Laboration 2 - P-Threads

DV1556

Simon Fredholm, Game Programming, Blekinge Institute of Technology
2018

Innehåll

Inledning

I den här laborationen ska jag experimentera och lära mig om flertrådade program. Jag ska ta reda hur man använder sig av flera trådar och hur man ska lösa kritiska sektioner med hjälp av mutexer.

Metod

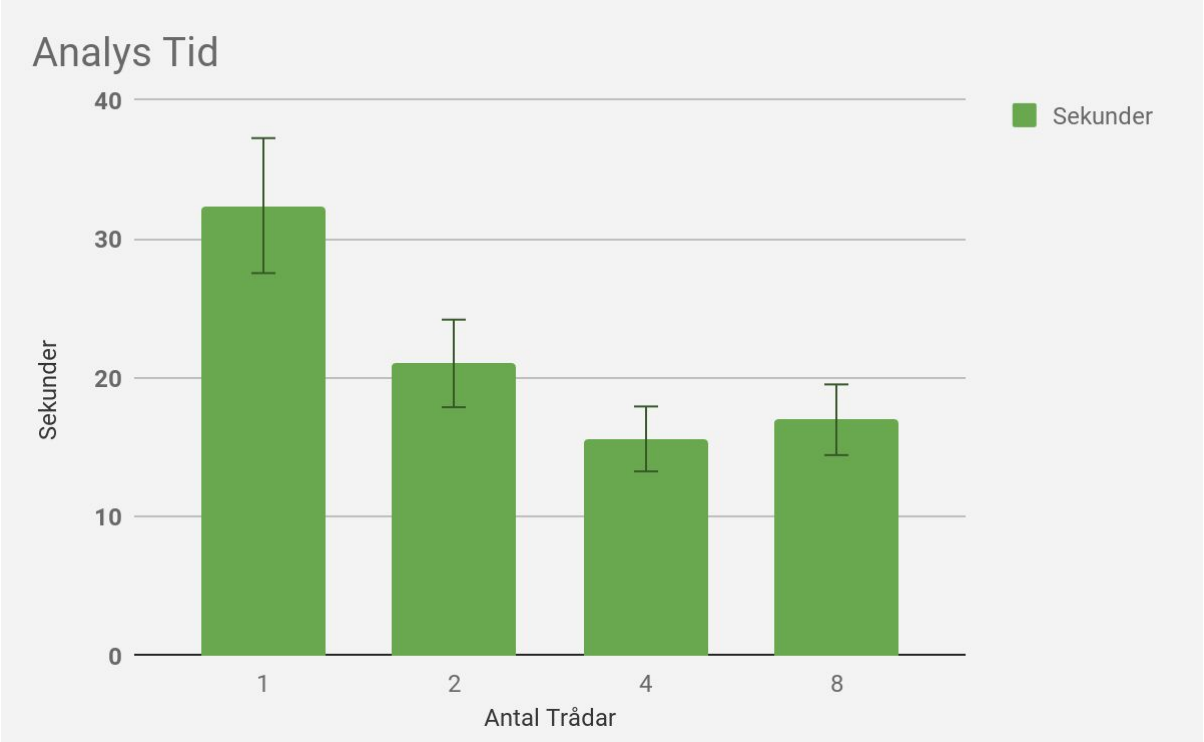
Jag använde mig av ett Ubuntu 64-bitars system med två CPU-kärnor. För att experimentera samt bekanta mig av olika problem så modifierade jag källkod som vi fick till hjälp för att underlätta laborationen. I laborationen så kommer vi att använda oss av POSIX Threads API för att skapa flera trådar i samma process.

Resultat

Jag börjar med att testköra den sekventiella implementationen av programmet med en stor text-fil. När texten läses in så är det omöjligt för mig att stänga av programmet förutom att göra en force-exit genom att stänga ner hela terminalen. Det finns alltså ingen tid för processen att ta hand om användarens input. Vi löser detta problemet genom att låta beräkningar av text-filen utföras av en annan tråd. Nu kan vi avsluta programmet när vi vill, även om vi samtidigt räknar ut resultatet från text-filen.

När vi ändå modifierar funktioner så ändrar vi även frekvensanalysen till att kunna skapas i en separat tråd. Vi gör det också möjligt för att dela upp frekvensanalyser i så många trådar vi vill. Innan vi gör detta så måste vi även ändra så att alla funktioner som frekvensanalysen använder är tråd-säkra, vilket betyder att programmet inte ska kunna krascha om flera trådar använder samma funktion under samma tillfälle. Vi hittar en kritisk sektion, när funktionen *mergeFreqTable* körs. Den behöver skyddas av mutexer eftersom funktionen ändrar på en global variabel.

Vi testar det uppdaterade programmet (*se bilaga 1*) på en text-fil på 300MB med 1, 2, 4 och 8 trådar och ser hur hastigheten påverkas. Vi kan se i diagrammet (*se figur 1*) så är en tråd överlägset långsammast i exekveringstid. Två minskar tiden det tar med ungefär en tredjedel medans fyra trådar halverar tiden. Åtta trådar tar längre tid att köra än fyra trådar, troligen eftersom det blir onödiga kontext-byten som kostar overheadtid.



Figur 1 - Diagram över tid passerad under frekvensanalys

Diskussion

Jag har lärt mig hur viktigt flertrådade program är för prestanda. Att låta viktiga saker som I/O fortsätta att fungera även under långa uträkningar är extremt viktigt för alla typer av program. Jag räknade med liknande resultat av analys-tiderna, förutom att redan efter åtta olika trådar så tog overhead för mycket onödig tid.

Bilagor

Bilaga 1, seq_freq_an.c