UMEÅ UNIVERSITET
Institutionen för datavetenskap
Obligatorisk uppgift 1

23 september 2024

Pretitle

5DV088 Systemnära Programmering

Obligatorisk uppgift 1 — Mexec

version 1.0

Namn	Jon Sellgren
Användarnamn	hed22jsn

Labrättare
Graders

Innehåll

1 Introduktion

Introduktion

Målet med denna obligatorisk uppgift var att hantera unix kommando och sedan exekvera dom pararellt med hjälp av child processer och pipor. Följande krav ställdes på koden.

- Programmet ska använda sig av: fork, pipe, close dup2 (eller annan variant av dup), execvp (eller annan variant av exec) och wait (eller annan variant av wait)
- Varje kommando ska läsa från föregående kommando i pipelinen, eller standard input för första kommandot.
- Varje kommando ska skriva till nästa kommando i pipelinen, eller standard output för sista kommandot.
- Alla kommandon ska köras parallellt.
- Programmet får inte ha några minnesläckor, använd valgrind.
- Programmet ska kunna köra ett godtyckligt antal kommandon och ska därför inte använda hårdkodade storlekar på datatyper för detta. Använd realloc eller en dynamisk datatyp för att lagra data kopplat till antal kommandon.
- Då programmet körs med en fil som indata får filen inte manipuleras under körningen.
- Koden ska vara väl strukturerad och skrivas med god kodkvalité (modularisering, indentering, variabelnamn, kommentarer, funktionsuppdelning, etc.). Programmets huvudfil ska heta mexec.c.

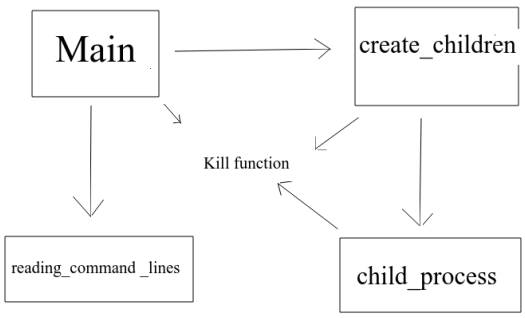
2 Beskrivning av programmet

Beskrivning av programmet

Programmet i sin helhet läser in kommandon från stdin eller en fil, bearbetar strängar pararellt för att sedan ge sin output till stdout.

2.1 reading_command_lines

Funktionen tar in en filpekare att läsa ifrån, en int pekare som sedan ska hålla mängden argument som läses in och en array av strings som håller alla argument på varsin index Funktionen läser in text från en textfil eller stdin för att sedan lägga i en buffer, bufferten lagrar då hela kommandot. Funktionen strtok lägger sedan in pekare på för varje sträng som urskilt med hjälp av ett mellanslag.Innan varje argument ska sparas undan så måste minnet för varje index omallokeras så att dom inte tar mer minne än det som behövs, detta görs genom att ta strlen på token som plockas och sedan omallokera med realloc. Slutligen så kopierar man in strängen med strcpy för att spara undan det, sen går man till nästa sträng. Funktionen returnerar mängden rader som läst och ger även ut mängden argument som läses in.



Figur 1: Anropsdiagram där kill$_function anropas av flera beroende på var programmet slutar (exit failure eller s

2.2 kill_function_arr

Tar in antal index en array har och en array av strings. Funktion frigör minnet som tilldelats till string arrayen.

2.3 create_children

Funktionen skapar först alla pipor som behövs vilket är antal kommandon minus 1. Därefter skapas child processer som kallar på funktionen för att starta sin process efter rätt index för kommandot har hittats. När children har skapats så går parent och väntar tills child processerna är färdig.

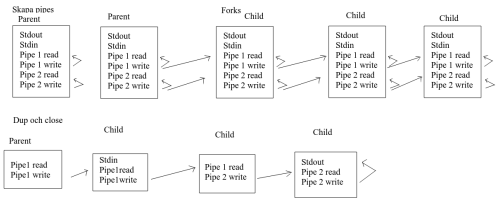
2.4 child_process

Child process ser till omdirigera så att rätt input/output går till rätt pipa eller eventuellt stdout, efter det är gjort så stängs pipor som inte längre behövs. Slutligen så körs kommandot som behöver exekveras med hjälp av en execute buffer som tidigare fyllts på från command bufferten. Exec funktionen fungerar så att den behöver en fil att köra och eventuella argument som ska köras, och eftersom unix kommandon är i grund och botten filer i själva operativsystemet så fungerar det även där. Om det är så att ingen path till filen är anged så kommer den gå till default pathen i operativ systemet.

3 Algoritm för pipor

Algoritm för pipor

Först skapas alla pipes där alla pipes write pekar mot deras egna read. Forks skapas och får kopior av alla file descriptors som parenten har alltså så är alla processer identiska. För att urskilja dom senare så måste pipes omdirigeras och stängas, detta ger då en pipelin. Om man är den första childet så måste stdout



Figur 2: Pipe diagram ifall man startar 3 processer

omdirigeras till pipens write ände, parent stänger sin write och read end för att undvika problem. Sista child behöver omdirigera sin stdin till föregående childs read. Alla childs som är emellan måste omdirigera sin stdin till föregående childs stdin och deras stdout ska omdirigeras till deras write end.

4 Diskussion och reflektion

Diskussion och reflektion

Under uppgiftens gång så vart det en del omskrivningar av kod pga missförstånd av hur man ska dela upp kommandon samt hur man skapar pipor och children. Mycket problem var pga ett missförstånd hur exec fungerar som kräver både filen i sig samt hella kommandot så det räcker inte bara med att köra dess argument separat, t.ex grep -B4 -A2 return så måste första argumentet vara filen alltså grepöch argumentet grep -B4 -A2 returndet räcker inte bara med kommandon. Det första försöket var gjort med hjälp av en struct som ska hålla kommandot och sedan argumenten då det går att lösa det med hjälp av struct så blir det väldigt onödigt då allt man behöver är en array av strings, annars för att exekvera i slutet så kommer man behöva göra någon typ strcat för att få ett komplett kommando.

Därefter så måste man tänke på att strängar inte är något som finns i c egentligen utan det är en char array, så om man ska spara undan strings i en array så måste man spara undan det i filtypen char** då varje enskild räknas som en egen array.

Det är even viktigt att förstå hur forks fungerar eftersom så fort en fork görs så kör child processen, om man inte gör någon typ av check ifall det är en parent eller en child så leder detta till att vissa kodbitar exekveras 1 gång av parent och sen en gång av varje child.