Pt2. First, when getpid() is called, execution starts in the user space at user.h, where it is declared as a system call. The call itself is handled in usys.S where SYSCALL(getpid) is a macro that expands into x86 that initiates a syscall by putting its syscall number (18) into %eax. Then the int $64 instruction is read which triggers an interrupt. Now in the kernel space trap.c then catches the interrupt and calls syscall(), defined in syscall.c. Syscall grabs the syscall number from the eax register and assigns it to a variable. This function looks up the syscall number, 18 in this case, from the array of all the syscall numbers and sends the control to the appropriate function. sys_getpid() is implemented in sysproc.c and returns myproc()->pid. myproc() returns a pointer to the PCB of the current running process, represented by the struct proc, which has PID. execution moves back to syscall() in syscall.c, which stores the return value into %eax, then the execution returns to trap.c, resets all the registers back to their original states and gives the return value to getpid(), officially going back to user space.