Deluppgift 21 Reders-writers lock

Bakgrund

När en process läser eller skriver till en fil samtidigt som en annan process skriver till samma fil, eller när en process skriver till en fil samtidigt som en annan process läser från samma fil, så kan det uppstå fel genom att processen som läser kanske läser en del av filen som den såg ut innan den ändrades, och en annan del som den såg ut efter ändring. Det kan också hända att ändringarna, om bägge processerna skriver till filen, blandas ihop om vartannat. Naturligtvis blir det ännu mer fel om fler än två processer är inblandade.

Det är inte acceptabelt att det kan bli fel. En process som läser från en fil medan en annan process skriver skall se all data som läses i ett systemanrop antingen helt och hållet som det såg ut innan uppdatering, eller helt och hållet så som det ser ut efter uppdatering. Vid skrivning gäller att allt som skrivs inom ett systemanrop skall lagras på filen i sin helhet. Det skall finnas en (kort) tidsperiod precis efter systemanropet då allt som skrivits syns i filen, även om någon samtidigt försöker skriva över vissa delar.

Det som speciellt kan observeras är att det inte kan bli några fel om två eller flera processer läser från en fil samtidigt (så länge ingen skriver till filen). Det är alltså möjligt att låta läsning ske samtidigt, parallellt. Detta kan öka prestanda genom att undvika att processer stoppar för att vänta på varandra, speciellt i situationer när en fil uppdateras sällan, men läses ofta (sidor på en webbserver t.ex.).

Liknelse för tillåtna fall

Det kan vara lite svårt att se hur skrivning skall fungera så här kommer en liknelse.

Tänk dig ett tomt litet bord som rymmer ungefär 4 papper i storlek A4 (om de läggs kant i kant). Ett dussin personer runt bordet håller var och en i ett papper i storlek A4. Varje papper har olika färg. Deras uppgift är att placera papperet någonstans (var och hur som helst) på bordet vid given signal. Hur ser bordet ut när alla är klara? Jo, den sista personens papper kommer alltid synas i sin helhet. Tar vi bort det papperet kommer den näst sista personens papper att synas i sin helhet. Och så vidare. *Detta är rätt resultat. Den senaste operationen skall alltid synas i sin helhet.*

Tänk dig nu att varje persons papper är delat i 8 delar. De har samma uppgift: hela papperet, alla 8 delar, skall placeras på bordet vid given signal. Hur ser bordet ut när alla är klara? Jo, förmodligen kommer inga av de först placerade pappersdelarna att synas, utan endast de sist placerade delarna av varje papper. Alla papper kommer ha någon del som ligger under de sista delarna från ett annat papper. Detta är fel resultat. Det som syns överst på bordet kommer vara en blandning av pappersdelar från olika personer, inget papper kommer synas i sin helhet. Det är inconsistency.

Rätt resultat får man när de 8 delarna behandlas som om de fortfarande satt ihop i ett papper, d.v.s. ingen får placera någon av sina delar medan någon annan håller på placera sina.

Operationen att lägga hela sitt papper (alla delar) på bordet motsvaras av att anropa systemanropet `write`. Buffern som skickas med `write` är papperet (alla delar) och det som ligger på bordet är innehållet i filen. I ett systemanrop kan “papperet” vara uppdelat i väldigt många delar. Trots det skall det senaste “papperet” alltid synas i sin helhet när filen läses, eller när den skrives. Filen skall alltså vid alla tidpunkter se ut som om *alla* haft *hela* papper när de placerade papperet på bordet.

Liknelse för algoritm

Föreställ dig en offentlig toalett med en toalettstol och en (oändligt lång) pissoar. Rummet har en ingång, men dörrlåset är tyvärr trasigt, så dörren går inte att låsa. Istället har någon satt upp en skylt med ordet “ledigt” på ena sidan, och ordet upptaget på andra sidan. Skylten går att vända. Det finns även en tavla på dörren där man antingen kan dra ett streck, eller sudda ett streck. Något annat kan inte ritas på tavlan.

