

UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Laborationsrapport

November 11, 2014

Laboration 5
Systemnära Programmering 7,5 hp

mfind_p

Namn	Christer Jakobsson
E-mail	dv12c.jn@cs.umu.se
Sökväg	~dv12c.jn/edu/sysprog/lab5

Handledare

Mikael Rännar, Adam Dahlgren Lindström, Niklas Fries, Oskar Ottander

Contents

1	Problemspecifikation	1
1.1	Problemsammanfattning	1
1.2	Originalspecifikation	1
2	Åtkomst och användarhandledning	1
2.1	Filer som ingår i lösningen	1
2.2	Kompilering och körning	1
3	Systembeskrivning	2
3.1	Systemöversikt	2
3.1.1	linkedlist.c	2
3.1.2	linkedlist.h	2
3.1.3	mfind_p.c	2
3.2	Synkronisering	3
3.3	När kan en tråd avsluta	3
3.4	Går det snabbare med fler trådar?	3
3.5	Globala variabler	4
4	Testkörningar	4
4.1	Test 1	4
4.2	Test 2	4
4.3	Test 3	5
4.4	Test 4	5
4.5	Test 5	5
4.6	Test 6	5
5	Lösningens begränsningar	6

1 Problemspecifikation

Denna laborationsuppgift har handlat om att ta ett existerande program som skrivits under kursen och göra en trådad variant av det.

1.1 Problemsammanfattning

Denna laboration har gått ut på att ta det fungerande programmet *mfind* som skapades i laboration 3 och sedan göra så att istället för att använda en tråd för arbetet så används det flera trådar. *mfind*

1.2 Originalspecifikation

Specifikationen i sin helhet finns på

<http://www8.cs.umu.se/kurser/5DV088/HT14/lab/lab5/>

2 Åtkomst och användarhandledning

2.1 Filer som ingår i lösningen

I katalogen `~dv12cjin/edu/sysprog/lab5` ...

`mfind_p.c` Fil som är huvuddelen i programmet.

`mfind_p.h` Header fil för *mfind_p.c*

`linkedlist.c` Innehåller implementation av länkad lista.

`linkedlist.h` Header fil till *linkedlist.c*

2.2 Kompilering och körning

För att kompilera så använder man medföljande makefil och kör den genom att stå i samma mapp som filerna finns i och skriva *make*. Detta kommer att kompilera de filer som inte har kompilerats förut och skapa ett exekverbart program som heter *mfind_p*.

För att köra programmet så skriver man `./mfind_p` följt av de argument som programmet tar. Programmets måste få minst två argument, det första vilken sökväg som programmet ska börja söka ifrån, och namnet på den fil som ska sökas efter. *mfind_p* kan även ta två stycken extra argument. Det ena är för att specificera vilken typ av fil som ska sökas efter. Vanlig fil, en mapp eller en länk. Det andra argumentet specificerar hur många trådar programmet ska använda sig av för att göra sökningen.

Exempel körningar:

- `./mfind_p /home bin`: Söker efter alla sorters fil med en tråd.
- `./mfind_p -t d /home bin`: Söker bara efter mappar.
- `./mfind_p -p 4 /home bin`: Söker efter alla sorters filer med 4 trådar.
- `./mfind_p -t 4 -t l /home bin`: Söker efter alla filer som heter *bin* och är av typen länk med fyra trådar.

`-t` specificerar att det som kommer efter är vilken filtyp som ska sökas efter och `-p` specificerar att det som kommer efter är hur många trådar som ska söka.

3 Systembeskrivning

3.1 Systemöversikt

3.1.1 `linkedlist.c`

Denna filen innehåller kod som används för att skapa en datastruktur som är en länkad lista. Det är en lista med noder som kan innehålla godtycklig data (`void*`), och en pekare till nästa nod i listan. Detta gör att en nod bara känner till den nod som kommer efter sig.

3.1.2 `linkedlist.h`

Detta är en `.h` fil som hör till `linkedlist.c` som innehåller alla funktionsdeklARATIONER.

3.1.3 `mfind_p.c`

`mfind_p` är en påbyggnad av programmet `mfind`. Tanken med programmet är att det ska söka igenom filsystemet efter en viss typ av fil. Det som har lagts till i `mfind_p` är att användaren kan välja hur många trådar som ska göra sökningen samtidigt. Detta är huvud programmet som genomför sökningen i filsystemet, programmet använder sig av `getopt` för att hantera de olika argumenten som skall stödjas. om användaren skriver `-t` så kommer via `getopt`, värdet på en variabel som säger vilken typ av fil vi ska söka efter att sättas. Även om skickat `-p` så kommer programmet på ett liknande sätt sätta hur många trådar som ska köra sökningen. Efter att argumenten har behandlats och inga fel har upptäckts i argumenten så läggs alla sökvägar som användaren har angett att den vill söka ifrån in i en länkad list, denna lista är delad mellan alla trådar som exekverar samtidigt och de kommer att turas om att plocka ut sökvägar ur listan för att sedan söka där. Om en tråd under sin sökning hittar en mapp så läggs dess sökväg in i listan och denna del är då gjord så att det bara kan vara en tråd i funktionen samtidigt.

Programmet kommer att, om användaren vill söka med mer än en tråd skapa en trådpool av antalet som användaren har angett som argument minus ett.

Detta för att även huvudprogrammet ska söka så det kommer bara att behövas skapa antal trådar minus ett, därefter så startas alla nyskapade trådar och börjar köra metoden *searchDir* och sist kommer huvud programmet att påbörja sin sökning. när huvudprogrammet har avslutat sin sökning så ställer det sig och väntar på att trådar som skapats är blir klara med sökningen, därefter så skrivs det ut hur många sökvägar varje tråd har sökt igenom.

