CS 4348/5348 Operating Systems -- Project 1

The goal of this project is to let you get familiar with how to create a process and how to communicate between processes. The project has to be programmed in C++ and executed under Unix (or Linux). We separate the project into three phases to help you build a working program. You only need to submit your final program.

1 Phase 1

First try out fork and pipe in the simplest way.

In main program:
create pipes to communicate with two child processes
create two subprocesses
send a message to the first child process
send a message to the second child process
wait for the child processes to terminate and then terminate

In each of the child process:
simply read in the message from the pipe and print it

2 Phase 2

In this phase, we implement the desired code.

In main program:
create pipes to communicate with child processes
create two subprocesses: adder and factorial
loop till end of input
read in input from file "instruction.dat"
each line in "instruction.dat" is either "fac <integer-x> <integer-field>"
or "add <file name> <integer-field>"
if input starts with add, then send the <file name> and field to the child process "adder"
if input starts with fac, then send the two integers to the child process "factorial"
end loop
wait till both child process to terminate and then terminate

In the child process "factorial":
loop till filed = 0
read in a number x and a number field from the pipe
compute factorial = x! mod field
print (process id, x, factorial) with a nice format;
end loop
terminate

In the child process "adder":
loop till the file name is "stopstop"
read in an input file name and the integer field from the pipe
open the input file, read in the integer numbers in the file
-- first integer from the file is N, the number of integers that follow

```
        -- Rest of the file are N integers to be added together
      compute sum of the N integers mod field
      print (process id, file name, sum) with a nice format;
  end loop
  terminate
```

Note that the integer numbers you read in could be very large numbers, for example 2^{30}, but will be a legitimate positive integer number that fits in one computer "word". Your program should be able to handle any of these numbers.

To make it easier to process the input strings, we assume the file names have the same length. A file name will always be 8 bytes.

You need to electronically submit your program **before** midnight of the due date (check the web page for due dates). The instruction for submitting your program will be posted separately.

3 Phase 3

In the final phase, you need to add synchronization into the communication to make sure that the parent and child processes can work in a synchronous way.

First, prepare a very long "instruction.dat" for the main process and let it read in the data without any constraint. Observe the behavior of the system and see whether there will be lost or garbled input in the pipe or the main process will block.

We will try to synchronize the main process with the child processes by ourselves. The main process will only send the next instruction to a child process if the child process has less than N pending instructions. Each child process should send a response (an acknowledgement) to the main process, indicating its job is done. The main process should keep a count of the number of pending instructions. When the count is N for either of the child processes, the main process stops reading and sending any instruction to either of the child processes. You need to design the synchronization logic. You can use either one pipe or two pipes for the main process to read from the two child processes. N will be a command line input.

The main process should check whether there is any count that reaches N at the end of each read and send cycle. If so, then go to the pipe of the corresponding child and read the acknowledgement. If the pipe is empty, then the main process blocks automatically. If N = 1, then main thread will not send any instruction till the child responds, i.e., main will read one instruction, read one ack, ...