Tredje aflevering
OSM

Davy Eskildsen og Mads C. Lund

March 5, 2012

1 Låse og betingelsesvariabler i Buenos

1.1 Implementation af lås

Vi har definere typen lock_t som et struct indeholdende en boolsk værdi for om der er låst eller ej, og en spinlock til af sørge for, at kun en bruger låsen af gangen.

Dertil funktionerne lock_reset, lock_acquire og lock_release:

lock_reset sætter låsen til at være åben, initalisere spinlock'en og retunere 0, vi kan ikke se hvordan der kunne ske en fejl, så den retunere aldrig en fejlstatus.

I lock_acquire og lock_release bruger vi spinlock'en til at sørge for, at vi er den enste der prøver at få låsen lige nu. Hvis den er låst ligger vi os til at sove indtil der bliver åbnet for låsen.

1.2 Betingelsesvariabler

Vi har ikke kunne finde på, hvad vi skulle bruge typen cond_t udover at bruge den til at sove på. Derfor har vi lavet den som en tom struct.

I condition_init er der ikke noget arbejde at lave, da vi har valgt at bruge en tom struct som beskrevet ovenfor. Derfor er det eneste som init laver er at sætte cond = cond for at undgå at der kommer "warnings" med ubrugte argumenter. Vi er begge enige om at det virker mærkeligt hvis der ikke skulle være noget at bruge cond_t og condition_init til, men vi kan samtidigt ikke finde nogen fejl i vores implementering og har derfor valgt ikke at gøre mere ved det.

condition_wait ligger kalderen til at sove i en sovekø og låser låsen op. Denne har vi implementeret med sleepq.

condition_signal og condition_broadcast kalder bare henholdvis sleepq_wake og sleepq_wake_all for at vægge en eller alle der er i kø.

2 CREW-problemet

2.1 Implementation af simpel CREW

I filen `CREW_test1.c` har vi implementeret en simpel implementation af CREW. Vi har fulgt strukturen beskrevet ved beskrevet ved forelæsningen i de medfølgende slides. Til forskel har vi brugt pthread_mutex_t i stedet for semafore, da vi kun har brug for en binær lås.

Hvis en tråd skal skrive til en ressource, R, er den afhænging af at have `wlock`, tilsvarende for en læse/skrive-lås som netop sikre, at der ikke er flere som kan tilgå R.

For at opnå samtidig læseadgang, sørger den første læse-tråd for at låse `wlock`. Efterfølgede læse-tråde kan nu tilgå R uden den bliver ændret. Når der ikke er flere i gang med at læse, sørger den sidste tråd for at åbne `wlock` igen. Håndteringen af antallet af læs-tråde håndteres af en fælles tællevariabel `readers`. Da denne også er en fælles ressource, er det også nødvendigt at beskytte denne mod samtidig adgang. Dette gøres også med en lås, `rcountlock`.

Skrivetråd

1. Lås `wlock`
2. Foretag skrivning
3. Åben `wlock`

Læsetråd

1. Lås `rcountlock`
2. Tæl `readers` op
3. Hvis det er den første læser: Lås `wlock`
4. Åben `rcountlock`
5. Foretag skrivning
6. Lås `rcountlock`
7. Tæl `readers` ned
8. Hvis det er den sidste læser: Åben `wlock`
9. Åben `rcountlock`

2.2 CREW med prioritering af skrive-tråde

Da den forrige implementation resultere i at skrive-trådene kommer til at sulte følger her en metode så vi imødekommer dette problem.

Vi kan betragte linje 1 for skrivetråden og linje 1-4 for læsetråden som indgange til R. Hvis skriveren kan spærre indgangen for læsere, mens den venter på at kunne låse `wlock` er problemet løst. Dette kan gøre med at skulle have lås for at komme gennem indgangen.

