki.osdev.org/Mouse

Milestones

1. (1 point) Keyboard support stores keys in a circular buffer.
2. (1 point) Functional *get_key* system_call.
3. (1 point) Functional *get_big* and *free_big* features.

4. (3 points) Functional *set_screen_callback* system_call.
5. (1 point) Functional game testing the previous features.
6. (2 points) Functional *get_small* and *free_small* features.
7. (1 point) Functional game using different cases to check memory features.
8. [(1 point) Mouse challenge]