



IT-Universitetet
i København

OBLIGATORISK OPGAVE # 2 I BOSC

Ooperativsystemer og C

Author:

Tom Mørk Christensen

Jonas Elbækgaard Jørgensen

ITU-mail:

TMCH@ITU.DK

JELB@ITU.DK

9. oktober 2014

Indhold

1	Delopgave 2	
	– Multitrådet FIFO buffer som kædet liste.	2
1.1	Implementationen, overordnet set	2
1.2	De specifikke løsninger	2
1.2.1	Tilføjelse af elementer	2
1.2.2	Fjernelse af elementer	2
1.2.3	Trådsiking	3
1.3	Fejl og mangler	3
1.4	Test	3
1.4.1	En anden tilgang til test	4

1 Delopgave 2

– Multitrådet FIFO buffer som kædet liste.

Denne delopgave har til formål at vise, at vi har forståelse for hvordan hægtedelister fungerer samt hvordan disse kan implementeres i C. Udover at demonstrerer vores forståelse for hægtedelister tjener opgaven det formål at demonstrer hvordan nogle af risiciene ved flertrådet systemer kan forebygges.

1.1 Implementationen, overordnet set

Implementationen af vores løsning til denne delopgave begrænser sig til klassen `list.c`. I forbindelse med test af vores implementation har vi også ændret filen `main.c`, hvor vi har introduceret to nye funktioner `test()` og `add(void *param)`. Vi vil beskrive implementationen af disse funktioner samt deres formål i afsnit 1.4 på side 3.

I `list.c` kan vores implementation splittes i tre dele.

1. Tilføjelse af elementer til listen. Implementationen af denne funktionalitet begrænser sig til funktionen `list_add(List *l, Node *n)` og har til formål at hægte det nye elemnt `n` bagerst på listen.
2. Fjernelse af elementer fra listen. Implementationen af denne funktionalitet begrænser sig til funktionen `list_remove(List *l)` og har til formål at fjerne og returnere de foreste element i den hægtede liste.
3. Trådsikring af listen, ved hjælp af gensidig udelukkelse. I modsætning til de to foregående dele er denne del spredt over flere af implementationens funktioner. Dette skyldes at mutexen skal oprettes sammen med listen, men låses og frigives i forbindelse med metodekald.

1.2 De specifikke løsninger

1.2.1 Tilføjelse af elementer

For at tilføje et element til en liste skal vi bruge en pointer til både listen og til det element som skal hægtes på listen. Når et element tilføjes til en liste hægtes dette på listens bagerste element således at de ældste elementer altid ligger tættest på listens start. I vores implementation ændre vi listens bagerste element således at det peger på det nye element. Her efter undersøger vi om den knude der tilføjes til listen indgår i en sekvens af hægtede knuder. Er dette tilfældet traversere vi listen af knuder igennem indtil vi når det sidste element. Når vi har nået det sidste element ændre vi den hægtede listes `last` pointer således at denne nu også peger på det nye slut element. Herefter opdatere vi listens længde med antallet af nye elementer.

1.2.2 Fjernelse af elementer

Ved fjernelse af elementer starter vi med at gennem den hægtede listes `rod` element til en variabel for at simplificere arbejds processen. Da roden ikke må fjernes tjekker vi om `next` pointeren på roden peger på en værdi. Er dette tilfældet ændres rodens `next` pointer så den perger på det element der ligger efter elementet der skal fjernes, hvorefter længden reduceres med en og det fjernede element returneres. Peger rodens `next` pointer ikke på en værdi er listen tom og værdien `NULL` returneres.

1.2.3 Trådsikring

For at introducere trådsikring til vores program har vi gjort brug af én `mutex`. Mutexen har til formål at sikre at kun en tråd kan eksekvere en given kommando ad gangen. Dette fungerer ved at den tråd der kun den tråd der i en given situation har låst mutexen kan eksekvere den efterfølgende kode, mens andre tråder der ønsker at eksekvere den samme kode, eller eksekvere anden kode, som også er beskyttet af mutexen, bliver holdt tilbage indtil låsen på mutexen fjernes og en ny tråd kan låse den.

