```cpp
//================================================================================================================
// Header Documentation
/**================================================================================================================
    CSCI 176 Program 1
    Kenneth Willeford

    This program tests a recursive and iterative version of fibonacci in parallel.
    From the command line call the program: ./<executable file name>.out <N>
    Where N is the Nth fibonacci number to calculate.

    The program will output the result of each function as well as the time it took to compute that value.

    Note: My buffer sizes were probably overkill.
================================================================================================================**/
// Imports
//================================================================================================================
#include<iostream>  // cout
#include<ctime>     // clock, CLOCKS_PER_SEC
#include<cstdio>    // atoi, sprintf
#include<cstdlib>   // exit
#include<cstring>   // Everything else C string related
#include<unistd.h>  // fork, pipe, read, write
using namespace std;
///================================================================================================================
/// Prototypes - Function Header Documentation
///================================================================================================================
// A simple function which handles the waiting for a pipe.
// When it can retrieve a value from the pipe it will print that value to the screen.
void waitForFinish(int (&inputFD)[2]);

// The Control Thread, It creates two additional threads(instances of testFunc) as well as a pipe for each of those threads.
// It feeds the output into the other threads, additionally when it is done it lets the main thread know it can terminate.
// Through a pipe it will recieve outputs from it's two child processes allowing it to print the timing data on an on-demand basis.
void conThread(int (&outputFD)[2],int argument);

// Child Process of the Control Thread, it has a tag to identify unique output("rec" | ("itr")) and takes in a function pointer(the fibonacci implementations).
void testFunc(int (&outputFD)[2],int (&inputFD)[2],char tag[],unsigned int(*f)(unsigned int));

// The Iterative Implementation of Fibonacci
unsigned int fibonacciIterative(unsigned int n);

// The Recursive Implementation of Fibonacci
unsigned int fibonacciRecursive(unsigned int n);
///================================================================================================================
/// Main
///================================================================================================================
int main(int argc, char *argv[]){
    // Construct Main Thread output pipe
    int outputChannel[2]; pipe(outputChannel);
    int pid = fork();
    // Child Process continue to control thread, Current Process will wait for output(conThread termination.)
    (pid == 0) ? conThread(outputChannel,atoi(argv[1])) :
        waitForFinish(outputChannel);
}
///=================================================================================================================
/// Implementation - Function Inline Documentation
///=================================================================================================================
unsigned int fibonacciIterative(unsigned int n){
    // Initialize Zeroeth and First Fibonacci Numbers
    unsigned int nMinus1 = 0, nMinus2 = 1;
    // Continue Iterating Through the Sequence Until Stopping Point
    while(n-- > 0){
        unsigned int temp = nMinus2;
        nMinus2 += nMinus1;
        nMinus1  = temp;
    }
    // Output Result
    return nMinus1;
}
unsigned int fibonacciRecursive(unsigned int n){
    // Construct Output Through Recursive Definition: f(1) = 1, f(2) = 1, f(n) = f(n-1) + f(n-2)
    return (n<=2) ? 1 :
        fibonacciRecursive(n-1) + fibonacciRecursive(n-2);
}

void waitForFinish(int (&inputFD)[2]){
    // Create Sufficiently Large Buffer
    char buffer[120];
    // Blocking Read from Pipe
    read(inputFD[0], buffer, sizeof(buffer));
    // Print Recieved Value
    cout << buffer << endl;
}
void conThread(int (&outputFD)[2], int argument){
    // Initialize pipes and process id
    int conFD[2], recPipe[2], itrPipe[2], pid;
    pipe(conFD); pipe(recPipe); pipe(itrPipe);
    // Create Child Process(recursive)
    pid = fork();
    if (pid == 0) testFunc(conFD, recPipe, "rec", fibonacciRecursive);
    // Create Child Process(iterative)
    pid = fork();
    if (pid == 0) testFunc(conFD, itrPipe, "itr", fibonacciIterative);

    // Close Recieving Ends of Pipes
    close(recPipe[0]);
    close(itrPipe[0]);

    // Send Output to Child Processes
    char in[10];
    sprintf(in,"%d",argument);
    write(itrPipe[1],in,(strlen(in)+1));
    write(recPipe[1],in,(strlen(in)+1));

    // Wait for Each Child Process to Finish
    waitForFinish(conFD);
    waitForFinish(conFD);

    // Alert parent thread that it can terminate.
    write(outputFD[1],"programFinished",(strlen("programFinished")+1));
    exit(0);
}

void testFunc(int (&outputFD)[2],int (&inputFD)[2],char tag[],unsigned int(*f)(unsigned int)){
    // Close Recieving End of Output
    close(outputFD[0]);
    // Close Outputting End of Input
    close(inputFD[1]);
    // Read Pending Input
    char buffer[120];
    read(inputFD[0], buffer, sizeof(buffer));
    // Convert Pending Input into integer
    int functionInput = atoi(buffer);

    // Run Benchmark
    clock_t t = clock();
    unsigned int result = f(functionInput);
    double seconds = ((float)(clock()-t))/CLOCKS_PER_SEC;

    // Pipe Benchmark Information to Parent Process
    char output[120];
    sprintf(output,"%s- inp:%d,out:%u,it took: %lf s",tag,functionInput,result,seconds);
    write(outputFD[1],output,(strlen(output)+1));
    exit(0);
}
```