CS354 Lab 1

Siddhesh Patil
patil33@purdue.edu

3.1

a. Data type of proctab[] is **procent** and is located in **process.h** in include.
   Members that define-
   Process state: prstate
   Priority: prprio
   Beginning address of nulluser()'s run-time stack: *prstkbase
   PID of process that created a process: prparent

b. Process state: 1 (Currently running according to the process.h)
   Priority: 0
   Beginning address of stack: 0x0efdcffc
   PID of parent process: 0

c. Process state: 5 (Process is suspended according to the process.h)
   Priority: 200
   Beginning address of stack: 0x0efd8ffc
   PID of parent process: 0

d. Process state: 5 (Process is suspended according to the process.h)
   Priority: 20
   Beginning address of stack: 0x0efc8ffc
   PID of parent process: 2

e. We do not see a difference even when one less process is created with nulluser1(). Main process is called directly by the null process so startup process is never created (thus one less process). The pid of main is 2 when we use nulluser1() but was 3 when it was being called by nulluser() We also see that ppid of main is 0 in case of nulluser1() and 2 in the case in nulluser().

3.2

DONE

3.3

DONE

3.4

pushl %ebp -> the base pointer is pushed onto the stack
movl %esp,%ebp -> base pointer is given the value of the stack pointer
pushal -> push all registers at once
movl %esp, %eax -> move the contents of stack pointer into eax
pushl %eax -> push the contents of eax
pushl $0 -> push interrupt number
jmp Xtrap -> jump to Xtrap

Trap takes interrupt number of datatype int and a saved stack pointer of datatype long as arguments

explain how the code in intr.S manages to communicate the two arguments passed to trap():
The assembly code pushl %eax pushes the contents of eax (which contains the stack pointer) and pushl $0 pushes the interrupt number so when the jump to Xtrap is made in the assembly code, the trap() function in evec.c in invoked and is given the appropriate arguments.

3.5

The information being outputted is: the kind of exception that occurred (divide by 0 in this case), the pid of the process and the name of the process. It then adjusts the stack pointers to get debugging information related to the interrupt and then it shows us the register values during the time the exception occurred.
Panic() first disables the interrupts, prints the message in its argument and then goes in an infinite loop.

3.6

After the process returns with OK, the return address points to the function exit() located in system/exit.c. exit() function then calls the kill() function where the kill command sets the state of the process as PR_FREE. This sequence is required for XINU to terminate the process normally.

BONUS PROBLEM:

Linux's clone(), unlike fork() allows the child process to share parts of its execution context with the parent process such as, virtual address space, table of file descriptors and table of signal handlers. Clone() is also used to implement threads and after the child process is created it starts its execution by calling the function pointed to by its first argument. On the other hand XINU's create() has stack address, thread ID and pointer to the new block and it's execution starts only after resume is called to change the suspended state of the child process to ready state.