

# Operating systems and multiprogramming

## G-assignment 2

Alexander Worm Olsen - bdj816

Allan Martin Nielsen - jcl187

Troels Thomsen - qvw203

February 24 2014

Department of Computer Science  
University of Copenhagen

## Types and functions for userland processes

First of we were to define a data structure to represent a userland process. To do this we used the already implemented datastructure, *process\_control\_block\_t*, in *process.h*, and expanded this with a state and id. Later on we found out, that we were in need of the name of the process because we changed the function *process\_start* (see below) and the exitcode which was to be returned when calling join.

The possible states of a function is defined in *process.h* as well, these are chosen with inspiration from the book and the assignment description.

When the data structure was defined we moved on to the implementation of the helper functions. These are found in *process.c* and include a *process\_spawn*, *process\_join*, *process\_finish* and *process\_init*.

The *process\_init* is straightforward, we initialise a process table with all its entries set to *process\_state\_dead*. When we are to add processes to this table we use the *process\_spawn*. This function calls yet another helper function; *add\_proc*. In our *add\_proc* we run through the process table looking for an element with a state of dead. If this exist we insert the new element, otherwise our process table is full and we therefore add it to a sleep queue, which is awoken when processes are joined together.

When the new process is inserted we make a call to *process\_start* within the spawn function. This is called with the process' id, and in order to change the already implemented *process\_start* function we therefore had to add the executable/process name to our data structure and fetch this in *process\_start*.

The *process\_finish* is also quite straight forward, we use the already implemented function, *thread\_get\_current\_thread\_entry* to find the appropriate process to be finished and afterwards we change the state to zombie and invoke wake to our sleep queue.

The *process\_join* helper function is inspired by the section of roadmap to buenos explaining the sleep queue, and it goes through the implementation of this step by step.

Please note that we've also changed the *main.c* in the init folder by calling *process\_init*. This we've have chosen to do, because the process table has to be initialised before any user process calls, and this was possible in *main.c*.

## System calls for user-process control

Our implementation of the system calls join, exec and exit is quite straight forward. The implementations are found in the kernel by *exec.c*, *exec.h*, *join.c* etc. and in the

file *syscall.c* in the folder `proc`.

## Tests

In order to test the functionality of our system calls and related user processes. We've used the hand-out tests *exec.c* and *hw.c*. These test files can be used in order to see if the functionality of one userland process invoking another userland process works. In this way `spawn`, `finish`, and `join` is required, and therefore the two files allows us to make sure all of our implementations works.

We have chosen not to make a test file for each possible error, e.g. a negative return value of `join` on errors etc.