Lab 6: Exam 1 solutions; pipes (Week of 22.2.16)

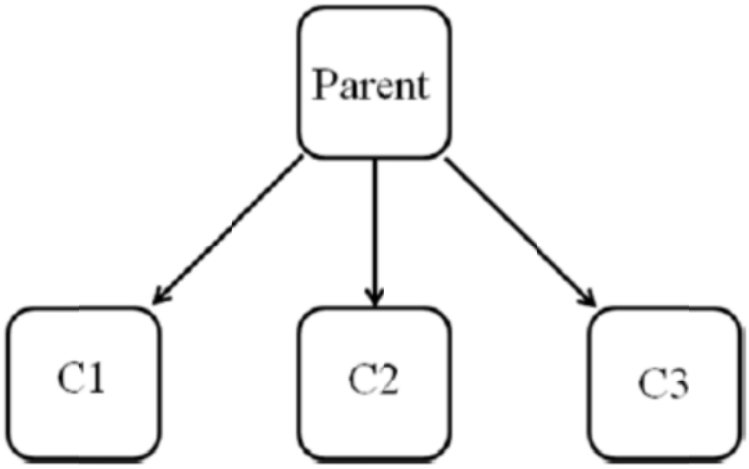
The lab TA will review problems from first in-sem. After that you will write a program that creates pipes. There will be no checkoffs on this lab.

Exam 1 solutions

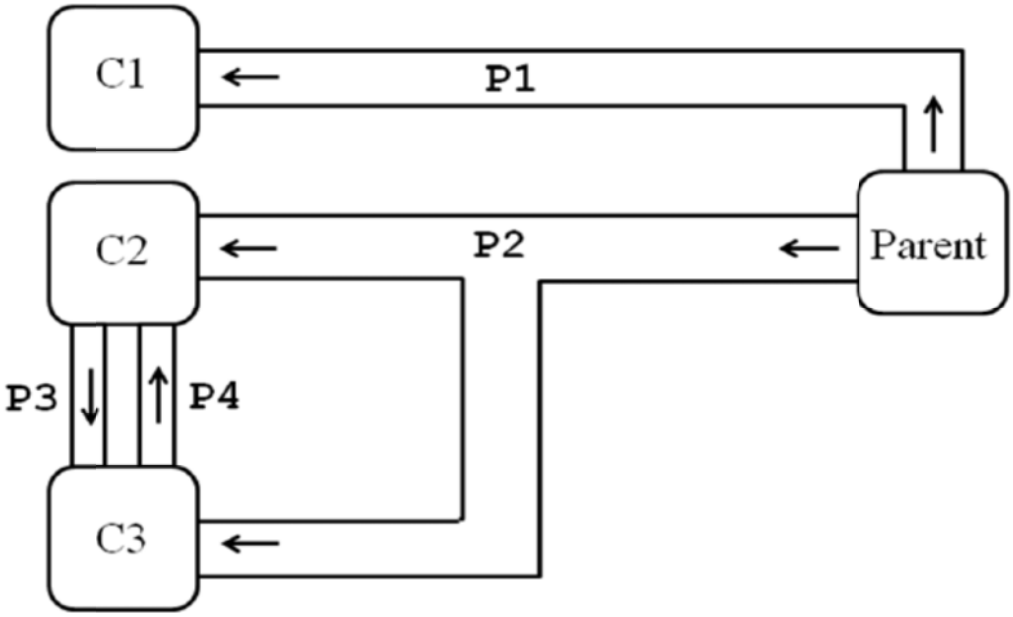
A copy of the first in-sem exam is kept in the Lecture folder. The Lab TA will solve the problems from the in-sem, so be sure to pay attention and take notes if necessary.

Pipes

You will write a C program which first opens FOUR pipes, P1, P2, P3 and P4. Your program should then create three children, C1, C2, and C3 (all in the same generation) as shown in the family tree below.



The structure of pipes and processes should look as follows:



Your program should work as follows:

- **AFTER** opening the pipes and creating the children, the parent process should prompt the user for the number of messages to pass. ONLY the parent should have access to this quantity. The children should not be aware of it!
- Once the parent knows how many messages to expect from the user, it should prompt the user for those messages in the form:
  `"<message string> <child to receive message>"`
- You may assume that messages are only one word in length (you do not need to handle spaces in messages).
- The parent will then use pipes P1 and P2 to send all the messages to the appropriate children.
- Because C2 and C3 share pipe P2, they may receive each other's messages. In this situation, they are responsible for using P3 or P4 to forward the messages as appropriate.
- Each process should ensure that its pipes are unidirectional.
- Once received, messages MUST be printed out in the form
  `"Child <x> read message: <msg>"`
  where `<x>` is the number of the child (1, 2, 3) that is printing the message and `<msg>` is the message itself.

**Hint:* To avoid blocking reads in the children, you should consider what happens when processes close one end of a pipe.
**Hint:* When sending messages to C2 and C3, you may want to append a special character to the message so that they will know if it was meant for them or not.
**Hint:* It's probably a good idea to perform all the writes before performing any reads.

Example execution

```
$ ./lab6
```

```
- Hello, I am the parent. How many messages should I send? 3
- Message 1 (<msg> <child>): Hello 1
  + Sent message along Pipe P1.
- Message 2 (<msg> <child>): World 3
  + Sent message along Pipe P2.
  * Forwarding message along Pipe P3.
- Message 3 (<msg> <child>): Goodbye 2
  + Sent message along Pipe P2.
- Child 1 read: Hello. C1 exiting...
- Child 2 read: Goodbye. C2 exiting...
```

```
- Child 3 read: World. C3 exiting...
- Parent exiting...
```