Problem 3.1:

1) proctab[] is declared in process.h. It is an extern array of the procent struct.
    a) prstate (uint16) holds the process state.
    b) prprio (pri16) holds the process priority.
    c) prstkbase (char *) holds the beginning address of the run-time stack
    d) prparent (pid32) holds the ID of the creating process
2) :
    a) prstate = 1
    b) prprio = 0
    c) prstkbase = 0xEFDEFFC
    d) prparent = 0
3) :
    a) prstate = 5 (PR_SUSP) initial state is suspended
    b) prprio = 20 (INITPRIO) initial process priority
    c) prstkbase = 0xEFD8FFC
    d) prparent = 0
4) :
    a) prstate = 5 (PR_SUSP) initial state is suspended
    b) prprio = 20 (INITPRIO) initial process priority
    c) prstkbase = 0xEFC8FFC
    d) prparent = 2
5) The startup process invoked by nulluser() is terminated (returned OK) after main is called, and ps only shows the PID of processes that haven't been terminated.

Problem 3.4

- `pushl %ebp` **This pushes the old stack frame pointer onto the stack**
- `movl %esp,%ebp` **This moves the value of the stack pointer into the ebp register.**
- `pushal` **Pushes all register values onto the stack**
- `movl %esp, %eax` **Moves the value of the stack pointer to eax**
- `pushl %eax` **Pushes eax onto the stack**
- `pushl $0` **Pushes 0 onto the stack**

To conclude: _Xint0 saves the values of the registers on the stack, the top two values on the stack are the first two arguments used in trap(), which prints out (dumps) the registers values to the console via the stack pointer. $0 is the first argument, the pushed value of %eax is the second argument. They were pushed in the order.

Problem 3.5

Panic essentially ends all processing in xinu by creating an infinite while loop after displaying the error message. The terminal is basically locked, one cannot type or execute any commands leading the backend machine to lock up until xinu is terminated.

Problem 3.6

After a process returns, the function userret() is called which calls kill(pid). The kill() function lowers the process count, frees the stack space that the process used, and frees the index in the proctab array then returns OK.

Since myapp() does not have an infinite loop, and since main() no longer has an infinite loop the only process (excluding rdsproc) is the null process which has a while(TRUE) loop that does nothing, forever (until xinu is killed by the user).

BONUS:
Fork() primarily creates new processes, clone() creates a new thread, or path of execution. clone() is similar to xinu's create() in the sense that memory space is shared because XINU has shared data, text, bss.