Birzeit University - Faculty of Engineering and Technology
Electrical & Computer Engineering Department - ENCS4330
Real-Time Applications & Embedded Systems - 1^{st} semester - 2025/2026

Project #1
Signals, Pipes & Fifos under Linux
Due: November 9, 2025

Instructor: Dr. Hanna Bullata

Matrix operations using signals, pipes & fifos

We would like to create a multi-processing application that does matrix operations using signals, pipes & fifos. By matrix operations, we mean matrix addition, subtraction, multiplication, computing matrix determinant, calculating matrix eigenvectors and eigenvalues. The matrix operations tool can be explained as follows:

1. When the executable is run, a menu should be displayed where the user can select an operation to do. Below is a list of operations that the user can select from. You can add to it as you see fit:

   ```
   1.  Enter a matrix.
   2.  Display a matrix.
   3.  Delete a matrix.
   4.  Modify a matrix (full row, full column, a particular value).
   5.  Read a matrix from a file.
   6.  Read a set of matrices from a folder.
   7.  Save a matrix to a file.
   8.  Save all matrices in memory to a folder.
   9.  Display all matrices in memory.
   10. Add 2 matrices.
   11. Subtract 2 matrices.
   12. Multiply 2 matrices.
   13. Find the determinant of a matrix.
   14. Find eigenvalues & eigenvectors of a matrix.
   15. Exit
   ```

   Of course, a matrix that results from an operation on 2 matrices is saved in memory for future reference.

2. When 2 matrices are added or subtracted, create as many children processes as you have elements in one of the matrices and send the numbers to add or subtract to the child processes to do the computation and send back the result to the parent process.

3. When 2 matrices are multiplied, create as many children processes as you have rows in the first matrix multiplied by the columns of the second matrix. You need to send each individual row from matrix 1 combined to each individual column from matrix 2 to the child processes to do the computation and send back the result to the parent process.

4. When computing the determinant of a matrix, find the appropriate number of children processes to help speed up computation. Same goes to computing eigenvalues & eigenvectors.

5. Use signals, pipes & fifos in the appropriate way to handle communication and data sharing between processes. You do not need to use all of them.

6. When an operation is terminated, the main menu described in step **1.** above is displayed for further operations.

7. Check on the time each operation needs to finish and compare it to the execution time if the operation runs single-threaded.

8. Use openMP wherever needed to further accelerate the operations by parallelizing loops. Check if you get better or worse results when using it.

What you should do

- In order to implement the above-described application, you need to use the signals, pies and/or fifos facilities. Be wise in the choices you make and be ready to convince me that you made the best choices :-)
- Write the code for the above-described application using a multi-processing approach.
- In order to avoid hard-coding matrices or entering matrices with each run of the application, think of creating a text file that contains a directory where matrices can be loaded from with each run. That will spare you from having to enter them over and over again. You can also use that file to customize the way users wish to see the menu list (e.g. change menu order). Give the file name as an argument to the application when run. That will spare you from having to change your code permanently and re-compile.
- Keep in mind you do not need to create children processes with each run. A better approach would be to keep the pool of created children processes for further computations. That should accelerate future executions. Nonetheless, you might want to use aging to get rid of old children processes that have been hanging around since a while and not doing any work.
- Test your program.
- Check that your program is bug-free. Use the `gdb` debugger in case you are having problems during writing the code (and most probably you will :-). In such a case, compile your code using the `-g` option of the `gcc`.
- The project lead is responsible to send the zipped folder that contains your source code and your executable before the deadline on behalf of the team. If the deadline is reached and you are still having problems with your code, just send it as is!