Assignment 4: Processes

Robin Kruit - 0936014

April 23, 2019

1 Context

Every modern computer is capable of carrying out multiple tasks at the same time. There are two different approaches to multitasking, commonly called preemptive multitasking and cooperative multitasking. With preemptive multitasking a process scheduler is in complete control of which program runs on the CPU at any given time. Changing which process is running, or context switching, is done by storing the CPU registers, program counter and possibly other data specific to each operating system.

Cooperative multitasking, while no longer used in regular computers, is still widely used in embedded systems. Here each process voluntarily gives up control back to the CPU. This simplifies the implementation of programs due to the lack of unexpected interruptions. For this assignment you will implement this form of multitasking.

2 Assignment

In the kernel's src folder you will find a source file called processes and in the src/include folder the corresponding header. You are expected to implement each function in the files and demonstrate a working scheduler. Deviating from the header is allowed and even encouraged when done for good reasons. Multiple processes that use multiple variables and functions should run simultaneously for the scheduler to be considered working. A working knowledge of variables, structs, functions and in particular function pointers are recommended. The functions that you will need to implement are summarized below as well.

```
int add_process(char *name, int (*func)(void));
int suspend_process(int id);
int resume_process(int id);
int kill_process(int id);
void do_round(void);
```

In the header you will also find the following struct:

```
struct process {
char *name;
int id;
char state;
int (*func)(void);
};
```

- *add_process* should receive the name of the proccess for diagnostic purposes and a function pointer to the function that should be called when the process is to receive cpu time. It should return the process id used in the other functions.
- *suspend_process* should set the process state so that it will not be run when the scheduler does a round.
- *resume_process* should set the process state so that it will be run when the scheduler does a round.
- *kill_process* should mark a process for removal so the process can do cleanup before being removed.
- *do_round* should check each process' state and act appropiately. Processes can be killed in here, after which the process table should be packed.