CSCI 176 (Parallel Processing) Spring 2021 Program 1 (20 pts) Due: Feb. 09 (T)

Practice for task parallel programming with message passing communication among multiple processes using Unix pipe:

Study the attached sample program in which two child processes communicate via Unix pipes (message passing), and write a task parallel program for the following task.

The given task is comparing the efficiencies of two different implementations of the Fibonacci number generation function, i.e., recursive version vs. iterative version.
Instead of doing serial testing in which the two versions of the Fibonacci function are called serially in a process, we plan to test them in parallel with a separate process for each.

Your program should be based on the SPMD task parallel model and include 4 processes (1 parent and 3 child processes).
Upon execution, a numeric value N (for Nth Fibo#) is accessed from the command line argument and the parent process creates needed pipes and forks three child processes.

The first child process is responsible for the control, i.e., sending a number (N) to both the second and the third child processes, receiving results (execution time taken) from those two child processes, and displaying them.
The second child process is responsible for executing the recursive version of the Fibonacci function upon receiving the number N, displaying the result (Nth Fibonacci #), and sends the execution time taken to the first child process (controller).
Analogously, the third child process is responsible for executing the iterative version of the Fibonacci function upon receiving the number N, displaying the result (Nth Fibonacci #), and sends the execution time taken to the first child process (controller).

For the execution time checking, you should include codes for time checking (any kind is OK) in the 2nd and 3rd child processes, i.e., inserting a wall-clock time checking code right before the Fibonacci function call and right after the function call.

All communications should be done in the message passing way using Unix pipes. It is the programmer's job to define and use appropriate number/kind of pipes.

Suggestion: please start with drawing a brief diagram showing needed processes and communication paths with pipes.
More detailed discussion will be done in class.

Submission:

- Source code (.cpp); Please include a good global documentation and each function head documentation in your source code (before compilation).
- Runtime output, e.g., a typescript file or screenshot (png or pdf); output should be readable. Please run with Fibo(20), Fibo(45) and Fibo(47).
- Compress these files into a .zip file, name it "Prog1-youFistName-yourLastName.zip" and send it to jpark@csufresno.edu.

///// sample program with Unix pipe

```
#include <unistd.h> //for fork, pipe, wait
#include <cstdlib>  //for exit(0)
#include <cstring>
#include <iostream>
using namespace std;
#define MAX 100

int main()
{       int pid, status, i;
        int p1[2], p2[2];  //pipe descriptors
        int response, response2;
        char s[MAX];

        pipe(p1); pipe(p2);
        for (i=1;i<3;i++)
         { pid = fork();
           if (pid==0 && i==1 )   //child process1
             { //close(p1[0]);
               //close(p2[1]);
               cout<<" Menu: \n";
               cout<<" ------------ \n";
               cout<<"  1. display Hello "<<endl;
               cout<<"  2. display Bye "<<endl;
               cout<<" ------------ \n";
               cout<<" choice ( 1-2 )?\n";
               cin>>response;

               write(p1[1], &response, sizeof (int));
               read(p2[0],s,MAX);
               cout<<s<<endl;
               //close(p1[1]);
               //close(p2[0]);
               exit(0);
              }//if
           else if (pid==0 && i==2 )   //child process2
             { //close(p1[1]);
               //close(p2[0]);
               read(p1[0], &response2, sizeof (int));
               switch(response2)
                {      case 1 : strcpy(s, "Hello\n");
                                break;
                       case 2 : strcpy(s, "Bye\n");
                                break;
                      default:  strcpy(s,"Try option 1 or 2.\n");
                                break;
                }
               write(p2[1],s,MAX);
               //close(p1[0]);
               //close(p2[1]);
               exit(0);
             }//elseif
          }//for_i

        //Now wait for the child processes to finish
        for (i=1; i<=2; i++)
         { pid = wait(&status);
           cout<<"Child (pid="<<pid<<") exited, status="<<status<<endl;
         }
        return 0;
}//main
```

```
fibo_iter(N)
{ f1=1; f2=1;
  if (N==1 or N==2)
    return 1;
  else
  { for (i=3 to N)
     { temp = f1 + f2;
       f1 = f2;
       f2 = temp;
     }
    return temp;
  }
}

fibo_rec(N)
{ if (N==1 or N==2)
   return 1;
 else
   return (fibo_rec(N-1) + fibo_rec(N-2));
}
```