Experiments with Pipes and Signals

Just as with lab courses in the physical sciences, the purpose of this project is to perform experiments to find answers that we already know, in order to gain proficiency with constructing and conducting such experiments. The code you will write for the 2 problems below will be an "experiment" to test for certain properties of pipes and signals. The answer is not the main point -- you already know the answer. Your code must be able to determine the answer, without manual intervention. This is what we call a "test case program" and we will use this concept again during the course.

Problem 1 -- Pipe

We know that a pipe is a FIFO of finite capacity. When the writer(s) is faster than the reader, the pipe fills up. When it reaches capacity, the writer is blocked to enforce flow control (rate limiting). Experimentally determine the pipe capacity on your system. A suggested approach is to set the `O_NONBLOCK` flag (using the `fcntl` system call) on the write side of a pipe that you've created. Write to it using a small write size but don't ever read from it. Report how many bytes you can write into the pipe before the write system call fails. Make sure the failure is `EAGAIN` and not some other error.

Problem 2 -- Signal

If multiple instances of the identical signal are posted to a process before the signal can be delivered, behavior depends on the signal number. For so-called "real time" signals (sig# >=32), as many signal deliveries will happen as there were signals generated. But for conventional signals (sig# < 32), only one instance of the signal will be delivered.

Develop a test program to prove this for one example of a real-time and one example of a non-real-time signal. Establish a signal handler which counts the number of times that the signal is delivered and compare that to the number of times it is generated by several other processes that you fork off. To make this test function automatically, you'll need to have your main process `wait` for all of the senders to complete, then report the signal receipt count. Since signals will be received and handled while the `wait` system call is blocking, you will see `EINTR` errors. These are not fatal errors!