OPERATING SYSTEMS (CS F372)
SEM I 2023-2024
ASSIGNMENT 1
MAX MARKS: 30
DEADLINE: 30th SEPTEMBER 2023, 11:59 PM (HARD DEADLINE)

NOTE: PLEASE READ THE ENTIRE DOCUMENT AND NOT JUST THE PROBLEM STATEMENT. THIS DOCUMENT CONTAINS IMPORTANT INFORMATION REGARDING SUBMISSION GUIDELINES, PLAGIARISM POLICY AND DEMO GUIDELINES IN ADDITION TO THE PROBLEM STATEMENT.

PROBLEM STATEMENT:

In this Assignment we are going to use the different concepts learnt so far like Linux commands, process creation, inter-process communication, wait() and exec(). The problem statement of the Assignment consists of the following parts.

1) **Write a POSIX-compliant C program client.c.**

   a) On execution, each instance of this program creates a separate client process, i.e., if the executable file corresponding to client.c is client.out, then each time client.out is executed on a separate terminal, a separate client process is created.

   b) When a client process is run, it will ask the user to enter a positive integer as its client-id. The prompt message will look like this.

   ```
   Enter Client-ID:
   ```

   If you are running four instances of client.out, then for the first instance, the user should enter client-id as 1, for the second instance, the user should enter client-id as 2 and so on. Out of the total clients running, who will get what id, can be decided randomly.

   c) Then, each client will display a menu to the user (no GUI or beautification is required):

   ```
   1. Enter 1 to contact the Ping Server
   2. Enter 2 to contact the File Search Server
   3. Enter 3 to contact the File Word Count Server
   4. Enter 4 if this Client wishes to exit
   ```

   For menu option 1, the client will not ask the user to enter any input. For each of menu options 2 and 3, the client will ask for a filename from the user. Assume the files to be ASCII (text only) files.

d) Upon receiving a command from the user (using the above menu), the client will communicate the command and the necessary argument(s) (if required) to the main server (described later). This communication with the main server should take place only using a message queue.

e) There can be any number of client processes communicating with the main server. All communications between all the clients and the main server (or the main server's children) (both ways, i.e., from client to main server and from server's child to client) need to be done through the single message queue mentioned in the above point. You are not allowed to create more than one message queue for this purpose. [Hint: use the `mtype`]

f) The client will wait for a reply from the server (actually it will be the server's child, as described later). On receiving the reply, the client will display the output to the user and redisplay the menu options, until the user chooses option 4. Upon choosing option 4, the client will gracefully terminate. Graceful termination implies performing any sort of cleanup activities that are required to be done and then terminate.

2) Write a POSIX-compliant C program server.c (let's call this the main server).

a) The main server is responsible for creating the message queue to be used for communication with the clients.

b) The main server will listen to the message queue for new requests from the clients.

c) Upon receipt of a request from a client, the main server will spawn a child server and offload the execution of the specific task (as specified by the client) to the child server. Thus, for every client request, a separate child server will be created and the main server will keep listening for more client requests.

d) All communications between the main server and its children will need to be done only through pipes. No other IPC mechanism should be used for this purpose. You may create as many pipes as required for communication between the main server and its children.

e) For option 1 in the client menu, the client will send a "hi" to the server via the single message queue and a child of the server (spawned by the server) will reply back to the client with a "hello". The "hello" should be sent back to the client using the single message queue described above. After sending the reply, the specific server child will perform the relevant cleanup activities and terminate.

f) For options 2 and 3 in the client menu, the child should use the `exec()` family of functions and the relevant Linux commands to accomplish the tasks. For the file search task, the child server (not the main server) will inform the client if the file exists or not and the client will display the search result to the user. For the file word count task, the child server will communicate to the client the word count for the specified file and the client needs to display the result to the user. However, the results generated by the child server instances need to be communicated to the client via the single message queue described above, in a programmer defined format. You will need to figure out how to do this given that the exec() functions replace the executable code in the current process and then accordingly figure out how to communicate the result back to the client.

g) The main server should loop forever listening for new requests from any of the clients. Upon a request, it spawns a child to service the request and listen for more requests.

