COSC 350 System Software
Lab #6

How to submit

- For each program, you need write detailed comments for each statement.
- You need demonstrate each task in front of me during the next lab hours.

Task 1 : <u>getenv() function</u>

- Write your own getenv() function called mygetenv() function which has same syntax and semantics.
- Write a simple C program to show your mygetenv() function works well.

Task 2: <u>Creating New Processes using fork</u>

A. Copy fork1.c from BLP 4th edition, page 474 (page 457 in BLP 3rd edition).
B. Compile and run it to be sure you understand what it does.
C. Modify your fork1.c to take four command-line arguments:
    1. Nc - number of iterations for child process
    2. Np - number of iterations for parent process
    3. Tc - sleep time for child process
    4. Tp - sleep time for parent process

    Then, modify the code accordingly.

D. Run the program as fork1 5 3 1 1. You should get the same result as running the original version.
E. Run the program with the following values (and any other values you find interesting):

Nc	Np	Tc	Tp
5	3	1	5
5	3	5	1

Task 3: <u>Using wait</u>

A. Copy your modified fork1.c to a file named forkWait.c.
B. Modify forkWait.c so the parent process waits for the child to finish. Use the code from BLP 4th edition in page 475 (page 458 in 3rd edition), for the wait portion.

C. Print your modified forkWait.c to hand in.
D. Run your forkWait with Nc = 5, Np = 3, Tc = 1, and Tp = 1.
E. Briefly describe how your result differs from the original program in Task 2. Explain .

Task 4: <u>fork with exec</u>

This task forks a child process and uses exec to replace its process image with another image. It's very similar to Task 3, but the child process is implemented as a separate program.

A. Copy forkWait.c from Task 4 to a file named forkExec.c
B. Modify forkExec.c so the child process image is replaced by the image of a program named child. Use one of the exec family of functions to do this.
C. Write child.c to do the same thing as the child process did in Task 4.
    - The child should give its pid each time it prints the message.
    - The parent should give its pid each time it prints the message.
    - The child program should take three command-line arguments, the message, Nc, and Tc.
    - To get an interesting exit status from the child, have it return 37, rather than 0.

Task 5 : <u>file sharing between parent and child.</u>

- Write a C program such that receive a text file name as a argument and open the file for read. And then create a child process. Two processes are sharing same files for read. Both the parent and the child read a byte at a time from the same file and write to independent files. The parent process collect number characters and child collect non-number characters. Without any form of synchronization, the output files for each process will be intermixed between child and parent's works.

- Modify the previous program and try to synchronize and get correct outputs for each process.