I vores implementation har vi gjort brug af én mutex som bliver brugt både når tråde ønsker at tilføje eller fjerne et element fra listen. Det gør sig gældende for både `list_add` og `list_remove` funktionen at når en tråd træder ind i funktionen skal den låse mutexen før end at den får lov at tilføje eller fjerne noget, og tilsvarende frigive mutexen når operationen er udført. Dette betyder at hvis en anden tråd er i gang med at redigere listen, må tråden vente på at mutexen bliver frigivet før den kan udføre sine ændringer på listen.

Mutexen bliver oprettet sammen med listen og ødelægges sammen med listen ved brug af funktions kaldet `list_destroy`.

1.3 Fejl og mangler

Selvom at vi i vores implementation gør brug af mutex til at sikre at tråde ikke tilgår listens elementer samtidigt, og at brugen af kun én mutex skulle forhindre eventuelle deadlocks i at opstå er der stadig en måde hvorpå vores system ville gå i knæ. Hvis en bruger tilføjer en sekvens af knuder som danner en løkke vil vores implementation træde ind i en uendelig løkke da den ved indsættelse af nye elementer læder efter det sidste element i rækken af elementer.

Vi har gjort os nogle overvejelser om hvordan denne situation kunne afværges og er fundet frem til at den bedste løsning ville være at gemme det foreste af de nye elementer til en variabel og så tjekke at ingen af de efterfølgende elementer peger tilbage på dette element. Denne løsning til sikre at en liste hvor bagerste element tilbage på det forreste ville blive opdaget, men samtidig udelukker det ikke situationer hvor fx bagerste element peger tilbage på det andet element i listen. Den eneste måde hvorpå vi har kunnet se at det er muligt at afvære en sådan situation, hvor et element peger tilbage på et tidligere, er ved at gemme alle nye elementer til variabler og sikre at ingen af disse bliver peget på senere i listen. Vi har valgt ikke at give os i kast med at implementere en sådan løsning da det ikke virker til at være opgavens primære mål, men det er bestemt en situation der er værd at overveje hvis programmet skal anvendes i større stil.

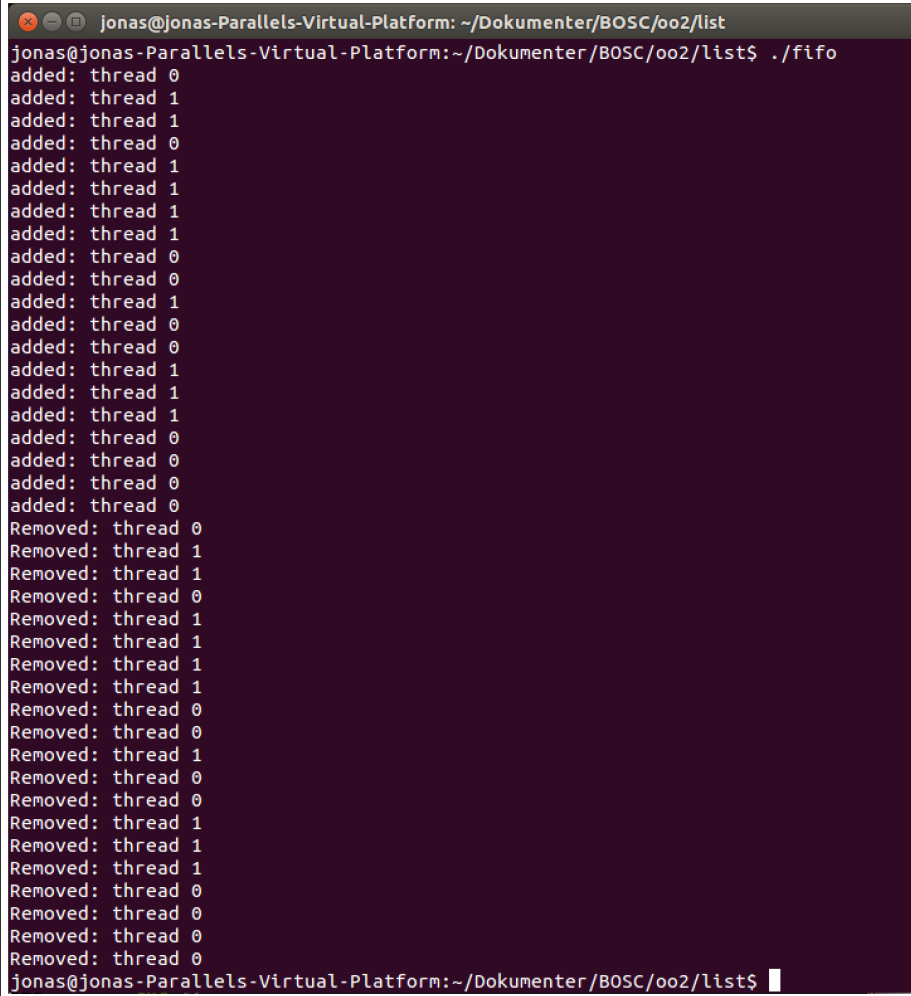
En anden ting som fangede vores opmærksomhed under implementationen af denne opgave var muligheden for at gøre brug af 2 mutex objekter. Så længe at den hægtede liste ikke er tom der er i og for sig ikke noget i vejen for at en process tilføjer et element mens en anden fjerner et element da de arbejder på hver sin del af listen og derfor ikke skal bruge de samme objekter. Den eneste situation hvor dette ikke er tilfældet er hvis listen er tom eller kun holder et element, da begge processer i såfald ville arbejde på listens rod og derved risikere at forårsage en race-condition. En tilgang til dette kunne være kun at låse `list_add` og `list_remove` enkeltvis så længe at listen holder to eller flere elementer, men låse begge funktioner hvis listen er tom eller kun holder et element.

