Pretitle

5DV088 Systemnära Programmering

Obligatorisk uppgift 2 — Mmake

version 1.0

Namn	Jon Sellgren
Användarnamn	hed22jsn

1 Introduktion

Introduktion

Målet med obligatorisk uppgift 2 var att skapa en förenklad version av Unix verktyget make, som används för att automatisera byggandet av program. Programmet ska kunna hantera flaggor -s som är för silent mode, -b som är för force build och -f som specificerar filväg. För uppgiften så fick man även två filer parser.c och parser.h för att kunna analysera makefilen. Utöver detta så fanns följande krav på kod

- Programmet ska använda sig av: stat eller lstat, fork, execvp (eller annan variant av exec), wait (eller annan variant av wait) och getopt (inklusive optind) för att läsa in argument.
- Programmet får ej ha några minnesläckor (använd verktyget valgrind).
- Ändringar får inte göras i de tillhandahållna filerna.
- Koden ska vara väl strukturerad och skrivas med god kodkvalité (modularisering,
- indentering, variabelnamn, kommentarer, funktionsuppdelning, etc.).
- Programmets huvudfil ska heta mmake.c.
- Inlämningen ska klara alla tester på Labres.

2 Beskrivning av programmet

Beskrivning av programmet

Programmet är en förenklad version av make som enbart tar flaggorna -s, -f, -B. Make är ett program för att analysera makefiler och sedan bygga mål beroende på regler och beteenden som skrivits i makefilen.

2.1 Flags

- s flaggan(silent flagga) speciferar ifall kommandona som körs i makefilen skrivs ut till stdout, om den är satt så kommer inget skrivas ut.
- B flaggan(force rebuild) säger att målet måste byggas om oavsett vad, även om det inte blir någon förändring.
- f flaggan(File path) finns ifall man vill sätta en egen fil att köras make istället för default vilket i det här fallet är mmakefile.

  En exempel körning är följande:

```
mmake -f annanmmakefile -B
```

Anropar programmet mmake sätter makefilen till annanmmakefile, tvingar programmet att bygga om och då s flaggan inte är satt så skrivs alla kommandon ut efteråt.

2.2 Hur en makefile är uppbygd

Nedan är ett exempel på dom första raderna för makefilen som användes för att kompilera programmet:

```
mexec: mexec.o parser.o
        gcc -o mexec mexec.o parser.o

mexec.o: mexec.c
        gcc -c mexec.c

parser.o: parser.c
        gcc -c parser.c
```

På första raden mexec: är målet som ska byggas, det får inte finnas ett mellanslag innan målet och det ska avslutas med ett kolon. Målet som ligger högst upp i makefilen blir default mål. Varje mål har två regler ena är beroenden och kommandon. Beroenden är en lista över saker som behöver göras innan kommandot får utföras, varje beroende är separerat med ett mellanslag. Beroenden är filer som måste finnas innan målet utförs, och om dom har egna beroende så måste det utföras först. Nästa rad så är kommandon som måste indenteras med tab och där kommer kommandon som målet utför i exemplet så kompileras program med gcc som kompilator. Längre ner så ser man mål som är beroende för första målet, dom målen har då även kommandon som säger att det ska kompileras med gcc. Programmet har även funktionalitet att kolla ifall filer har blivit ändrade och se ifall dom behöver kompileras detta så att man inte kompilerar i onödan. Alla dessa funktioner kopplas direkt till separatkompilering där alla källkodsfiler kompileras separat till objektfiler, ifall objekt fil redan finns och ingen ändring gjorts så kompileras det ej. Detta leder till att inget kompileras i onödan vilket är fördelaktigt ifall man sitter med större projekt eller ifall man använder standardbibliotek.

3 Diskussion och reflektion

Diskussion och reflektion

Ett stort problem som uppstod var checken för att kolla ifall filer finns eller om dom behöver skapas, detta är löst genom att använde funktionen access eftersom man inte använt innan så var det lite oklarheter med att hantera det som funktionen returnerar. Tillslut så blev att ifall ingen regel fanns och filen finns så returnerar man. Efter det så kommer det till exekveringen av kommandot och checken som görs innan. Om regel finns och filen inte finns så måste det exekveras då filen inte körts tidigare.

Den här versionen av make är funktionell men inte lika praktiskt som GNU make då den saknar några saker som skulle kunna underlätta för användaren att skriva sin makefiler. Den första är avläsning av variabler för att slippa onödig klippa klistra vid större projekt. Att implementera den här funktionalitet skulle vara lite rörigt men fullt möjligt att implementera med lite mer tid.