https://folk.ntnu.no/michaeng/tdt4186_21/
michael.engel@ntnu.no

Practical exercises
Spring 2021

Practical Exercises 5

Unix IPC

Please submit solutions on Blackboard by Thursday, 15.04.2021 12:00h

5.1 Pipeline performance

In this exercise, you are going to measure the *bandwidth* provided by Unix IPC methods.
In order to do this, write a benchmarking program. After creating a pipe, your program should split into two process:

- a child process that endlessly writes data blocks of the given size (the content of the data blocks is irrelevant) using `write(2)` over the given IPC method *as fast as possible* and
- a parent process that reads the data blocks of the same size *as fast as possible* using `read(2)` over the same IPC method.

Your program should accept a block size (number of bytes) of the data blocks to be sent from child to parent as its only command line parameter.

a. **Pipeline functionality** (3 points)

   Write the program using *unnamed pipes* (system call `pipe(2)`) and output the cumulative number of received bytes after each `read` call of the parent process.

b. **Performance** (3 points)

   Use the `alarm(3)` libc function to trigger a Unix *signal handler* for the `SIGALRM` signal once a second.

   Implement the signal handler so that it prints the current pipeline *bandwidth*, i.e. the number of bytes received in the previous second. See the `signal(3)` and `alarm(3)` man pages for details.

   Test your program for different block sizes (powers of ten), so 1 byte, 10 bytes, 100 bytes, ... up to the maximum block size possible on your system.

   Investigate the following questions:

   - What is the largest block size supported on your system?
   - What is the highest bandwidth you can achieve and at which block size is this bandwidth achieved?
   - Does the bandwidth change when you start several instances of your program at the same time?

   Hint 1: Setting up a signal handler requires you to pass a *function pointer*. If you are unsure how to use function pointers, you can refer to the tutorial at https://www.cprogramming.com/tutorial/function-pointers.html.

   Hint 2: Comment out the print function that outputs the cumulative number of bytes from part a, otherwise the throughput measured will be off, since the parent would not receive as fast as possible.

c. **Trigger statistics printing** (2 points)

Register and write an additional signal handler to handle the `SIGUSR1` signal. You can trigger the signal by executing `kill -s USR1 pid` in a shell, where `pid` is the process ID of the parent process of your benchmark program. In response to receiving the signal, in the corresponding signal handler, your program should print the *cumulative* number of bytes received over the pipe so far.

Your program should continue to run after printing the information.

d. **Named pipes** (2 points)

Create a variant of your program that uses *named pipes* (see the `mkfifo` man page) instead of unnamed pipes and repeat the measurements from part b.

Hint: Open the named pipe (FIFO) file (using `open(2)` and specifying either `O_RDONLY` or `O_WRONLY` as appropriate) only after executing `fork(2)`. If your program seems to hang when running it a second time, delete the created FIFO file from the file system using the `rm` command in the shell or by calling the `unlink(2)` system call in your program before calling `mkfifo`.