Midterm Exam

After you leave the lecture session, you will have 60 minutes to finalize your work and work on the video and submission. Please, leave enough time to submit the files. IT difficulty may occur.
You will get zero immediately after 60 min if you do not submit the required files. No exceptions. No excuses..
When you leave the lecture, type in the chatting room (public) that you are leaving the exam along with the time. You will have 60 min from the time you leave the lecture to submit your final solutions. Please, when you type in the chatting room, do not chat privately to me... Chat to public..
No communications at all. The exam is very clear but if you have a question, please, chat to public and I may answer your question.

Please, do not ask me after the exam or during the exam if I received your files. If I do not receive your files, I will contact you. I should be able to receive your files if you follow the instructions.

Submission policy.

- **A single pdf file** includes:
  - All codes of any file you edited or modified or created [Highlighting code modifications, showing the full path of each file, and fully commented code (not necessary line by line)]. If the file is too long and you only edited a small part, just show the part you edited. No need to copy hundreds of lines that you did not touch/edit.
  - Screenshots/printscreen of all outputs you got step by step with a short explanation of each output.

In this pdf file, you should show ALL the codes first then show all the outputs at the very end.

- **A video file** explaining your code and showing the results. Please, do not submit huge files and do not show the unnecessary steps such as make, make clean, and sudo minicom, etc.. The video should be around 5 minutes max.

In this video, please, explain ALL the codes first then show all the outputs at the very end of the video. Do NOT show code-output-code-output. Show all codes then all outputs.

Do not place the previous files in a zipped folder. Do not submit a zipped folder that contains previous files. **Just submit the two files.** Any other formats will not be graded.
Please, make sure that you uploaded the right files before clicking submit. Do not include links to the video. Upload the video itself. Please, do not ask me after the exam or during the exam if I received your files. If I do not receive your files, I will contact you. I should be able to receive your files if you follow the instructions.

Video for some outputs used as guideline.

https://zoom.us/rec/play/u5Esfu-uqjk3SYHGswSDVKUtW47pe6ms03VM_vRYxU_gW3kGZFKuYbYbYrDLvVfGuS_edgcaQMUsFh7T

It is required to add new commands to the Xinu shell.

Recall:
The steps to follow for adding a new Shell command.
Add a new file called shell/xsh XXX.c, where XXX is the name of your shell command. This file should be structured as follows:

```
/* xsh_XXX.c - xshXXX*/
#include <xinu.h>
/*------------------------------------------------------------
* xsh_XXX - shell command to do whatever you want it to
*------------------------------------------------------------
*/
shellcmd xsh_XXX(int nargs, char *args[])
{
// Your code here
}
```

Add an entry to cmdtab in shell/shell.c for your new command.

Add a prototype for your new command function to include/shprototypes.h.

If your command will call a function such as “runforever”, “runforever.c” file should be created and saved in the “system” folder. Then, you have to add the information of the new function “runforever” in the “prototypes.h” (all **.h files** are in the “Include” folder). If they are in any other folder, you may move them into the “Include” folder (it is up to you).

Finally, based on the machine you are using, you may need to open the compile/makefile.c and add the new files you created in the **makefile.c** file. Be careful .. This step is critical.

If you want to use a global semaphore, you may add **extern sid32 globalsemaphore;** in the semaphore.h and declare it in **main.c** as **sid32 globalsemaphore;** above the main ().

It is assumed that you know the meaning of create a process, wait, signal, and resume and their states’ rules.

Functionality of your commands

You are required to create a few commands for the following tasks:

Q1. Create a new process with a user specified priority that starts running in an empty infinite loop after announcing its PID.

In other words, if you type (**create**) as a shell command, you should see the PID of the current process and if you list the processes, you should see the new created process. Create without any priority means default priority of 20.

You can type (**create 10**) for example to determine the priority of the new created process.

Q2. Create a new shell command called "createsleep". The command creates a new process with a user specified priority that starts running in an empty infinite loop after announcing its PID after calling sleep () or sleepms () [reminder: exam is open book/internet] to sleep for 10 seconds. So, the difference between "create" and "createsleep" is the "createsleep" prints the PID after 10 seconds. So, the "createsleep" command will create a process and put it in the sleep state for 10 seconds and then it will wake up and prints its ID.

Q3. Create a new shell command called "psready" to list the content of the ready list only. So, the difference between the "psready" and the "ps" is the "psready" is a new command that only prints the ready processes.

Q4. Create a new shell command called "wait" to create a new process that starts waiting on a semaphore that you created with the initial count of zero after announcing its PID. So, after executing this command, if you list processes by typing "ps", you must see this process in the wait state.

Q5. Create a new shell command called "signaln" to signal the given semaphore using signaln. The signaln should be given a number as an argument such as signaln 1 or signaln 2 to determine how many processes you would like to signal (increment the semaphore count). It must be able to signal more than one process in addition to signal a process since the original "signal" can only signal a single process.

It is up to you to decide if the process that is signaled should terminate or should go to the ready queue (Hint: While (1) or not)

Q6 (not shown in the video) Create a shell command – resumen - that will take process PIDS as arguments and "resume" them all at once. As you know, when you resume a process, a context switch takes place immediately if there is a ready process with equal or higher priority than the current process. Analogous to signaln, we want to be able to resume a number of processes and make sure that the one with the highest priority will run first after the resumption of all. The new resumen has the same meaning and features as resume except it "resume" more than one process.

Feel free to use any programming skills you have. No restrictions. One sample way to perform the tasks Q2, Q3, Q4, and Q5 is given below.

Q2. xsh_createsleep may call a new function called "runafterwait". So, runafterwait.c file needs to be created and defined in the "prototypes.h".

Q3. You can copy the xsh_ps.c file and call the new copy xsh_psready.c and edit the new copy to only print the ready processes.

Q4. In the main.c file, you may create a global semaphore variable to be seen from anywhere (you may need to add this statement "extern sid32 globalsemaphore; in the semaphore.h.). Then, you can create the new command .c file called "xsh_wait.c" that calls a function called "waiter" that waits for the global semaphore.

Q5. You just need to create the new command .c file called "xsh_signaln.c which can just call the default "signaln" function. The signaln command should take an argument (which should be a number). So, it should look like this "signaln 5" to free up five waiting processes/increment the semaphore count by 5.

It is recommended after you signal a waiting process, this waiting process should be placed in the ready queue but of course, it may be placed in the ready queue then runs to completion, So, it is better to force it to run forever so, it can be seen in the ready queue.