Q1 2025-2026 SOA Project

The professor Baka Baka likes the potential of our Zeos system and wants to implement some kind of video game like the Dig Dug(™) game in which different objects 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 threads: one that will be executing all the time-dependent tasks such as controlling enemy movements. Another thread will be responsible of drawing the videogame screen.

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.

Thread support

To implement the architecture of the video game, as defined at the beginning, thread support must be implemented. Therefore, we want to add the system call:

```
int ThreadCreate(void (*function)(void* arg), void* parameter);
```

This creates a new thread that will execute function *'function'* passing it the parameter '*parameter'* as in '*function(parameter)*'.

When a thread is created, its initial user stack size is one page. To prevent this stack from being too small, a mechanism to dynamically make the user stack grow must be provided. For that, user stacks are created with memory gaps between them. The operating system must check in a page fault if the exception is because a memory gap before a user stack has been accessed. In that case, the user stack must grow when recovering the exception. This is also valid for the first thread of any process.

The current thread and its stack must be freed after calling the system call:

```
void ThreadExit(void);
```

This call must be called to finish the current thread. Returning from a thread without calling this function must crash the system. If there is only one thread in the process, the process must be finished.

Create a thread wrapper to ensure that threadExit will always be called.

The fork system call must be modified to create a process with only the current thread (and its associated resources).

Keyboard support

A new system call is required:

```
int KeyboardEvent(void (*func)(char key, int pressed));
```

This system call programs a function that will be called every time a keyboard event (pressing or releasing a key) is triggered. This *func* will receive, as parameters, the scancode of the key that has triggered the event and an integer to know whether the event is a key pressed or released. Whenever a key is pressed or released, the operating system immediately executes *func* inside the context of the current thread. To resume the execution of the current thread, *int 0x2b* must be executed. But this must be transparent to the user, so, a wrap function must be provided in libc that executes *func* and *int 0x2b* (like a thread wrapper).

To avoid problems with a malformed user stack, an auxiliar user stack must be created for executing *func*.

When *int 0x2b* is executed somewhere in the code outside of the keyboard support, it will do nothing.

Any system call executed inside *func* must return the error EINPROGRESS.

Screen support

Access the screen buffer through the system call write using file descriptor number 10. Review the printc function to know how to write to the video memory at address 0xb8000. A valid call to write system call to show a the whole screen should be:

```
write (10, frame, 80 * 25 * 2);
```

in which *frame* is a pointer to an 80 x 25 matrix having 2 bytes (color, character) per position that store the frame to draw.

The implementation has to be fast enough to ensure that write can be executed, at least, 30 times per second in the case that only one process with one thread are present in the system.

Time support

Implement the following system call:

```
int WaitForTick();
```

that will block the current thread until a clock interruption rises. More than one thread can be blocked simultaneously.

Final remarks

The previous system calls and features may need modifications to some other parts of the operating system, and some decisions must be taken in order to implement them since not all details are provided.

It is your responsibility to take those decisions and make those modifications.

Milestones

For every milestone you must deliver:

-The source code with elaborate comments to show that you fully understand what you have done and that you have not used any AI-based tool to write the code.

-A test benchmark that shows the correct behavior of your implementation covering all the cases described in this document

-A text file with a short description of all the decisions taken during the implementation of the feature (if any) and the modifications performed in other parts of the operating system.

Milestones:

1. (3 points) Implement the thread support.
2. (3 points) Implement the keyboard support
3. (1 point) Implement the screen support
4. (1 point) Implement the time support
5. (2 points) Implement a colorful and enjoyable video game