1.4 Test

For at teste at vores implementation kan anvendes af flertrådet programmer har vi skrevet en test funktion i `main.c`. Funktionene opretter to tråde som begge har tilopgave tilføje ti

elementer til listen. For at sikre at begge tråde forsøger at tilgå listen samtidig har til tilføjet et kald til funktionen `sleep(1)` som får den aktuelle tråd til at sove i et sekund.

For at kontrollere at trådene ikke har overskrevet hinandens ændringer afsluttes funktionen med at printe alle elementerne i den hægtede liste til terminalen. Af figur 1 ses det at alle 20 elementer er blevet tilføjet til listen, og at trådene på skift har haft adgang til listen.



```
jonas@jonas-Parallels-Virtual-Platform: ~/Dokumenter/BOSC/oo2/list
jonas@jonas-Parallels-Virtual-Platform:~/Dokumenter/BOSC/oo2/list$ ./fifo
added: thread 0
added: thread 1
added: thread 1
added: thread 0
added: thread 1
added: thread 1
added: thread 1
added: thread 1
added: thread 0
added: thread 0
added: thread 1
added: thread 0
added: thread 0
added: thread 0
added: thread 1
added: thread 1
added: thread 1
added: thread 0
added: thread 0
added: thread 0
added: thread 0
Removed: thread 0
Removed: thread 1
Removed: thread 1
Removed: thread 0
Removed: thread 1
Removed: thread 1
Removed: thread 1
Removed: thread 1
Removed: thread 0
Removed: thread 0
Removed: thread 1
Removed: thread 0
Removed: thread 0
Removed: thread 0
Removed: thread 1
Removed: thread 1
Removed: thread 1
Removed: thread 0
Removed: thread 0
Removed: thread 0
jonas@jonas-Parallels-Virtual-Platform:~/Dokumenter/BOSC/oo2/list$
```

Figur 1: Udskrift af test resultat til terminalen.

1.4.1 En anden tilgang til test

En anden mulig måde at teste hvorvidt vores hægtedeleliste understøtter at flere tråde forsøger at tilgå den er ved brugen af funktionskaldet `pthread_mutex_trylock()` som returnere en fejlmeddelelse hvis den givne mutex allerede låst. På denne måde kan man få de enkelte tråde til at printe en besked til terminalen når de er tvunget til at vente på at låsen på mutexen bliver frigivet.