LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Den första processen

Klas Arvidsson, Daniel Thorén, Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018

2020-03-06

1 Mål

Det finns flera olika metoder som sköter processer och trådar i Pintos, det finns även ett antal strukturer ämnade för synkronisering av processer. Funktionerna `process_execute` och `start_process` är ansvariga för att starta en ny process. För mer information om hur en process startar se Pintos-Wiki under rubriken ”Tråd- och processhantering i Pintos”, se särskilt underrubriken ”Starta en process”.

2 Uppgift (90% förarbete, 10% kodning)

Den nuvarande implementationen av `process_execute` och `start_process` är inte kompletta. Mer specifikt fungerar inte synkroniseringen mellan dem. I en korrekt implementation vill vi att `process_execute` inte returnerar förrän vi är säkra på att den nya processen har kommit igång ordentligt. Gör vi inte det så kommer vi dels inte att veta om vi lyckades starta den nya processen eller inte (`process_execute` ska returnera ett så kallat process-id om allt gick bra, eller -1 ifall något gick fel), och dels kanske vi börjar städa upp saker som den nya processen behöver för att kunna starta.

I Pintos-Wiki, under rubriken ”Tråd- och processhantering i Pintos” finns en mer ingåendebeskrivning av vad som händer. Där framgår att punkt 3 (vänta), 4 (se om allt gick bra) och e (skicka resultatet) inte fungerar korrekt i den nuvarande koden (punkt d löses senare).

Implementera den synkronisering som behövs för att lösa dessa tre punkter korrekt och utan att använda ”busy-wait”. När du är klar skall det inte finnas några spår av `power_off` raden. Du skall se till att vänta precis så mycket som behövs (vare sig det är inget alls eller 7 timmar). Tänk på att det måste fungera korrekt för godtyckligt antal processer. Att använda globala synkroniseringsvariabler kommer *inte* att fungera (Varför?).

Det finns debugutskrifter i början på `process_execute`, i början av `start_process`, och i slutet av `process_execute`. De skall *alltid* skrivas ut i samma ordning som i föregående mening när du gjort rätt. Om `start_process` utskriften vid någon körning kommer sist har du gjort fel. Gör du fel är det även mycket troligt att du får minnesfel, då `start_process` kommer använda variabler som blivit ogiltiga sedan `process_execute` returnerat. Att allokera dynamiskt minne med `malloc` i `process_execute`, och sedan frigöra det minnet i `start_process` är **inte** en godkänd lösning på sådana problem (det är alltid en god vana att samma funktion, modul eller tråd som allokerar minne ansvarar för att frigöra detsamma). Lös allt genom att vänta lagom länge.

När du är klar, tänk noga igenom din lösning med ledning av ovan information, och testa sedan genom följande tre testfall:

- Starta ett användarprogram enligt föregående uppgifter, t.ex. `sumargv`. I debug-utskriften för returvärdet från `process_execute` skall du se ett giltigt id, t.ex. 3.
- Gör som ovan, men skriv fel programnamn, skriv t.ex. `sumsum` istället för `sumargv` sist på raden. Detta gör att `load` i `start_process` kommer misslyckas. I debug-utskriften för returvärdet från `process_execute` skall du se -1.
- Kör ett program med argument `-tcl=2`. Detta simulerar att `thread_create` misslyckas skapa tråden vid andra anropet. I debug-utskriften för returvärdet från `process_execute` skall du se -1, eftersom det inte blev någon process (inte ens en tråd!).

```
pintos -p ../examples/sumargv -a sumargv -v -k --fs-disk=2 \
        -- -f -q -tcl=2 run 'sumargv 1 2 3'
```

När allt fungerar, be till slut assistenten kontrollera din lösning.