3) Write a POSIX compliant C program cleanup.c.

a) This cleanup process can keep running along with the clients and the main server. This process will keep displaying a menu as:

`Do you want the server to terminate? Press Y for Yes and N for No.`

b) If `N` is given as input, the process keeps running as usual and will not communicate with any other process. If `Y` is given as input, the process will inform the main server via the single message queue that the main server needs to terminate. Remember when the main server terminates, the child servers should also terminate. Assume that the main server will never force one or more child servers to exit without servicing the relevant client requests. After passing on the terminate information to the main server, the cleanup process will terminate. When the main server receives the terminate information from the cleanup process, the main server deletes the message queue, performs other cleanup tasks (if required) and terminates.

4) Use `wait()`s appropriately. Perform all relevant error handling properly without which marks will be deducted.

5) Note the files that are used in options 2 and 3 are simple ASCII (text only) files.

6) You can assume that all the C files as well as the files for options 2 and 3 are present in the same directory.

7) You should not use more than one message queue in the entire assignment. The communications which are specified to be done using the single message queue, should be done using the message queue only and no other IPC mechanism should be used in those cases.

SUBMISSION GUIDELINES:

- **All programs should be POSIX compliant C programs.**
- **All codes should run on Ubuntu 22.04 systems.**
- **Submissions are to be done on the CMS course page under the Assignment section.**
- **There should be only submission per group.**
- **Each group should submit a zipped file containing all the relevant C programs and a text file containing the correct names and IDs of all the group members. The names and IDs should be written in uppercase letters only. The zipped file should be named as GroupX_A1 (X stands for the group number). You will be notified of your group number after a few days.**
- **Your group composition has been frozen now. You cannot add or drop any group member now.**

PLAGIARISM POLICY:

- **All submissions will be checked for plagiarism.**
- **Lifting code/code snippets from the Internet is plagiarism. Taking and submitting the code of another group(s) is also plagiarism. However, plagiarism does not imply discussions and exchange of thoughts and ideas (not code).**
- **All cases of plagiarism will result in awarding a hefty penalty. Groups found guilty of plagiarism may be summarily awarded zero also.**
- **All groups found involved in plagiarism, directly or indirectly will be penalized.**
- **The entire group will be penalized irrespective of the number of group members involved in code exchange and consequently plagiarism. So, each member should ensure proper group coordination.**
- **The course team will not be responsible for any kind of intellectual property theft. So, if anyone is lifting your code from your laptop, that is completely your responsibility. Please remember that it is not the duty of the course team to investigate cases of plagiarism and figure out who is guilty and who is innocent.**
- **Please be careful about sharing code among your group members via any online code repositories. Be careful about the permission levels (like public or private). Intellectual property theft may also happen via publicly shared repositories.**

DEMO GUIDELINES:

- **The assignment also consists of a demo component to evaluate each student's effort and level of understanding of the implementation and the associated concepts.**
- **The demos will be conducted in either the I-block labs or D-block labs. Therefore, the codes will be tested on the lab machines.**

- **No group will be allowed to give the demo on their own laptop.**
- **The codes should run on Ubuntu 22.04.**
- **All group members should be present during the demo.**
- **Any absent group member will be awarded zero.**
- **The demos will be conducted in person.**
- **The demos will not be rescheduled.**
- **Though this is a group assignment, each group member should have full knowledge of the complete implementation. During the demo, questions may be asked from any aspect of the assignment.**
- **Demo slots will be made available in the last week of September. You need to book your demo slots as per the availability of your entire group.**
- **The code submitted on CMS will be used for the demo.**
- **Each group member will be evaluated based on the overall understanding and effort. A group assignment does not imply that each and every member of a group will be awarded the same marks.**
- **Any form of heckling and/or bargaining for marks with the evaluators will not be tolerated during the demo.**

****