Concurrent Processing in C Using Threads, Pipes, and Signals

**Summary**

In this project we will use pipes, threads, and other synchronization mechanism to implement inter-process communication.

**What we will be doing in the project?**

1. **Process Management:** The threads of the parent process will generate the random numbers and write them to the pipe.
2. **Pipe Operation:** These numbers are then read by the child process from the pipe it will process them and calculates the final average.
3. **Signal Handling:** This will ensure that the child process only begins after the completion of the parent process.
4. **Concurrency:** the pthread\_mutex here will help us prevent the race condition during the concurrent process operations

**Approach for the project**

1. **Creating the parent process :**
   1. Created three threads
   2. Each thread will generate 500 random integers and will write them to the pipe.
   3. Once the data is generated then the child process will be signaled by the parent process.
2. **Child process (functioning)**
   1. Before processing the child process will wait for the signal from the parent process.
   2. To read the data concurrently from the pipe it will create 10 threads.
   3. Each thread will calculates the sum of 150 integers.
   4. The result will then be written to the file after performing all the aggregation on this integer (Summing and Averaging)
3. **Signal Handling**
   1. We will use the SIGUSR1 for the data synchronization that will notify the child process to begin execution in short it will ensure that the child process start processing the data only after the parent process has finished generating the numbers.

**Prompts used to get help from the AI tools (Microsoft Co-pilot)**

1. “Give me some examples of how to use threads and pipes in c programming for the inter process communication”
2. “Examples for signal handling in c programming”
3. “How to calculate the sum and average and write it to a file in C”.

**Steps to run and compile the program**

**Compilation**

1. Save the Code.
   1. Save the code in a file
2. Compile the code
   1. Open a terminal and navigate to the directory where the program file is located
   2. Use the gcc to compile the code using the command below

**gcc -o HW-4\_code HW-4\_code.c -lpthread**

* 1. The -o HW-4\_code: Specifies he output executable fiel name as HW-4\_code.
  2. -lpthread is used to link the posix thread library, which is necessary for multi threading

**Running the code**

1. Run the compiled program
2. Monitor the output
   1. Send the random numbers generated by the parent process through the pipe created.
   2. The child process will take this number from the parent process and process these numbers here we are using SIGUSR1 to do that.
3. Output File
   1. The average of these sums will bestored in the output file after the child process finishes it
4. Signal handling before processing the numbers the child will wait for sigusr1
5. Cleanup
   1. Here the dynamic allocated memory is freed within the threads and the pipe will be closed from both the ends after usage.

**Findings/ Learning**

1. Concurrency : Proper use of pthread\_mutex ensures will be used to ensure thread safe pipe operations
2. Synchronization: It manages the correct execution and order of the signals between parent ad the child process
3. Pipe communication: Data Integrity was maintained without race condition

**Short Explaination of the code**

1. Threads in parent process
   1. Generate random integers and write them to the pipe
   2. We will ensure only one thread accesses the pipe at a time using mutex locks
2. Threads in child process
   1. The integers will be read concurrently and the individual sums will be calculates
   2. The average will be calculated using the data aggregation
3. Signal Handling
   1. SIGUSR1 is used that the parent process stop generation of the data and that the child process is ready to process the received data