Inspecting XINU's Run-time Environment

Problem 3.1

Where in system/ is the source code of nulluser() defined?

In initialize.c

Where in the XINU source code is this ancestor of all processes made up/named and what name (i.e., PID) does it give itself?

In sysinit() of initialize.c
The PID of the ancestor, NULLPROC, is 0, defined in process.h. The name of the ancestor is prnull.

Does nulluser() ever return after being called by the code in start.S?

No. It is in a infinite while loop.

What does the ancestor process running the function nulluser() do for the rest of its existence?

It guarantee that the CPU has something to run when no other process is ready to execute.

What function is called after nulluser() in start.S? Find the function's source code and determine what it does.

halt(). The function halt() is doing nothing forever (source code is in intr.S), and it makes processor halt.

What happens if you remove this function call from start.S? Does XINU run as before? Explain what you find.

Nothing happen. The XINU runs as before. It is because the nulluser() never returns. The halt is used just in case nulluser() returns, and it halt the processor.

Problem 3.2

In XINU what happens after a new process is created using create()? What happens in Linux after a new process is created using fork()? Who runs first: parent or child?Try to guess the answer by running test code on the frontend Linux PCs.

The process was created by create() will not run until it is resumed by resume(). In Linux, a new process created by fork() will run concurrently with other processes. After testing in Linux, it is hard to say which will run first seems no priorities are set when fork.

In Linux, an app programmer writes C code, saves it to, say a file app.c, runs gcc to generate binary a.out, then uses the file a.out as command-line input to a shell or by calling execve() directly from a process to execute. How does the same app programmer write/port the C code so that it runs over XINU?

In XINU, an app programmer writes C code, saves it to, say a file app.c, modify /include/ prototypes.h to include app.c. Make clean and make rebuild. After the modification, programmer need to recompile the whole OS, XINU, to enable the new functionality.

What are the disadvantages of writing app code under XINU? What are the advantages?

The disadvantage of writing app code under XINU is that you have to recompile the whole OS to enable the new codes.

The advantage of writing app code under XINU is that you don't need to manually compile your code to use it, everything your wrote is integrated into the OS, all you need to do is decide where to use it in the code.

Problem 3.4

To determine what happens after all processes terminate, including the ancestor process which can be terminated by calling kill() by another process which then terminates itself, inspect the code of kill() and track down what ultimately happens to XINU when no processes remain. Explain what you find.

After all user processes terminate, not including null process, kill() will call xdone() and xdone() will call halt to make processor halt.

If XINU were to be run on a mobile platform such as a smart phone, what change would you make in the software based shutdown procedure described above and why?

In order to run XINU on a mobile platform, we need to add a functionality to kill all user processes and halt, instead of waiting the last user process to be killed and halt. Also, we need to change halt to turn off the power, instead of doing nothing.

Bonus Problem

Added xsh_pcount.c in /shell/, which is a shell command to print the number of processes.