Part 4:

I created a new entry in the process table called ***void (*cbf)(void)*** that will hold a reference to the callback function if one has been assigned. It is initialized to ***NULL*** in create and gets updated whenever ***cbchildregister()*** gets called. Upon entering cbchildregister.c, we check to make sure that the current process does not already have a callback function assigned (i.e. ***prptr->cfb*** is null), since we are only allowed to assign the callback function once. If one is already assigned (i.e. ***prptr->cfb*** is not null), then we should return ***SYSERR***.

There is also a global variable called ***void (*globalCBF)(void).*** This is initialized to null inside of ***initialize.c***. We then update this ***globalCBF*** inside kill.c upon termination of a process. Inside ***kill.c***, we get the callback function assigned to the parent of the process that was just terminated. We then set ***globalCBF*** to the callback function that we got from the process table of the parent process. This tells us that a callback function should run the next time ***clkdisp.S*** is executed.

Inside clkdisp.S, right before calling iret we check if ***globalCBF*** is null. If it is null, we continue without detour and call iret immediately. If ***globalCBF*** is not null, that means we need to perform a detour and run the callback function before returning to the original return address. We do this by saving the first 3 general registers (***EAX***, ***EBX***, ***ECX***) into their corresponding global variables which were created and initialized in ***initialize.c***. This is done to allow us to preserve the values in those registers. After this, we pop the ***EIP***, ***CS***, ***EFLAGS*** in that order and save them into ***EAX***, ***EBX***, and ***ECX*** respectively. We then push the ***EIP***, ***EFLAGS***, ***CS*** onto the stack using the general registers in where they were saved. We then push the pointer to the callback function onto the stack that was saved in ***globalCBF***. Now the stack has been set correctly, all that is left is to reset ***globalCBF*** to null since we don't want it to continuously run the callback function, just once per termination of a process. Afterwards, we reset the general registers by moving the values that we saved in their corresponding global variables back into the registers. This allows us to mess with the general registers but ensure that they are correctly reset for other functions.

To test this, I just used the example given in the lab handout. We register a callback function for the main process by calling ***cbchildregister(&callbackFunction).*** We then create and resume a child process that does nothing but waste time. It wastes time by doing some unnecessary computations inside of a big for loop. Our expectation is that we run the main function, and when the child process terminates, the callback function immediately runs and then returns to where it left off in main. Upon testing the example, this is exactly what happens. I also test to make sure that changing the call to **xchildwait()** inside of the callback function to be a blocking call instead of non-blocking and confirmed that it still behaves the same.