Q2 2022 SOA Project

The respectable professor Baka Baka wants to implement a video game on our Zeos system. He would like to design a snake-like game in which a snake must eat food that appears randomly in the screen avoiding collisions with itself and the borders of the screen. The more food the snake eats, the larger it becomes. Unfortunately, Zeos currently does not provide the necessary device access support (keyboard and screen).

Keyboard support

Therefore, we must add a new system call:

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

That allows a user process to obtain one of the keys pressed in ´c´. If there are no keys available, the system call must return an error. If different processes execute this call, the keys must be served in strict sequential order (FIFO). The keyboard device support implementation must store the keystrokes in a circular buffer. If this circular buffer becomes full, no more keystrokes will be stored until there is some empty space.

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 programme 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(char *(*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.

Since the refresh of the screen is guided by the clock interrupt, at most, 18 frames per second should be drawn.

If the user callback has not been programmed, i.e. no set_screen_callback syscall has been called, the clock interrupt will not execute any callback.

Finally, the user callback is set by process. Several processes, at the same time, can have their own user callback. Only the callback of the current process will be invoked.

Memory support

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 is currently displayed in the screen and one (or more buffers), secondary buffers, in which the new frames are 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_screen()
int remove_screen(char *s)
```

These calls allocate (and destroy) a memory region to the user process with 25x80x2 bytes to contain a screen content.

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_screen* feature.
4. (1 point) Functional *remove_screen* feature.
5. (1 point) Functional game testing the previous features.
6. (1 point) Functional *set_screen_callback* system_call.
7. (2 points) Functional FPS display.
8. (2 points) Functional game using all the features.