3.2 Synkronisering

3.3 När kan en tråd avsluta

För att programmet ska funka optimalt så måste trådarna ha ett smart sätt att veta när dom är klara. Det initiala är att en tråd säger sig vara klar när listan är tom. Men detta fungerar inte för att en tråd kan märka att listan är tom och avsluta och därefter så har en tråd som håller på med en sökning hittat en ny sökväg och därmed är inte listan tom och en tråd har avslutat utan att jobbet är klart.

För att åstadkomma detta så har jag använt mig an semaforer och en uppräknare, uppräknaren säger hur många trådar som anses sig vara färdiga och om den är lika med antalet trådar som körs innebär det att ingen tråd kan hitta flera mappar och sökningen är slut För att se till så att trådarna väntar om listan är tom så har jag använt en semafor som initieras till ett. För varje gång en tråd lägger till en ny sökväg i listan så postar den detta i semaforen som ökar sitt värde med ett. För varje gång som en tråd ska söka så anropar den *wait* på semaforen och kan fortsätta så länge semaforen inte blev noll.

Detta medför då att trådarna kommer att kolla i listan om det finns jobb innan tråden plockar en sökväg ur listan. Om listan var tom så kommer tråden att vänta tills en annan tråd har hittat en sökväg och lagt till den i listan.

3.4 Går det snabbare med fler trådar?

För att undersöka hur exekveringstiden ändrades så använde jag mig av två givna filer där det var möjligt att hämta den exakta tiden. Den kod som borde vara del i tidtagningen är både skapandet av trådarna och startandet av dem.

Jag körde tester där jag sökte igenom en mapp hierarki både med 1 tråd, 10 trådar, och 20 trådar. Hierarkierna jag testade mot var av olika storlek och jag testade även att söka efter olika filer som förekom olika ofta. Resultatet blev då som följer, där jag körde varje sökning tio gånger och tog medelvärdet av tiderna.

1 Tråd
1.88
1.94
4.0
Utifrån dessa tider för de olika metoderna och även en del odokumenterade tester så kommer jag ti

3.5 Globala variabler

Lösningen innehåller en del globala variabler, detta för att alla trådar ska kunna komma åt dem.

`dirList` Den länkade listan, alla trådar måste kunna hämta och lägga till sökvägar.

`nrOfThreads` En variabel som säger hur många trådar som ska köras, måste kunna nås av alla trådar för att dom ska kunna avgöra om dom är klara.

`threadsComplete` En global variabel som varje tråd ökar värdet med ett medans dom väntar och sänker värdet med ett om dom har arbete.

`fileToSearch` En variabel som innehåller namnet på den fil som ska sökas efter.

`lock` Ett mutex lås får att synkronisera trådar i vissa funktioner.

`fileType` En variabel som berättar vilken sorts fil vi söker efter.

`semafor` En semafor som alla trådar använder sig av.

`threadSearchCount` En array som innehåller hur många sökvägar varje tråd har sökt igenom.

4 Testkörningar

4.1 Test 1

```
itcny:~/edu/sysprog/lab5> ./mfind_p -p 3 /pkg/comsol comsol
/pkg/comsol/bin/comsol
/pkg/comsol/4.1/configuration/comsol
/pkg/comsol/4.1/bin/comsol
/pkg/comsol/4.1/bin/glnx86/comsol
/pkg/comsol/4.1/bin/glnxa64/comsol
/pkg/comsol/4.1/doc/html/comsol
/pkg/comsol/4.1/doc/pdf/comsol
Thread searched 87 directories
Thread searched 210 directories
Thread searched 92 directories
```

4.2 Test 2

```
itchy:~/edu/sysprog/lab5> ./mfind_p -p 3 /pkg/comsol/ /usr/local/bin/ comsol
/pkg/comsol//bin/comsol
/pkg/comsol//4.1/configuration/comsol
/pkg/comsol//4.1/bin/comsol
/usr/local/bin//comsol
/pkg/comsol//4.1/bin/glnx86/comsol
/pkg/comsol//4.1/bin/glnxa64/comsol
/pkg/comsol//4.1/doc/html/comsol
/pkg/comsol//4.1/doc/pdf/comsol
Thread searched 202 directories
Thread searched 104 directories
Thread searched 84 directories
```

4.3 Test 3

```
/pkg/comsol//4.1/configuration/comsol
/pkg/comsol//4.1/doc/html/comsol
/pkg/comsol//4.1/doc/pdf/comsol
Thread searched 84 directories
Thread searched 103 directories
Thread searched 203 directories
```

4.4 Test 4

```
itchy:~/edu/sysprog/lab5> ./mfind_p -p 3 /opt/comsol/ /usr/local/bin/ comsol
cannot read dir /opt/comsol/: No such file or directory
Could not close file: Invalid argument
/usr/local/bin//comsol
Thread searched 1 directories
Thread searched 0 directories
Thread searched 0 directories
```

4.5 Test 5

```
itchy:~/edu/sysprog/lab5> ./mfind_p -p 3 -t 1 /pkg/comsol/ /usr/local/bin/ comsol
/usr/local/bin//comsol
Thread searched 154 directories
Thread searched 77 directories
Thread searched 159 directories
```

4.6 Test 6

```
itchy:~/edu/sysprog/lab5> ./mfind_p -p 3 ~mr/src cdecl.c
cannot read dir /Home/staff/mr/src: Permission denied
Could not close file: Invalid argument
Thread searched 0 directories
Thread searched 0 directories
Thread searched 0 directories
```

5 Lösningens begränsningar

I detta avsnitt beskrivs alla begränsningar som lösningen av uppgiften innehåller. Detta innefattar även funna begränsningar som strider mot specifikationen.