LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Övning på synkronisering

Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018

2021-03-14

1 Mål

Målet med denna laboration är att öva på att identifiera och synkronisera delad data inför kommande laborationer (främst labben ”Synkronisering”). Utöver detta liknar problemet hur filsystemet i Pintos är strukturerat, så idéer från den här labben kommer att kunna användas framöver. Denna laboration görs utanför Pintos, vilket gör det lite enklare att se vad som händer (exempelvis med visualisering) och felsöka eventuella problem.

2 Uppgift

I mappen `src/standalone/lab11` finns filen `data-files.c` som implementerar ett enkelt system för att läsa in och använda ”stora” datafiler. Tanken är att det finns två datafiler som programmet behöver komma åt med jämna mellanrum. När någon del av programmet behöver en av filerna så vill vi ladda in hela filen i RAM så att det går snabbt att läsa datan i filen. I och med att filerna är stora vill vi se till att 1) vi inte laddar in varje fil mer än en gång i RAM och 2) vi vill ta bort kopian i RAM så snart vi vet att vi inte behöver den längre.

Det finns två intressanta funktioner i implementationen:

`data_open` Öppnar datafilen med det givna indexet (0 eller 1). Om filen redan var öppen så ska den öppna filen återanvändas.

`data_close` Stänger en tidigare öppen datafil. När alla som använder en fil har stängt den så ska datastrukturen frigöras för att spara minne.

För att hålla koll på hur många gånger en fil har blivit öppnad används en räknare vid namn `open_count`. Den ökas varje gång en fil öppnas, och minskas varje gång en fil stängs. På det viset kan implementationen hålla koll på när det är säkert att ta bort kopian i RAM. Detta liknar hur Pintos hanterar inodes.

Implementationen innehåller för närvarande synkroniseringsfel som gör att implementationen inte fungerar som den ska. Exempelvis kan det bli så att samma fil läses in i RAM mer än en gång. Det kan också bli så att en fil tas bort innan alla som öppnat den är färdiga med att använda den. Din uppgift är att lösa de synkroniseringsproblem som finns.

3 Kompilera och kör koden

Du kan antingen köra programmet i visualiseringsverktyget eller i terminalen. Nedan beskrivs båda sätten.

3.1 Visualiseringsverktyget

Starta visualiseringsverktyget enligt instruktioner på kurshemsidan (välj ”Visualiseringsverktyg” i vänstermenyn). På skolans datorer kan du köra kommandot `/courses/TDIU16/progvis/start` men du kan även hämta verktyget till din egen dator om du vill.

När du har startat verktyget kan du sedan välja *File* → *Open...* och sedan välja källkodsfilen för labben. Koden kompileras då automatiskt, och du kan sedan stega fram de olika trådarna i programmet som du vill. Om du ändrat något i källkoden kan du välja *File* → *Reload* (Ctrl+Shift+R) för att se dina ändringar.

Tips: Testa även *Run* → *Look for errors* för att få en indikation om din lösning innehåller synkroniseringsfel eller inte.

3.2 I terminalen

Gå till mappen där `data-files.c` finns, och skriv `make data-files` för att kompilera koden. Du kan sedan köra programmet med `./data-files`. Du kan också använda GDB som vanligt med kommandot `gdb ./data-files` eller `gdb -tui ./data-files` om det behövs.

4 Testa din implementation

Som ofta är fallet är det svårt att testa ifall kod är korrekt synkroniserad. I allmänhet kräver detta att man noga funderar över vilken data som kan vara delad mellan olika trådar, och vilken kod som måste synkroniseras för att lösa de problem som kan uppkomma. I det här problemet görs detta lämpligtvis med hjälp av lås. Ett bra tillvägagångssätt är då att tänka att varje lås skyddar någon eller några specifika variabler. Sedan kan man titta igenom sitt program och se till så att tråden håller låset när den använder skyddad data.

I just det här fallet kan visualiseringsverktyget användas för att testa din implementation. Det vi vill ska hända är att implementationen är fri från synkroniseringsproblem. Vi vill också att det aldrig ska finnas fler än två instanser av `struct data_file` samtidigt, och att alla instanser frigörs i slutet av programmet. Väljer du *Run* → *Look for errors* får du en bra indikation på om du har gjort rätt eller inte.

Det är en bra idé att diskutera din lösning med assistenten när du känner dig klar. På så vis kan du undvika problem i nästa laboration.