Styresystemer og Multiprogrammering

G-opgave #2

Jenny-Margrethe Vej
(rwj935@alumni.ku.dk)
Klaes Bo Rasmussen
(twb822@alumni.ku.dk)

Datalogisk Institut, Københavns Universitet
Blok 3 - 2013

Types and Functions for User Processes in Buenos

1. We defined the required datastructure `process\_control\_block\_t` within process.h. For now it only contains what we belive to be the required fields to complete the given task.

We can save it state, given by the `enum process\_state\_t`.

It keeps a return value, which is used in `process\_join`, `process\_finish and italso` acts as the exit code for `syscall\_exit`.

It contains a small buffer which is used for saving the name of the executeable to be run, currently programs can have a name of maximum 32 chars.

Last, and most importantly, it keeps track of which process it is. This process id is used any time a process is called upon for any functionality.

2. We have implemented all the given functions in the second task inside the file `process.c`. We will not say much about this, mostly explain the problem we came to when we was able to test the code.

So, first, everything does compile. But, I have added the "hw" program and the "exec" programs from the "tests" directory, to the disk, and running exec presents a quite big problem. Looking at the exec.c file, we figured that the program would run the "hw" program by default. But when exec is run, the text that is thrown with it, which is supposed to be the name of the program to be run, is instead "[arkimedes]exec" itself. We were unable to fix this, and thus cannot test the rest of the functionality, suspected problem is the way we edited `start\_process` to account for the new data structure. But this will be looked in to and fixed asap!

System Calls for User Process Control in Buenos

There is not much in this task, but maybe something have gone wrong here too since obviously running executables does not work. Otherwise, the system calls merely provide a shell for calling the functions defined in the first part of the task. All are defined in `syscall.c`, where the functions are defined first, and then the cases in the actual syscall.