Operativsystemer og multiprogrammering
Handin 2

Malte Stær Nissen and
Jacob Daniel Kirstejn Hansen

February 21, 2011

Contents

1 Overall assumptions

We've made the following assumptions that our implementation of the different functions depend on:

- We've chosen to make a static sized bound of 32, on the number of processes we can have running at the same time.
- We've agreed to run a 1-to-1 strategy between user-level treads and kernel-level threads.
- We're representing alive and dead processes as enums instead of integers, so we didnt́ have too many integers in play to confuse. Free process slots are marked as free with an enum as well
- Every time we change something in the process table, we have made sure to disable interrupts and acquired the needed spinlock for the process table.

We choose to create a static array to represent our process table. The reason is that the BUENOS kernel doesn't have a heap implemented, therefore we can't dynamically allocate more memory for our array. Furthermore it's simpler to work on a static array, because we want to save it across system calls and processes.

By using a 1-to-1 strategy, we just create a thread aswell whenever we wish to initialize a new process, and associate it with the process. This however isn't the case when we start up the first process.

2 List of changes

- initmain.c: Initialization of process table added to init function and function to start and link the process to the first thread implemented.
- kernelthread.h & kernelthread.c: Function for getting a thread entry added
- procprocess.c procprocess.h: Processes implemented
- procsyscall.c: System calls implemented using process functions
- kernelconfig.h: Boundary on max number of processes defined

3 Syncronization

We need to make sure, that our process table doesn't get corrupted when performing changes on it. This is made sure by use of syncronization principles. Whenever we need to perform changes on the process table we start

out by disabling interrupts. This is followed by an acquisition of a spinlock, which is shared by all the functions, that performs changes on the process table. After having initiated these precautions we can alternate the process table according to our wishes without having to fear that some other process or threads perform operations on the process table before we're done. When we are done performing the desired changes to the process table, we start out by releasing the spinlock and finish off by setting back the interrupt mask to whatever it was before we disabled it.