Skrivetråd

1. Lås `access`
2. Lås `wlock`
3. Åben `access`
4. Foretag skrivning
5. Åben `wlock`

Læsetråd

1. Lås `access`
2. Lås `rcountlock`
3. Tæl `readers` op
4. Hvis det er den første læser: Lås `wlock`
5. Åben `rcountlock`
6. Åben `access`
7. .....

3 Analyse af de forskellige metoder

Vi har testet de forskellige implementationer af CREW ved at kigge på antallet af skrivninger og læsninger over tid. Tiden er målt med antal CPU clock-cykler. Til måling har vi brugt antallet af clock-cykler siden systemstart. Vores måledata baserer sig på 50 kørsler af testprogrammerne med 10 læse- og 10 skrivetråde som hver kører 10 iterationer. Det betyder sammenlagt 20000 læsninger og skrivninger.

Figur 1, 2, 3 og 4 indeholder histogrammer for vores to implementationer af CREW og det udleverede eksempel som bruger rwlock. Histogrammerne er opdelt i intervaller af $2 \cdot 10^6$ clock-cykler. Farverne indikere antallet skrivninger (blå) i forhold til læsninger (rød).

I figur 1 ses det tydeligt, at skriverne bliver sultet og først kommer til efter der ikke er flere læsninger. Samtidig ses det også, at denne implementation medføre at der kan foretages mange flere læsninger på kortere tid end for de to andre. Figur 2 giver som forventet en mere fair fordeling. Figur 3 viser resultaterne for rwlock-implementationen. Her skal det bemærkes at pthreads-rwlock desuden bruger en prioriteringsfaktor mellem læsning og skrivning, som afhænger af implimentationen. Figur 3 viser en køsel på Mac OS mens figur 4 viser en kørsel på Tuxray. Det bemærkes at figur 3 kørere ligesom vores implementation, mens figur 4udnytter bedre at den kan foretege samtidig læsning.

