Operating Systems Lab 3 report

Joshua Marple

September 2014

Introduction

1

My program creates 4 child processes and 3 pipes. These 3 pipes send data between the 4 processes. Child 1 sends to pipe 1, child 2 reads from pipe 1, writes to pipe 2, etc.

2

I had substantial problems writing this program. The main issue was with hanging pipes. Debugging was not of much use to me, as multiple thread debugging lay outside of my ability to handle with GDB. Instead of using GDB, I went back to the tried and true method of various print statements. I found through some trial and error that closing every pipe inside of every child process ultimately fixed the issues I was having with the code. The execl statements were mostly formatted properly, so after getting the pipes working there were only minimal changes to make.

3

Pipes contribute to the Unix philosophy in a number of ways. First and most importantly, they allow inter-process communication through file writing. While in reality, these files are never written to disk, they allow the appearance that they are, which is good from a software engineering viewpoint. This allows the programmer to use the same interfaces that they are used to and the same commands they're already acquainted with.

I think this is an extremely good philosophy for software engineering. Writing programs that do one thing well is a very basic aspect of software engineering: modularity. Having programs work together is obviously critical, as it enables a modular design. Therefore pipes and inter-process communication in general are a critical part of any software engineers toolbox, as it allows solid, modular code.