UMEÅ UNIVERSITET
Institutionen för datavetenskap
15 november 2013
Labbrapport

Laboration 5

Systemnära programmering 7.5hp

mfind_p

Namn Robin Lundberg
E-mail ens10rlg@cs.umu.se
Sökväg /home/ens10/ens10rlg/edu/sysprog/lab5

Handledare
Mikael Rännar, Ola Ringdahl

Innehåll

1 Problemspecifikation

Jag har implementerat en trådad sökalgoritm som ska kunna söker efter filer på hårddisken rekursivt ner i mappstrukturer. Målet är att programmet ska vara snabbare för datorer med flera kärnor än vad ett otrådad program som utför samma uppgift är. Detta kräver att trådarna belastas ungefär lika.

Programmet körs med kommandot `mfind_p [-t type] [-p nrthr] start1 [start2 ...] name`. `mfind_p` är programmet; `startx` är mappar som man ska börja söka rekursivt i; `name` är namnet på den fil man ska söka efter; `type` specifierar vilken typ av fil du vill hitta, man kan välja mellan d, f och l som innebär mapp, vanlig fil eller mjuk länk respektive; `nrthr` anger hur många trådar man vill använda för att köra programmet.

För att genomföra denna sökning ska varje tråd ges en mapp—från en *taskpool* som enabrt den får söka i—tråden ska sedan fylla på denna taskpool med mappar som den hittar i denna mapp; sedan ska den också leta efter det *namn* som användaren vill söka efter och sedan skriva ut den. Detta sker i en loop tills alla undermappar har sökts igenom.

För att undvika *deadlock* och *odefinierade beteenden* när trådarna utnyttjar gemensamt minne så måste trådarna synkroniseras. Trådarna måste även kunna hantera en tillfälligt tom taskpool och veta när dom kan avsluta.

2 Åtkomst och användarhandledning

Programmet ligger i mappen `/home/ens10/ens10rlg/edu/sysprog/lab5`. Programmet kompileras genom att köra `make mfind_p`; och körs med kommandot `mfind_p [-t type] [-p nrthr] start1 [start2 ...] name` där `type` är typ av fil som man vill söka efter; `nrthr` är antalet trådar man vill använda sig av; `startx` är vilka mappar man vill starta söka i; och `name` är det man vill söka efter.

3 Algoritmbeskrivning

Main programmet kommer sköta initialisering av variabler, mutex och tolkningen av inargument. Sedan så skapar huvudtråden N-1 trådar som kör funktionen `taskManager` som i sin tur kallar på `mfind_p`. Huvudtråden kommer sedan själv köra taskManager och när alla trådar är färdiga så städar huvudtråden upp efter processen. Det behövs fyra globala variabler för att sköta synkningen mellan trådarna och dom är: `waiting` som anger hur många trådar som inte kör `mfind_p`, utan väntar på att en task ska läggas till i taskpoolen; `blocked` som är en array där varje element anger huruvida den motsvarande tråd väntar på en task eller inte, detta är för att förhindra att en tråd ökar på `waiting` flera gånger då tråden inte kommer sluta loopa bara för att den inte har någon task; `quit` som signalerar till trådarna att nu kan dom avsluta funktionen för sökningen är färdig, detta kommer ske när `waiting` är lika med antalet trådar; och till sist en `taskpool` som innehåller relativa sökvägar till dom mappar som behövs gås igenom.

Programmet använder tre mutexar för att hantera synkroniseringen mellan trådar. `initThreadID` används för att ge varje tråd ett unikt id mellan 0 och $N-1$

om N är antalet trådar. `accessTaskPool` används för att förhindra att olika trådar försöker komma åt `taskpoolen` samtidigt vilket kan ge odefinierat beteende. `getTask` används för att kunna bestämma när trådarna kan avsluta.

3.1 taskManager

1. `count = 0`
2. Ge tråden ett unikt `id`.
3. Kör loop tills färdig.
   (a) Lås getTask.
      i. Lås accessTaskPool.
         A. `mapp = getFolderFromTaskPool()`
      ii. Lås upp accessTaskPool.
      iii. Om `mapp == NULL` och om `!blocked[id]`. Öka då på `waiting` med ett och sätt `blocked[id] == true`.
      iv. Om `mapp != NULL` så sätt `waiting = 0` till 0 och alla element i `blocked` till `false`.
      v. Om `waiting == N` så sätt `quit` till `true`
   (b) Lås upp getTask.
   (c) Om `mapp == NULL` och `quit == true`
      i. Gå ut ur loopen.
   (d) Om `mapp == NULL` och `quit == false`
      i. Börja om loopen.
   (e) Om `mapp != NULL`
      i. Kör `mfind_p` med `mapp` som inargument.
      ii. Öka på `count` med ett.
4. Skriv ut tråd-id och `count`.
5. Avsluta funktionen.

