Obligatorisk uppgift 2: mmake

Namn: **Abdiaziz Ibrahim Adam**
CS-användare: **dv23aam**
Kurs: **Systemnära programmering**

December 3, 2024

1 Användarhandledning

Programmet **mmake** används för att automatisera byggandet av program enligt regler i en make-fil. Det är en enklare version av Unix-verktyget make. Programmet förenklar processen för att kompilera och bygga program genom att låta användaren specificera mål (targets) och deras beroenden (prerequisites) i en make-fil.

I make-filen definieras varje mål som en fil som ska skapas eller uppdateras. Beroenden anger vilka andra filer som måste vara aktuella innan målet kan byggas. När mmake körs, analyserar det dessa regler för att avgöra vilka filer som behöver byggas om, baserat på deras senaste modifieringstider. Om ett mål inte existerar eller om någon av dess beroenden har ändrats sedan den senaste bygget, kommer mmake att exekvera de angivna kommandona för att uppdatera målet.

Make verktyget är särskilt användbart för projekt med många filer och komplexa beroenden, eftersom det automatiskt hanterar dessa relationer och sparar tid och ansträngning för utvecklaren.

1.1 Köra Programmet

Kommandoformatet för att köra ‘mmake‘ är som följer:

`mmake [-f MAKEFILE] [-B] [-s] [TARGET ...]`

Beskrivning av kommandoraden:

- **mmake**: Namnet på programmet som ska köras.
- **-f MAKEFILE**: En valfri flagga som låter dig ange en specifik make-fil istället för den förinställda filen mmakefile. Om denna flagga utelämnas kommer programmet automatiskt att söka efter mmakefile i den aktuella katalogen.
- **-B**: Med denna flagga tvingas programmet att ombygga alla mål, oavsett om de har ändrats eller inte. Detta kan vara användbart för att säkerställa att alla komponenter i programmet är aktuella.
- **-s**: Denna flagga döljer de kommandon som körs, vilket innebär att inga meddelanden skrivs ut till standardoutput under programkörningen. Detta är praktiskt om du föredrar en renare utskrift utan extra information.

För att köra programmet `mmake` måste man först kompilera det med en C-kompilator som `gcc`. Använd en makefile som är skriven manuellt och som säkerställer att programmet kompileras utan varningar. Vid kompileringen bör följande flaggor användas:

```
-g -std=gnu11 -Werror -Wall -Wextra -Wpedantic ....osv
```

När programmet har kompilerats kan man köra det genom att skriva `./mmake` i katalogen där det finns installerat. Om du vill ange specifika argument, placera dem direkt efter `mmake`. Till exempel:

```
./mmake -f custom_makefile -B -s
```

Här anger `-f` en anpassad make-fil, `-B` tvingar ombyggnad, och `-s` döljer kommandoutskrift.

1.2 Filformat

Make-filen ska följa följande format:

```
target : [PREREQUISITE ...]
        program [ARGUMENT ...]
```

Där `target` är namnet på målet som ska byggas, exempelvis en exekverbar fil eller ett objekt, och `PREREQUISITE` är de beroenden som måste vara uppfyllda innan målet kan byggas. Dessa beroenden kan vara andra filer eller mål som är nödvändiga för att framställa `target`.

Det är viktigt att kommandot som ska exekveras för att bygga målet föregås av ett tab-tecken, vilket är en grundläggande regel i formatet för make-filer. Utan detta tab-tecken kommer kommandot inte att tolkas korrekt av programmet. Kommandot kan inkludera ett program som `gcc` för kompilering, följt av eventuella argument som behövs för att utföra kommandot, såsom filnamn eller flaggor.

Här är ett exempel på hur en enkel make-fil kan se ut:

```
my_program: my_program.o other.o
        gcc -o my_program my_program.o other.o

my_program.o: my_program.c
        gcc -c my_program.c

other.o: other.c
        gcc -c other.c
```

I det här exemplet definieras `my_program` som ett mål som beror på `my_program.o` och `other.o`. Kommandot för att skapa `my_program` exekverar `gcc` för att länka de objektfilerna. Varje regel i make-filen måste ha ett mål och ett motsvarande kommando, vilket gör det tydligt vilka operationer som utförs för att bygga programmet.

1.3 Vad programmet kan göra

Programmet `mmake` fungerar som en förenklad version av Unix-kommandot `make`.

En av de centrala funktionerna av detta program är dess förmåga att automatiskt avgöra vilka filer som behöver byggas om. När en make-fil anges, kontrollerar `mmake` modifieringstiderna för målen och deras beroenden. Om en fil har ändrats sedan det senaste bygget, kommer programmet att se till att det aktuella målet byggs om.

En annan viktig funktion i `mmake` är separat kompilering. Denna funktion tillåter kompilering av enskilda källkodsfiler till objektfiler utan att behöva kompilera hela projektet varje gång. Detta är särskilt användbart för större projekt med många källfiler. Med `mmake` kan användaren enkelt definiera kommandon för varje källfil i make-filen.

Till exempel, om du har källfiler som `my_program.c` och `other.c` som nämndes tidigare, kommer varje källfil att kompileras till en objektfil, vilket gör det möjligt att länka dem senare för att skapa det slutliga programmet. Tack vare `mmake` behöver användaren endast bygga om de filer som har ändrats sedan den senaste kompileringen.

Genom att använda separat kompilering sparar utvecklare både tid och resurser, särskilt när projekten blir mer komplexa. `mmake` gör utvecklingsprocessen mer effektiv och organiserad genom att automatisera dessa uppgifter.

2 Diskussion och Reflektion

Utvecklingen av programmet `mmake` har varit en intressant och lärorik, men den har också medfört flera utmaningar. En av de största utmaningarna var att implementera den automatiska beroendehanteringen. Detta var viktigt eftersom man behövde överväga noggrant hur filernas modifieringstider skulle kontrolleras för att avgöra om ett mål behövde byggas om. Att hantera dessa tidsstämplar korrekt var avgörande för att programmet skulle fungera som förväntat. Jag använde funktioner som `stat` för att hämta information om filernas senaste modifieringstider, vilket ibland ledde till förvirring kring hur programmet skulle avgöra när en fil var "uppdaterad".

En annan betydande utmaning var att säkerställa att programmet klarade av testerna i Labres. Jag stötte på flera problem när jag försökte få `mmake` att fungera enligt de krav och specifikationer som angavs i uppgiften. Det var viktigt att noggrant följa anvisningarna för hur kommandon skulle skrivas och hur beroenden skulle hanteras. När jag testade programmet upptäckte jag olika buggar, särskilt relaterade till hur programmet hanterade saknade filer och hur det rapporterade fel till användaren. Processen att identifiera och åtgärda dessa problem var mycket lärorik. Jag är glad att kunna rapportera att jag nu har lyckats åtgärda alla identifierade buggar, vilket förbättrar programmets stabilitet och tillförlitlighet avsevärt.

Trots dessa utmaningar har jag haft mycket roligt med uppgiften. Jag har uppskattat möjligheten att skapa en enkel version av make-verktyget, och har fått en djupare förståelse för hur Make-filer fungerar.

Sammanfattningsvis har detta projekt gett mig värdefull erfarenhet inom programmering och testning. Jag har insett att jag har mycket kvar att lära, särskilt när det gäller felsökning och kvalitetssäkring av programvara. Framöver behöver jag fokusera mer på att identifiera och åtgärda buggar för att förbättra mina program och göra dem mer robusta. Dessa insikter kommer att vara avgörande i mina framtida projekt.