LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Synkronisering

Klas Arvidsson, Daniel Thorén, Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018

2020-03-06

1 Mål

Filsystemet i Pintos fungerar i nuläget bra så länge inte flera trådar (eller processer) försöker använda det samtidigt. Det är såklart väldigt vanligt att flera program eller trådar försöker läsa från eller skriva till filer i filsystemet, så vi måste se till att filsystemet fungerar korrekt även i dessa fall, vilket är målet med laborationen.

Information om hur filsystemet är uppbyggt och hur det fungerar finns under rubriken ”Filsystem” i Pintos-Wiki.

2 Uppgift (99% kodförståelse, 1% modifiering)

Från dina systemanrop använder du egna datastrukturer (arrayer och/eller listor), och anropar flera delsystem, som är helt eller delvis osynkroniserade. Om processerna/trådarna råkar exekvera i ”fel” ordning, med trådbyten på ”fel” ställen kommer din implementation att gå sönder.

Gå igenom frågeställningarna nedan och se vilken kod som kommer att exekveras i de olika fallen. Notera vilka funktioner som används och vilka variabler och datastrukturer som kommer att användas samtidigt. **Synkronisera sedan koden för filsystemet.**

Du kan utifrån beskrivningen i Pintos-Wiki under rubriken ”Filsystem” och speciellt underrubriken ”Struktur” avgöra vilka filer som är intressanta. Det finns även en bild över hur de olika modulerna i filsystemet beror på varandra på kurshemsidan. Det är viktigt du tänker efter först för att få så enkel lösning som möjligt. Det finns flera godkända lösningar av varierande kvalité.

Synkronisera även alla egna datastrukturer och funktioner där så krävs.

I denna uppgift ska du titta speciellt noga på *variabler som kan nås av flera trådar* samtidigt, d.v.s. antingen *globala variabler* eller variabler som flera trådar kan ha en *pekare* till. Identifiera kritiska sektioner och synkronisera med lås (eller semafor om så krävs). Conditions behöver du bara om du vill vänta på att en synkroniserad (låst) variabel skall uppfylla ett visst villkor. Tänk på att globalt placerade lås skapar onödiga köer för alla trådar som använder dem. Om det är möjligt, använd ett lås per resurs som delas.

Synkronisering av läsning och skrivning till filer skall du lämna till nästa uppgift, alternativt tillfälligt använda ett enkelt lås i *inode_read_at* och *inode_write_at*. Ett enkelt lås är hursomhelst en bra start på nästa uppgift.

Var ska man synkronisera?

En till synes enkel lösning på problemet skulle vara att lägga all synkronisering direkt i hanteringen av systemanrop. Detta är dock inte en bra lösning av två anledningar: Först och främst använder andra delar av Pintos exempelvis filsystemet, så med den här lösningen kommer Pintos att kunna anropa filsystemet samtidigt som program gör det, vilket kan orsaka problem. Den andra stora anledningen är att detta ofta är ineffektivt. Att ha ett (eller ett fåtal) lås direkt i systemanropen innebär att bara en process i hela systemet kan göra ett systemanrop åt gången. Men att två processer läser olika filer är ju helt ofarligt, så det vill vi tillåta. Försök därför att lägga synkroniseringen så nära de resurser som är delade som möjligt. Försök också att i den mån det är möjligt baka in synkroniseringen i de funktioner som finns, så att det inte går att glömma att låsa något innan man anropar en viss funktion.

3 Några frågeställningar

1. Katalogen är tom. Två processer lägger till filen ”kim.txt” samtidigt. Är det efteråt garanterat att katalogen innehåller endast en fil ”kim.txt”?
2. Katalogen innehåller en fil ”kim.txt”. Två processer tar bort ”kim.txt”, och en process lägger samtidigt till ”kam.txt”. Är det efteråt garanterat att katalogen innehåller endast fil ”kam.txt”?
3. Systemets globala *inode*-lista är tom. Tre processer öppnar samtidigt filen ”kim.txt”. Är det garanterat att *inode*-listan sedan innehåller endast en cachad referens till filen, med *open_cnt* lika med 3?
4. Systemets globala *inode*-lista innehåller en referens till ”kim.txt” med *open_cnt* lika med 1. En process stänger filen samtidigt som en annan process öppnar filen. Är det garanterat att *inode*-listan efteråt innehåller samma information?
5. Free-map innehåller två sekvenser med 5 lediga block. Två processer skapar samtidigt två filer som behöver 5 lediga block. Är det efteråt garanterat att filerna har fått var sin sekvens lediga block?
6. Katalogen innehåller en fil ”kim.txt”. Systemets globala *inode*-lista innehåller en referens till samma fil med *open_cnt* lika med 1. Free-map har 5 block markerade som upptagna. En process tar bort filen ”kim.txt” samtidigt som en annan process stänger filen ”kim.txt”. Är det efteråt garanterat att *inode*-listan är tom, att free-map har 5 nya lediga block, och att katalogen är tom?
7. Katalogen innehåller en fil ”kim.txt”. En process försöker öppna filen samtidigt som en annan process tar bort filen ”kim.txt” och skapar sedan en ny fil ”kam.txt”. Är det efteråt garanterat att den första processen antingen lyckades öppna filen ”kim.txt”, eller att den misslyckades? Eller kan det bli så att den råkar öppna ”kam.txt” i stället?
8. Liknande frågor skall du själv ställa dig i relation till din process-lista och till din(a) fil-list(or).

3.1 Testa din implementation

Efter denna laboration varje enskilt test i de inbyggda testerna i Pintos alltid fungera, eller aldrig fungera (förutom testet `klaar/pfs`). Det ska alltså inte finnas några test som fungerar ibland, men inte alltid. Om något fungerar bara ibland är synkroniseringen fel. Det krävs att alla testerna körs igenom många gånger.

I `examples/pfs.c` finns ett testprogram (som också är en del av de automatiska testerna). Det skall fungera att köra utan att det kraschar, men skall skriva ut felmeddelanden i form av ordet *INCONSISTENCY* (givet att du inte har synkroniserat `inode_write_at` eller `inode_read_at`). Du startar testet med följande kommandorad:

```
pintos -v -k -T 120 --fs-disk=2 \
       -p ../examples/pfs -a pfs \
       -p ../examples/pfs_writer -a pfs_writer \
       -p ../examples/pfs_reader -a pfs_reader \
       -g messages -- -f -q -F=2000 run pfs
```

Om inga *INCONSISTENCY* skrivs ut har du antagligen varit lite för duktig på att synkronisera koden (bra!). Testa också att öka parametern `-F=2000` till en lite större siffra (den ökar sannolikheten för olägliga trådbyten, men det går inte att dra upp den hur långt som helst). I senare uppgift skall du lösa problemet med inconsistency med *ett speciellt lås* som tillåter att filer läses samtidigt. Se till att filer kan skrivas och läsas samtidigt så du ser att inconsistency uppstår och därmed kan se när nästa uppgift är löst (ingen inconsistency uppstår).

Notera att det på vissa system är kan vara att se inconsistency (detta beror på vilken hårdvara du har bland annat). Du kan i så fall tillfälligt lägga till anrop till `timer_msleep(1)` inuti looparna i `inode_read_at` och

`inode_write_at`. Detta gör att vissa typer av fel blir mer sannolika, men visar tyvärr inte allt som kan gå fel.