Nu är det så att herrar endast behöver nyttja pissoaren (bortse från andra behov), och damer endast behöver toalettstolen. Tiden besöket tar varierar slumpmässigt för olika personer. Kommer flera in samtidigt är det alltså inte nödvändigtvis den som är först in som är först (eller sist) ut. En allvarlig begränsning är att ingen kommunikation mellan olika personer kan förekomma, annat än genom skylten och tavlan.

Att åstadkomma enskild tillgång till rummet för var och en är enkelt. Vilket protokoll skall alla följa då? Jo, titta först på skylten. Ledigt? Vänd skylten och stig in. Upptaget? Vänta på att skylten visar ledigt. När du går ut vänder du skylten till ledigt. Detta protokoll är enkelt men skapar lätt en kö om många behöver nyttja faciliteten.

Utgå från att det är möjligt att flera herrar nyttjar rummet samtidigt, medan damer skall få enskild tillgång till rummet. Din uppgift är att uppfinna ett protokoll för att tillåta flera herrar samtidigt.

Det som är intressant då är hur herrar respektive damer skall bete sig när de kommer och när de går. Mest intressant är kanske hur skylten och tavlan hanteras av herrarna, så att skylten alltid visar ledigt när rummet är tomt. Vem vänder fram “upptaget”-sidan? Vem vänder tillbaka “ledigt”-sidan? Om det är upptaget, hur vet ankommande herrar om de skall vänta eller stiga på?

Uppgift

Implementera synkronisering av läsning och skrivning av filer så att inga fel kan uppstå. En bra början är att se till ingen kan läsa eller skriva förrän tidigare läsningar eller skrivningar slutförts helt och hållet. Detta är dessutom enkelt att implementera.

Utöka sedan din lösning så att *flera kan läsa samtidigt*, d.v.s. innan andra som läser är helt klara. Det får fortfarande inte vara möjligt att någon skriver samtidigt som någon annan läser eller skriver. D.v.s senaste skrivna data i filen skall fortfarande synas i sin helhet, aldrig delvis. I din lösning får *starvation* vara möjligt. Du får däremot inte begränsa antalet processer som väntar på sin tur, vill någon process läsa eller skriva skall den alltid få en “köplats” om den inte kan få direkt tillgång till filen. Antalet processer som får läsa samtidigt är obegränsat. Skillnaden från enkel synkronisering är att endast en av de som läser samtidigt behöver markera att filen är upptagen (den markeringen gäller gemensamt för samtliga som läser just då). Frågan är bara vem som skall markera att filen är upptagen, och vem som markerar att den är ledig igen? Och hur skall du hålla reda på det?

Tänk noga på var du placerar dina synkroniseringsvariabler. Använd den kunskap du om filsystemets implementation som du skaffade dig i tidigare uppgift. Du måste se till att en fil som används samtidigt från två olika processer faktiskt också använder samma synkroniseringsvariabler, medan andra filer som används samtidigt använder andra synkroniseringsvariabler.

Testa din implementering

Ett speciellt testprogram finns i examples-katalogen. Det körs på följande sätt:

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

Programmet `pfs_reader` är det som kontrollerar att senast skrivna "papper" alltid syns i sin helhet. Om det är så skrivs `cool` i filen `messages`. Annars skrivs `INCONSISTENCY`.
Tre `pfs_reader`-processer startas, och **varje** `pfs_reader` utför kontrollen **500** gånger (samtidigt som `pfs_writer` skriver till filen). När allt är klart skall du kontrollera hur många `cool` som skrivits till filen:

```
grep -c cool messages
```

Du får själv fundera på vilken siffra som är rätt. 1500 eller 500? Hur kan det komma sig?

Du skall även kontrollera att resultatet inte innehåller något annat än cool:

```
grep -v '^cool$' messages
```

Finns det något annat fungerar inte din synkronisering perfekt. Då kan det även finnas lite för många `cool` i filen. Hur kan det komma sig att det blir för många `cool`?

Tips: Hur håller filen reda på aktuell läs- och skrivposition? I vilken datastruktur lagras det?