jkhong_pipex

Considerations for pipex:

- Whenever executing **exec** functions, all existing functions are terminated.
- Only option to execute more than 1 process from a single program is via **forking** to multiple child processes
- As pipex involves communication between parent and child processes, **piping** is used to establish these connections using file descriptors. These file descriptors are generated in pairs, i.e. they have both a read and write output.
- Int dup[2];
- pipe(dup);

Considerations for our processes in reading input:

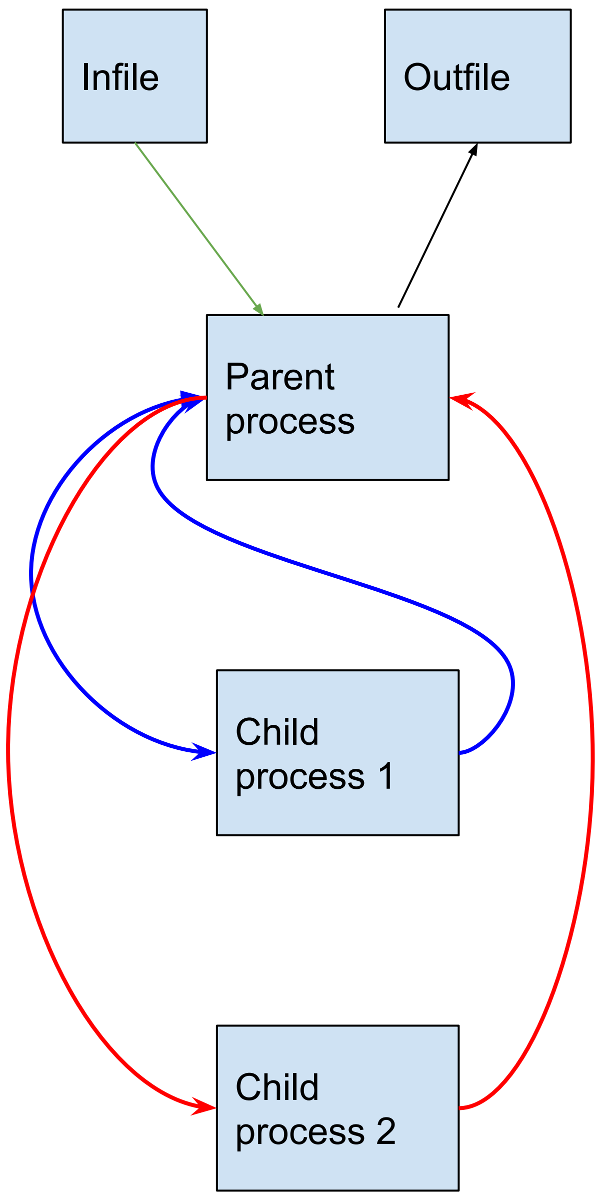
- By default, shell commands read data via STDIN, i.e. 0.
- However as described earlier, information will be transferred between parent and child via pipes in form of file descriptors. These file descriptors clearly are not STDIN, or 0.
- Hence, we use **dup2** to temporarily:

  (i) change file descriptor 0 (STDIN) to the piped fd read

  (ii) change file descriptor 1 (STDOUT) to piped fd write
- So whenever the exec functions are executed, it will read from fd 0 STDIN (which is our file) and fd 1 STDOUT (which is another file)

Piped connections:

- While we are maintaining pipes, order or reading and writing is very important.
- It is tricky to maintain because child processes are ran in parallel to the parent and with each other
- There may be instances where the same pipe is used in the wrong order, hence the correct party is not writing/reading properly.
- I have established 2 pipes, 1 for reading and 1 for writing to mitigate this error
- STDIN requires all write pipes to be closed off. Hence ordering in closure is also required



For each communication between parent and child

(i) Establish pipes, one specifically for read, and another specifically for write

(ii) Fork process

Child

(iii) Switch STDIN read pipe to our read pipe defined in .i

(iv) Switch STDOUT write pipe to our write pipe defined in i.

(iii) Close pipes (order is important)

(iv) Execute command (shell cmd will read from STDIN (i.e. our swapped read pipe) and write to STDOUT (i.e. our wrapped write pipe)

Parent

(iv) Store pipe read values in temporary file descriptor.

(v) Close pipes established in i.

(vi) After looping through process above for all child processes/commands, write output to a file