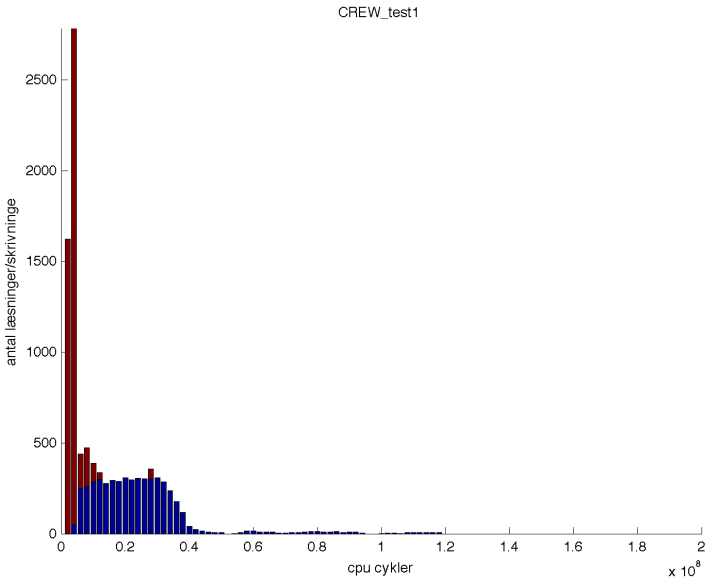

Figure 1: CREW med sultning af skrivere. Rød: læsninger. Blå: Skrivninger

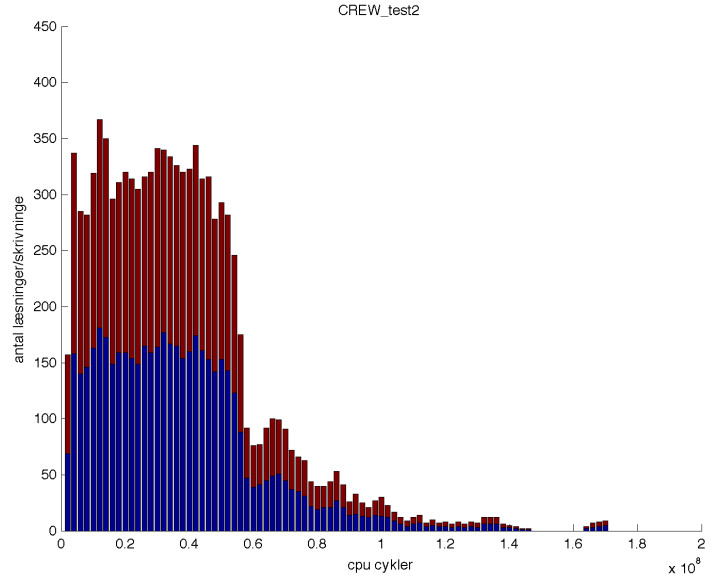

Figure 2: CREW med prioritering af skrivere. Rød: læsninger. Blå: Skrivninger

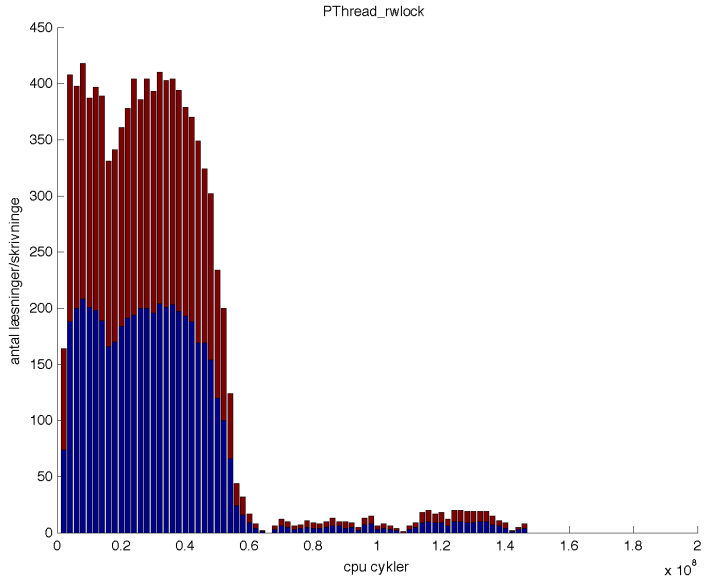

Figure 3: CREW med pthreads rwlock på Mac OS. Rød: læsninger. Blå: Skrivninger

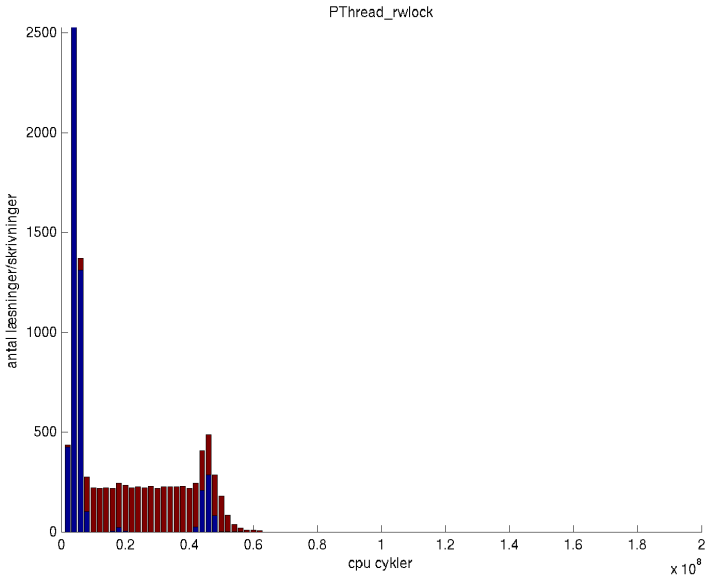

Figure 4: CREW med pthreads rwlock kørt på tuxray. Rød: læsninger. Blå: Skrivninger