Operating Systems - Assignment 2 (due 01/06/2020)

Question 1 (IPC) (34 points)**.**
Implement a ping-pong game involving two processes (the parent and son) that are communicating through **pipe()**. The processes send each other an integer, each time increasing an integer by one, and when the number reaches a certain number (**five** in our case), both processes are terminating. The program gives an example of two processes communicating through the pipe to transfer data. Processes require additional synchronization determining the order of readi/write that should be implemented through signals in this case.

Detailed description:
The process creates a pipe and starts a child process. The child process sends a value 0 through the pipe and signals to the parent by using the **SIGUSR1** signal (see kill -l). To get a parent process id use **getppid**(). The signal handler of the parent process checks if the received value is less than 5, increases the value, sends it through the pipe, and raises a signal to the child. In the signal handler of the child if the received value is less than 5, increase the value, send it through the pipe and signal to the parent. The exchange between the processes stops once the value is five and both processes terminate. As for sample output use the following program implementing this game.

To see if a process is overloading default signal handlers, you can check:
cat /proc/2084/status | grep SigCgt
SigCgt: 0000000000000200

Where SigCgt is a bitmask (one bit per signal number). For instance, the current output is for the process id 2084 overloading **SIGUSR1** whose number is 10 and, hence, the 10th bit is turned on.

Output of your program:
0
1
2
3
4
5
Child is going to be terminated
Parent is going to be terminated

Note that your sources will be checked that they are communicating through pipe and use SIGUSR1 signal.

Submission:

Create a folder named IPC for the sources and makefile. You can add a readme file with any explanations if required.

Question 2 (36 points)**:**

The dining philosophers problem

In this question you will investigate a few process synchronization issues using the "dining philosophers problem". You are encouraged to look for references in the web, including for solution suggestions. Links for relevant references are provided in the text.

2.1 Deadlock

Consider the program dining_philosophers.c

2.1.1 Edit the program dining_philosophers.c such that it will get into a deadlock condition with a high probability. (6 points)

2.1.2 Edit the program dining_philosophers.c such that it will never get into a deadlock condition. (6 points)

2.2 Starvation

2.2.1 Edit the program dining_philosophers.c such that it will get into a starvation condition with a high probability. (6 points)

2.2.2 Edit the program dining_philosophers.c such that it will never get into a starvation condition. (6 points)

2.3 Livelock

2.3.1 Edit the program dining_philosophers.c such that it will get into a livelock condition with a high probability. (6 points)

2.3.2 Edit the program dining_philosophers.c such that it will never get into a livelock condition. (6 points)

General: in each file you edit and submit please emphasize the changes you made. Give a short explanation about your algorithm (at the header of the source code file in comments).

Reminder: to compile with the pthread library, you have to link the library pthread. For instance, to compile the file dining_philosophers.c use:

gcc -pthread -o dining_philosophers dining_philosophers.c

Submission:

- create a folder named SYNC for this question.
- The folder will include one source code per bullet named 2_1_1.c, 2_1_2.c, .., 2_3_2.c and one makefile for all the files in this folder.

- Please make sure that your files compile and run correctly. You will not get credits otherwise.

Question 3 (GDB) (30 points):

Congratulations ! You have finished your CS degree, and are looking for a job.

A friend, working in a secret agency suggested you to apply for a job, as they are looking for new talents. In order to apply, you need to generate a secret key, based on your personal ID. The organization will check if you are suitable for their needs and invite you for an interview. The key generator is provided but contains only the best talent names. Try to add yourself to this list, and get a key for you (hint use GDB). The test tool is supplied, please don't hesitate to use it before submitting your request, as only one is allowed.

Note: Company will check authenticity, so no tricks this time.

Submission:
In a folder named GDB put a file with the generated keys: one key for each id of submitting persons.

Important: For questions 1 and 2. Not compiling bullets will lead to zero grade for the corresponding bullets. So check your makefiles in advance!!!

Combined submission:

1. Create a main folder teudat_zeut1_teudat_zeut2 containing three folders with your solutions: IPC, SYNC, GDB, one per question. If a single person is submitting, the name of the main folder is teudat_zeut.

2. Create a zip of the main folder from 1 with the same name teudat_zeut1_teudat_zeut2.zip
3. Submit the zip file to Moodle.