Pretitle

5DV088 Systemnära Programmering

Obligatorisk uppgift 1 — Mexec

version 2.0

Namn	Jon Sellgren
Användarnamn	hed22jsn

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

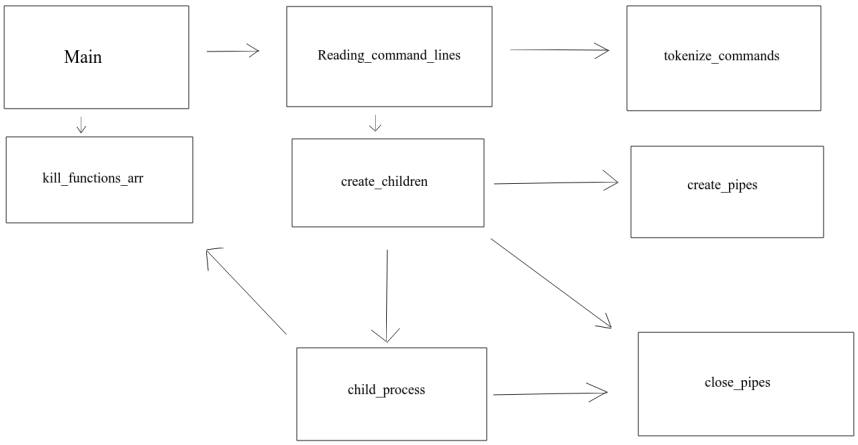
Programmet i sin helhet läser in kommandon från stdin eller en fil, bearbetar strängar parallellt för att sedan ge sin output till stdout.

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.

2.2 tokenize_command

Funktionen strtok lägger in pekare på för varje sträng som urskilt med hjälp av ett mellanslag.Innan varje argument ska sparas undan så måste minnet för



Figur 1: Anropsdiagram

varje index omallokeras så att dom inte tar mer minne än det som är nödvändigt, detta görs genom att ta strlen på token som plockas och sedan omallokera med realloc. Här allokeras även hela fältets längd så mängden argument som kan tas in inte är hårdkodat. 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.

2.3 kill_function_arr

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

2.4 create_children

Skapar alla barn som är samma mängd som antalet kommandon som behöver köras. Efter det så anropar varje barn child_process medans föräldern väntar på att dom ska vara klar

2.5 child_process

Barn processen 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 är nödvändigt. Slutligen så körs kommandot som behöver exekveras med hjälp av en exekveringsbuffer som tidigare fyllts på från kommando 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 filväg till filen är angett så kommer den gå till standard filvägen i operativ systemet.

2.6 Create_pipes

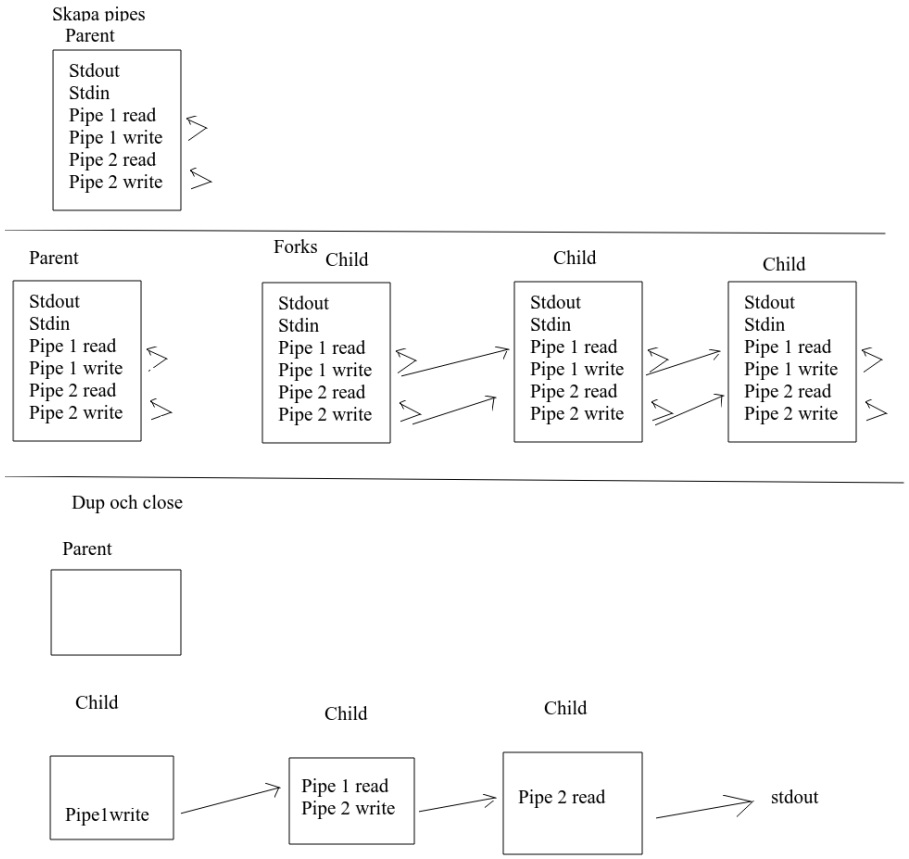
Skapar pipor för antal kommandon minus 1.

