Operating systems and multiprogramming

G-assignment 1

Alexander Worm Olsen - bdj816

Allan Martin Nielsen - jcl187

Troels Thompsen - qvw203

February 17 2014

Departmen of Computer Science
University of Copenhagen

A space-efficient doubly-linked list

We have implemented a doubly-linked list in the C programming language, which uses bitwise XOR operations to save memory. We have implemented the list operations using standard C libraries. The code includes a few comments to improve reading, but should otherwise be straight forward. The program dlisttest.c demonstrates the implementation.

Buenos system calls for basic I/O

We were asked to implement the functionality of the two system calls; read and write. In order to do this several changes to Buenos had to be made, including the kernel and procedures. In the folder *proc* the file *syscall.c* has been changed in order to implement the two system calls. These system calls vary from the one already implemented, halt, because read and write expects function arguments. Therefore we had to get these from the mips registers a1, a2 and a3, and pass them as function arguments to the functions.

In the kernel of Buenos we have included *read.c, write.c, read.h* & *write.h*. The header files are inspired from the one of halt, while the actual code for the two system calls are inspired by the testconsole from *main.c*. The write function is straight forward using gcd to write from the buffer and into the console. We have included several kernel asserts in order to make sure no unexpected errors occur.

Our implementation of read is almost as straight forward as the write. Only here we have made use of a while loop in order to allow several character click for the user. This while loop is controlled by the max byte length, which is given as input. During our while loop we increment our buffer which prevent previous input to be overwritten.

Test of read and write

In our buenos archive we have included a *readwrite.c* test file. This file tests the functionality of both read and write. While running the test the user is supposed to enter 9 digits, which will be read by our syscall_read() and put into the buffer. Afterwards the syscall_write() will write from the buffer what characters the user gave as input.

At last a new buffer is initialised and characters are stored in this automatically which is afterwards written to the console.