CS 503 Lab 0 Answers

Prashant Ravi — ravi18@purdue.edu

January 23, 2016

1 Part 3.1

1.1 What happens after nulluser returns?

I tried changing the code from call halt to call pause, and to my surprise the code after call nulluser never executes. That's because the nulluser method has a tight while loop at the end to signify a process running when no other process is running, hence the name - "null". In any case, I was interested in including the HLT instruction in some way so that if the processor could go into C1 power saving state in the case that it really is devoid if other non masking interrupts or processes queued up. So, I made the changes at the end of nulluser() where an idle function is invoked and defined as a function in intr.S as a function that would run a while loop that in turn calls the pause function in its scope. This hopefully would result in lower power consumption by the CPU if and when its idling.

1.2 How is creating process in Xinu different from Linux?

In Unix the create method created a process by taking in the arguments, priority, stack size and leaves it in a suspended state returning only the pid of the process. The resume method then takes this pid and unsuspends the process so it starts executing. the create and resume methods return immediately after starting the process allowing the original invoker of these methods to find their execution concurrent.

On the other hand, fork() and execve() are quite different from Xinu's way of creating a process. fork() invokes clone() which creates a exact replica of the calling process in terms of the program code, memory allocation, register states. And both the original and cloned processes are indistinguishable but for the return values. Now, if a new process must be created, it must be from this child process(cloned process) and the execve() is called on the child which overwrites its memory with that of the new program.

Clone() vs Create(): Create takes in the arguments to create a new process with new memory location allocated to the program. Whereas, in copy-on-write systems, the memory is shared between the original process and cloned process, and only when a write is issued to the clone, it would make a copy of the orig and make changes to it, such as when an exec function is called on it, it would allocate new memory and fill it with the new program's memory. In non-copy on write systems the new memory allocated to the clone is an exact replica of the original process. This is the major difference in the create and clone

2 Part 3.3

2.1 New System Call

I made a new system call called, getpname() in the getpname.c file, it desribes the name of the process given the process's pid by looking up the procent data structure.