2.7 Close_pipes

Stänger pipor som inte används.

3 Algoritm för pipor

Algoritm för pipor



Figur 2: Pipe diagram ifall man startar 3 processer

Pipor används i Unix-liknande system för att möjliggöra kommunikation mellan processer, genom att skicka data från en process till en annan utan att behöva lagra den tillfälligt i filer. En pipa består av två filbeskrivare: en för att läsa

och en för att skriva. Genom att koppla samman flera pipor kan man skapa en pipeline där utdata från en process fungerar som indata till nästa.

I detta exempel körs tre processer, som visas i figur 2. Först skapas två pipor i föräldraprocessen för att länka processerna. Varje pipa har en skrivände och en läsände, och piporna skapas innan några barnprocesser startas. Vid varje barnprocess måste filbeskrivarna omdirigeras med systemanropet `dup2()`, som låter en filbeskrivare (till exempel standardutgången `stdout`) kopplas till en annan filbeskrivare (till exempel en pipas skrivände).

För den första barnprocessen stängs läsänden av den första pipan, eftersom processen endast ska skicka utdata till nästa process. `dup2()` används för att omdirigera standardutgången (`stdout`) till pipans skrivände, så att utdata skickas till den första pipan istället för till terminalen.

För mellanstegens processer (som varken är den första eller sista) omdirigeras både `stdin` och `stdout`. Standardingången (`stdin`) omdirigeras till föregående pipas läsände, och standardutgången (`stdout`) omdirigeras till nästa pipas skrivände, så att data kan passera genom processen.

I den sista processen stängs alla onödiga filbeskrivare, och standardingången omdirigeras till den sista pipans läsände, så att processen kan ta emot data från föregående steg i pipelinen.

Efter att alla processer har fått sina filbeskrivare omdirigerade stängs alla pipor som inte längre används. Varje barnprocess ersätter sedan sin egen kod med det program den ska köra (till exempel `ls`, `grep`, eller `wc`) med hjälp av `execvp()`. När programmen har exekverats terminerar barnprocesserna, och föräldraprocessen väntar på att alla barn ska avslutas innan den själv terminerar.

4 Diskussion och reflektion

Diskussion och reflektion

Under arbetet med uppgiften var det en del omskrivningar av koden på grund av missförstånd angående hur man ska dela upp kommandon samt hur man skapar pipor och barnprocesser.. Mycket problem var pga ett missförstånd hur exec fungerar som kräver både filen i sig samt hela kommandot så det räcker inte bara med att köra dess argument separat, t.ex grep -B4 -A2 returnså 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 struktur som ska hålla kommandot och sedan argumenten då det går att lösa det med hjälp av struktur så blir det väldigt onödigt då allt man behöver är en fält 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 ett char fält, så om man ska spara undan strängar i ett fält så måste man spara undan det i filtypen char** då varje enskild räknas som ett egen fält.

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

Det uppstod också en del problem med att dela upp varje kommando i ett strängfält när man skulle skicka det för att undvika att köra alla samtidigt. Detta leder till många exec felkoder men löste med lite hjälp och diskussion med en annan från klassen. Man måste hålla koll på vilken index kommando börjar på och sedan tilldela varje index till en ny sträng fält tills kommandot är slut, därefter går det bara att mata in till exec funktionen.