– Lab1 –
Introduction

Preparing for planetary lab by encapsulating (creating wrappers) system call.

1 Purpose

Get a sense of how Linux functions, i.e. the programmer's interface over Windows. Get a feeling for thread and process concepts, mutexes, as well as communication via mail slots. Preparation for the planet lab by encapsulating system calls (make wrappers).

2 Content

- Linux API
- Creating threads
- Critical sections
- Mail Slot Management

3 Preparation and Tips

- Get an IDE or get familiar with compiling projects via GCC (gcc main.c –o outfile)
- You will have to use the –lpthread and –lrt flags in order to compile files with mailslots and pthreads
- Get familiar with how to use the Linux manual webpage
- Check wrapper.h to see what system call you need to encapsulate! You are not allowed to modify this file, i.e. the interface should not be changed!
- wrapper.c contains all of the skeletons.
- Create file intro.c and do the assignments in this file.
- Fill out the wrappers directly (in wrapper.c) and use it in this and the next lab.

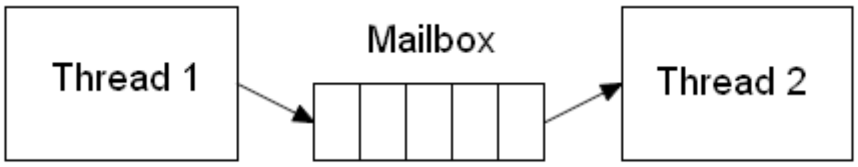
4 Demonstration

You only need to demonstrate 4b)! Just be aware that wrapper.c will be used in later labs.

5 Assignments

1. You must run a process that writes "Hello world!" 10 times on screen with a delay of 1 second between each. The process will terminate after this.
2. Now create another thread which writes "Hello Moon" on the screen forever. There should be a delay of 0.2 seconds between each printout. Create a new thread before the program prints "Hello world!" ten times. Why does ‘hello moon’ thread terminate?
   Try to send a parameter to the new thread. If you have not done so, code the wrapper in ***threadCreate*** in wrapper.c.
3. **a)** Change the program from task 2, so that 10 "Hello world!" is printed, then 10 "Hello moon ", then this is repeated forever. (Still using the two threads)
   b) Can you guarantee that your program will behave like this under all conditions? If not, modify the program so that it always does. Hint: Check out critical sections.
4. **a)** Create a process that contains exactly two threads. Users enter text in one thread, which must connect to a mailbox created by the other thread. Then the other

thread reads from the mailbox and print received text on the screen. Systems should keep running until the string ‘**END**’ is given.



b) Now do the same thing but send a **struct** instead of a string. (The struct can contain any type of data. In this lab, you can simply put only one string in the struct.)
c) If you have not done so, fill in the wrappers in the functions in wrapper.c.