Lab0 Solution

jain117@purdue.edu

None of the exercises of Lab0 need any modification of the Makefile.

Problem 3.1

1. The nulluser() function idles the CPU by having an infinite while loop at the end of its definition. Tracing the nulluser() definition, the "while(TRUE){}" at the end of the definition is the infinite loop. This is so that there is always one process in the ready queue and nulluser() will always be in the ready state because of this infinite while loop. The alternate way is to use the pause() function in nulluser() instead of the infinite loop. The way this works is: pause() calls enable which enables necessary interrupts and makes the CPU go into Halt state, thus reducing unnecessary CPU cycles. Once an interrupt comes because of an I/O device or a new process is in the ready queue, the CPU is started again and execution of the relevant process begins.

2. In Unix/Linux, on the fork() call the parent process context is copied in memory and a new processId is attached to this copy. This will be the child process. The copy includes all the segments: data, text(code), .bss, heap and stack. Thus, the two processes will have different memory. The child process then uses the execve() call to initiate it's execution and overlay itself with the parent and other processes. In Xinu, on the other hand, the analogous to the fork() call is the create() call which creates the child process by allocating just the stack for the child process. The other segment like data, text(code), .bss(uninitialized) and heap are shared between the parent and the child process. The code segment is shared among all the processes on Xinu. In the case of the data segment, when the child, as a part of execution, does a write on some variable the segment is copied for the child process and the change happens on this copied segment rendering the original ustouched and thus still usuable by the parent. This is called "Copy on Write". The resume() call on this child process is analogous to the execve() call and initiates the execution of the child process by putting it in ready state.

   In comparison to fork(), Xinu's create() is closer to the clone() system call. In the clone() system call, the child process shares the execution context with the caller(parent). Thus parts of the process memory is shared between the parent and the child process similar to Xinu's create().

Bonus Problem 3.4

In the bonus problem, I have created a function called prtprstate() in the new file prtprstate.c which prints the name and the state of the process whose process ID is the argument to this function. The state of the process is identified using the proctab structure in process.h. The different states are:

FREE: The process block is cleared and possibly killed.
SLEEP: The process is sleeping.
READY: The process is ready to be executed and is in the ready queue of the scheduler.

RECV: The process is waiting for message
SUSP: The process is in suspended state
WAIT: The process is in the wait state for a semaphore resource.