Today we are going to be getting more of a head start on problem 3 from homework 3. These exercises will outline:

- changes that we need to make to the `task_struct`
- the struct we need to add to `sched.h`
- code to initialize this that we will add to the `copy_process` function in `fork.c`
- A general outline of our `delegate_usr` system call.

1. Modify the `task_struct` to add in a pointer to the head of a linked list that will hold the list of users that we have delegated access to. Insert the following lines at the end of the definition of the `task_struct` in `sched.h`:

   ```c
   #ifdef CONFIG_DELEGATE_USR
   struct list_head *user_list;
   #endif
   ```

   Note: We are surrounding ALL the code we add inside conditional compilation blocks. (#ifdef... #endif) This way the code we add will only be compiled when we have set our configuration to include our new functionality.

2. Add in our struct to `sched.h`. Place it just below the end of the `task_struct` so it is easy to find. This structure will serve as the nodes in our linked list. Each node will contain the uid of a user.

   ```c
   #ifdef CONFIG_DELEGATE_USR
   struct user_delegate{
       uid_t uid;
       struct list_head user_list;
   };
   #endif
   ```

3. We will add code to the `copy_process` function in `fork.c` to initialize the head of our linked list. Placing the code in this function will cause it to be executed every time a new process is created. This means that all processes running on the system will have the list of uid’s initialized.

   Find the `copy_process` function in `fork.c` (line 981). At the top of the function add the code below under the line `int cgroup_callbacks_done = 0;`:
#ifdef CONFIG_DELEGATE_USR
    struct user_delegate *ud;
    ud = kmalloc (sizeof(struct user_delegate), GFP_KERNEL);
#endif

Next, right below the lines (around line 1206)

    /*
     * sigaltstack should be cleared when sharing the same VM
     */
    if ((clone_flags & (CLONE_VM|CLONE_VFORK)) == CLONE_VM)
        p->sas_ss_sp = p->sas_ss_size = 0;

Add in the following lines: (You should be around line 1212)

    #ifdef CONFIG_DELEGATE_USR
    ud->uid = p->real_cred->uid;
    INIT_LIST_HEAD(&ud->user_list);
    p->user_list = &ud->user_list;
    #endif

4. Now that we have added in the functionality we need we can implement our system call. First we need to modify the parameters our system call is taking slightly. Change your function declaration to be this:

    asmlinkage long sys_delegate_usr(char request_type, pid_t pid, uid_t uid, unsigned int *addr, int size)

    Note: We have changed the system call so it takes in a type of uid_t which is the proper type for uid's. We have also changed the addr pointer to be of type unsigned int because this is the size of the uid_t type.

    You will also have to updates the syscall.h file we modified in the last exercises to reflect this change.

5. Our system call will need three helper functions to add users, delete users, and list the users. Create function declarations for these three helper functions. Note: The add function will use the pid and the uid parameters. The delete function will use the pid and the uid parameters. The list function will use the pid, addr and size parameters.
6. For the add user function take a look at the code we added to fork.c. This should give you a good idea of what you need to do for this function. Take a look at the list.h header file. http://lxr.linux.no/linux+v3.8.2/include/linux/list.h There is a function called list_add that you will need to use here.

7. For the delete user function and the list user function take a look at the macro in list.h called list_for_each_entry. This is a nice way to iterate through the linked list. Note that because of the way this macro works (take a look at how it is defined in list.h) it will not print the head of the list which in our case contains the uid of the user that created the process. To print the head of the list you will have to find another macro to help. Make sure you do NOT delete the head of the list in your delete user function. This means you should have some way to check and make sure you are not deleting the head of the list.

8. In our sys_delegate_usr function make sure you check for the validity of the arguments that are being passed in. This means that you need to make sure a valid option has been selected, a valid pid was given, a valid uid was given, a valid pointer was given and a valid size was given. You can assume that the maximum size will be 100. If anything is invalid be sure to return a value that indicates that an error has occurred. Otherwise return a value indicating successful completion.

9. In your add user, delete user, and list user functions make sure you check the return values of any other calls you make. For example when you get a pointer to the task_struct for a process make sure you have a valid pointer. When you use kmalloc make sure it returns successfully.

10. Finally, once again we are dealing with code that needs to be placed in a critical section. In each of your functions determine Where your critical sections are and place locks around them.

11. You will need to include headers in your system call. Which ones you include is left as an exercise for you again. Hint: Where are all the list macro’s we are using? Which header file did we modify?

I make no claims that the above list is ALL you need to do for the system call. I easily may have missed something. If you think I did then add it in.
12. Once you have made the above changes (or before you are finished all of them if you want to test). Run the `make` command again to recompile your Kernel. It shouldn’t take as long this time as it did the first time. Keep in mind that we have modified `sched.h` which is included in a lot of files. Each of those files must be recompiled this time. You can halt the compilation process (`ctrl-c`) if you notice errors/warnings for modifications you have made. Once your Kernel is compiled complete the build process and boot into it. When you are in the running Kernel you can create your user land program to utilize your system call.

This should give you a good head start on problem 3 of homework 3. Good Luck with the rest of the assignment!