3.2 mfind_p

Funktionen `mfind_p` har som argument en sökväg till en mapp och två variabler som används för att bestämma om en fil är den man söker efter.

1. Öppna `mapp` på given sökväg.
2. Gå igenom alla filer i mappen.
   (a) Kolla om filen är den du söker efter. Skriv i så fall ut dess sökväg på standard output.
   (b) Om filen är en mapp och inte "."eller "..", gör följande
      i. Lås accessTaskPool
         A. Lägg till ny mapp i `taskpool`
      ii. Lås upp accessTaskPool
3. Stäng mappen.
4. Avsluta funktionen.

4 Systembeskrivning

4.1 Datastrukturer

4.1.1 node

En länkad lista används för att implementera en stack. I det här programmet behövs en stack för att spara dom mappar som programmet ska söka i, kallad `taskpool`.

4.2 Viktiga variabler

4.3 Funktioner

4.3.1 int main(int argc, char *argv[])

Läser in argument och initialiserar mutexar och variabler. Huvudtråden ser till att de andra trådarna kör `taskManager`. Huvudtråden kör sedan också `taskManager`. När alla trådar är klar med `taskManager` så städar huvudtråden upp och programmet avslutas.

4.3.2 void * taskManager(void *pArg)

Hanterar synkroniseringen mellan trådarna. Det som i slutändan görs är att kalla `mfind_p` tills alla mappar som ska sökas igenom; har blivit genomsökta.

4.3.3 void mfind_p(char *path, pcre *find, int type)

Kollar upp alla filer i gen givna mappen på `path`. Ifall filen är den som söks—d.v.s. den är av typen `type` och passar det reguljära uttrycket `find`—så skrivs sökvägen ut på standard output. Ifall filen är en mapp men inte "."eller ".."så sparas den i stacken (`taskpoolen`) för att någon tråd ska kunna använda den som argument i `mfind_p`.

4.3.4 void read_arguments(int argc, char *argv[], par_t *par)

Tolkar inargument och verifierar att dom är på rätt format.

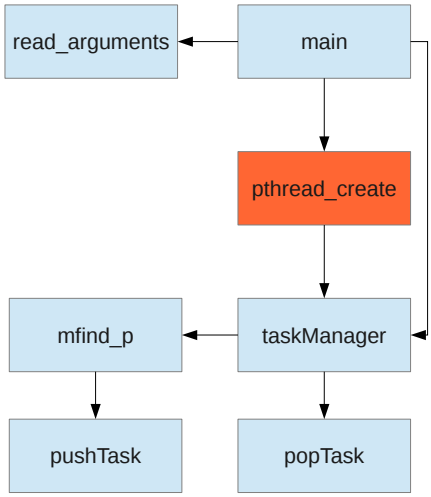
4.3.5 char * popTask(void)

Tar bort den mapp som ligger högst upp i stacken från stacken(`taskpoolen`) och returnerar den.

4.3.6 void pushTask(char* v)

Lägger till en mapp till stacken (`taskpoolen`).

4.4 Anropsdiagram



Figur 1: Anropsdiagram för programmet mfind_p.

5 Testkörningar

För att testa programmet har en känd mappstruktur skapats med följande skript.

```
#!/bin/sh
mkdir newtest
ln -s newtest linktest
cd newtest
mkdir a
ln -s a b
cd a
mkdir b c d
ln -s b a
cd b
mkdir a b c
chmod 000 c
touch d
cd a
touch a
cd ..
touch b/a b/c
```

Resultaten ska bli, och blir följande:

```
%./mfind_p -p2 newtest b
newtest/b
newtest/a/b
newtest/a/b/b
No permission to open newtest/a/b/c
140329278105344: 4
```

```
140329288730368: 4

%./mfind_p -p2 -t l newtest b
newtest/b
No permission to open newtest/a/b/c
139734134916864: 3
139734124291840: 5

%./mfind_p -p2 -t d newtest linktest b
newtest/a/b
linktest/a/b
linktest/a/b/b
No permission to open linktest/a/b/c
newtest/a/b/b
No permission to open newtest/a/b/c
140144795088640: 9
140144805713664: 7
```

6 Begränsningar

Man kan inte ange hur djup man vill söka i mappstrukturen. Programmet kollar inte om antalet trådar som användaren anger är ett korrekt nummer. Det finns inget som begränsar användaren att ange samma startmapp mer än en gång.

7 Problem och reflektioner

Det var väldigt svårt att få till synkronisering mellan trådar så att dom avslutar när dom är färdig; det tog mer tid än allt annat. Från början hade jag tänkt använda mig av semaforer men så vitt jag vet inte fanns inte så kallade *counting semaphores* implementerade, dvs. semaforerna kunde inte räkna upp till mer än ett. Vilket är ett problem om du exempelvis vill utnyttja dom för att hålla koll på antalet tasks som finns i taskpoolen; det jag istället gjorde var att använda mutexar och globala variabler.