UMEÅ UNIVERSITET
Institutionen för datavetenskap

15 september 2023

Systemnära programmering 5DV088

OU1 - Rapport

version 1.0

Noel Hedlund c22nhd

Kursansvarig
Labrättare

Innehåll

1 Introduktion

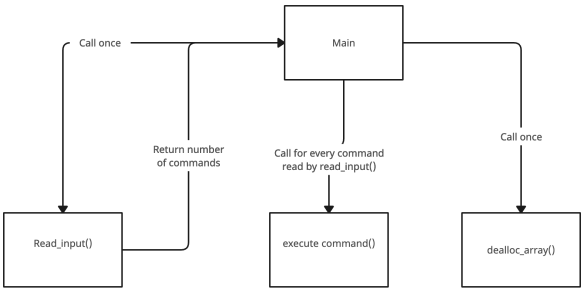
Rapport tillhörande programmeringsuppgiften **Mexec**. Uppgiftens bestod utav att skriva ett program som kan exekvera en kommandopipeline. Vilket innebär att exekvera en rad av program parrarellt, där varje programs utdata blir nästa programs indata. Programmet skrevs i språket C för en Unix-miljö. Ett exempel på hur exevering kan se ut är följande
./mexec
cat -n mexec.c
grep -B4 -A2 return
less

2 Systembeskrivning

Programmet tillåter ett eller noll argument. Vid körning med argument skall en fil innehållande kommandon ges. Vid körning utan argument läses commandon in med **STDIN** och slutar när den påträffas **EOF**.

Lösningen på uppfiten centras kring att skapa barn-processer och i dem exekvera de givna kommandona. Programmet börjar med att läsa in data genom en given **STREAM** och sparar detta i en dynamiskt allokerad array av pekare till pekare till **char** (**char **array**). Samt sparar ett heltal för antalet kommandon som ska exekveras.
Programmet skapar därefter det korrekta antalet pipes motsvarande antalet kommandon. För varje inläst kommando skapas en barnprocess där kommandot exekveras. Innan avslut avallokeras allt använt minne av funktionen **dealloc_arr()**. Därefter väntar **main** på att samtliga barnprocesser ska avslutas korrekt, innan programmet stängs.
Vid en korrekt körning skriver programmet inte ut någon input, förutom output ifrån de körde kommandona.
Vid fel printas ett error meddelande ut och programmet stängs.

Figur 1 illustrerar programmets funktionsanrop.



Figur 1: Anropsdiagram - mexec.c

2.1 Inläsning av data

Inläsningen av kommandon sker genom funktionen **int read_input(char ***command_arr, FILE *fp)**. Funktionen tar två argument, en trippelpekare till **char** där kommandon ska lagras, samt en filpekare för filströmmen där indata skall läsas. Funktionen kallas med en filpekare till en inputfil, eller med **STDIN**. Funktionen lagrar alla inlästa kommandon i den medskickade arrayen, samt returnerar antalet inlästa kommandon.

2.2 Fork och Exekvering

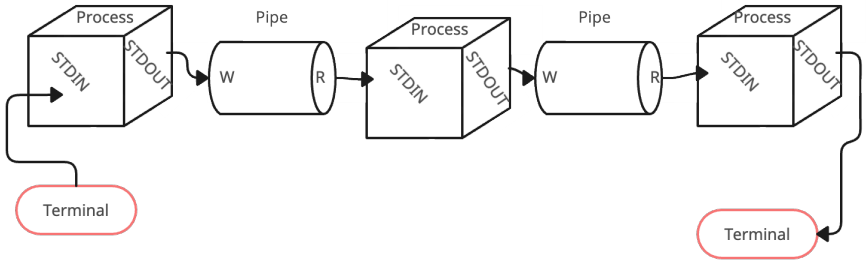
Programmet använder funktionen **void exec_command(char *command, int number_of_commands,int pipe_array[number_of_commands - 1][2], int current_iteration, pid_t *process)** för skapa nya processer, samt för att exekvera de givna kommandona. Funktionen börjar med att kalla på **fork()**. Detta skapar en barnprocess där commandot kan exekveras. Funktionen diregerar om **stdin** och **stdout** med pipes(se sektion 3).

2.3 Dynamiskt minne

Minne allokeras av funktionen **int read_input(char ***command_arr, FILE *fp)**. Funktionen allokerar dynamiskt minne för commandraderna som läses in.
Funktionen **void dealloc_arr(char **command_arr, int number_of_commands)** avallokerar minne för den givna arrayn.

3 Algoritm beskrivning - Pipes

För att omdirigera output och input för processer används pipes. En pipe är en väg för data att färdas mellan två processer.
Programmet skapar (n-1) pipes där n är lika med antalet kommandon. Alla pipes skapas efter inläsningen och kommandon och sparas i en array. Eftersom alla pipes har skapats innan **fork()** används ärver alla barn-processer alla pipes. När processer skapas en efter en, tilldelas en pipe den korrekta änderna av dess tillhörande pipes och stänger resterande.
Figur 2 illustrerar hur pipes ser ut i ett exempel med tre processer.



Figur 2: Illustration pipe

4 Diskussion och reflektion

Den stora delen av arbetet har gått snabbt och smidigt. Mindre problem vid starten av arbetet på grund av ett längre uppehåll ifrån programmeringspråket C. Där C skiljer sig stort språk som JAVA, som använts mycket på senare tid i kurser, där minne inte behöver allokeras av användaren. Utöver detta gick uppgiften problemfritt. Uppgiften var lärorik och gav en förståelse för grunderna i systemnära programmerings. Mindre problem med pipes gav en godförståelse av fel som kan uppstå vid systemnära programmering i unix.