Kurs: DVA248 – Datorsystem

Version: 1.1, uppdaterad 2021-04-15

Utvecklad för Python av: Dag Nyström, dag.nystrom@mdh.se

Lab 1: Trådar och synkronisering

Introduktion

Syftet med den här laborationen är att du ska förstå hur pseudoparallellism kan uppnås med multitrådade processer samt hur trådar kan synkroniseras.

Genomförande och redovisning

Laborationen utförs gemensamt av labgrupperna.

Den färdiga koden skickas in via CANVAS efter redovisning för labassistent.

NOTERA: Räkna INTE med att ni kommer att hinna bli klara under labbtillfällena, de flesta behöver lägga tid utanför schemalagd tid.

Några ord om Python innan vi sätter igång

Syftet med den här laborationen är egentligen att lära sig att använda operativsystemsstöd för

- Skapande av trådar
- Synkronisering av trådar

I de flesta programmeringsspråk så brukar man direkt använda sig av operativsystemets tjänster genom att man importerar en operativsystemsspecifik modul till sitt program och får då tillgång till just det aktuella operativsystemets tjänster.

Python är egentligen ett scriptspråk, alternativt ett högnivåspråk, som är tänkt att vara operativsystemsoberoende. För Python är dessutom innebörden av högnivåspråk en produkt med ”extra allt” (eng. all bells and whistles), vilket naturligtvis gör det bekvämt och effektivt att koda i men tyvärr abstraherar sig rätt långt ifrån de tjänster som tillhandahålls av operativsystemet (så som det presenteras på kursens föreläsningar).

Detta får följande effekt på den här laborationen:

1. Trådar i Python beter sig som standard lite annorlunda än trådar i ett operativsystem. För att få samma beteende som i ett operativsystem kommer vi att använda oss av så kallade ”deamon threads” i denna laboration.

Min förhoppning är dock att ni, ”trots Python” ska få samma kunskap som om ni kodat detta direkt mot ett operativsystem.

Nu kör vi!!

Uppgift 1 – Lets stay single for a while….

a)

Skapa ett program som består av:

- En mainfunktion som skriver ut ”Hello World” på skärmen 10 gånger

b)

Nu skall vi lugna ner programmet lite så det inte går så väldigt snabbt.

Hur gör man detta??

Det enklaste borde ju vara att skriva en loop som snurrar en stund utan att göra något….

Men det känns som väldigt mycket CPU-arbete utan att något nyttigt görs. Tänk om vi kunde frysa processen istället, dvs sätta programmet i tillståndet ”Waiting” en viss tid innan vi släpper tillbaks den till ”Ready”…..

I modulen `time` finns funktionen `sleep()` som gör just detta. Den fryser en tråd i det antal sekunder som parametern säger (notera att man kan kan ange t.ex. 0.5 för en halv sekund)

Modifiera programmet så att det blir en sekunds paus mellan varje utskrift.

c)

Lägg till en funktion som heter `myMoon()` som skriver ut ”Hello Moon!” på skärmen var femtedels sekund i en oändlig loop.

Först i mainfunktionen lägger ni nu till ett anrop till `myMoon()`

Innan ni kör programmet, fundera på vad ni förväntar er för utdata.

Kör programmet!

Fick ni det förväntade beteendet? Om inte, vad hände igentligen?

Varför skrevs inte ”Hello World” ut på skärmen?

Spara koden från Uppgift 1 innan ni fortsätter med Uppgift 2 för redovisningen.

Uppgift 2 – Single is boring, lets go multi instead..

Man kan ju tänka sig att programmerarens önskan i Uppgift 1 skulle vara att både ”Hello World!” och ”Hello Moon!” skulle skrivas ut…. Eftersom dessa två utskrifter har olika periodtid så blir det ganska krångligt att skriva programmet som en loop som sköter båda utskrifterna. Ett bättre sätt är ju då att skapa en multitrådad applikation.

I Python implementeras trådar i modulen *threading* och objektet vi skall använda heter kort och gott *Thread*. Läs på lite kring objektet Thread (finns massor på nätet).

Titta lite extra på parametrarna *target* och *deamon*, samt metoden *start*.

Tips: När ni tittar på parametrarna på konstruktorn så kan det se lite komplext och mastigt ut. I denna uppgift behöver ni bara använda *target* och *deamon*. Det finns ett klurigt sätt att hoppa över parametrar i Python genom att istället för att lista alla parametrar, namnge de man vill ha med. Om du t.ex. har funktionen add(tal1, tal2) så kan du anropa den med add(2,3) ELLER add(tal1=2, tal2=3).

a)

Ändra i er main-funktion så att `myMoon()` startas i en egen tråd istället för det funktionsanrop till den som ni implementerade i Uppgift 1. För att emulera operativsystemets trådar så måste ni sätta `daemon=True`.

Innan ni kör programmet, fundera på vad ni förväntar er för utdata.

OBSERVERA: Vissa miljöer, t.ex. Spyder kommer att ge fel beteende för just denna dellaboration. För att se till att programmet beter sig korrekt så bör ni starta denna i terminaläge. Skapa ett terminalfönster, ta er till rätt katalog och kör programmet med ”python *filnamn*.py” alternativt ”python3 *filnamn*.py”.

Kör programmet i minst 10 sekunder!

Fick ni det förväntade beteendet? Om inte, vad hände igentligen?

Varför terminerade plötsligt processen?
(Om processen inte terminerade, se observationen ovan)

b)

Så i princip har vi bara ”anropat funktionen” lite annorlunda så att den börjar exekvera parallellt med den anropande main-tråden. Men vi saknar fortfarande möjligheten att skicka med parametrar till tråden.

Modifiera `myMoon()` så att den tar en sträng som inparameter och istället för ”Hello Moon!” skriver ut den strängen.

Modifiera main funktionen så att den först låter användaren mata in en sträng.

Utöka sedan trådanropet så att strängen skickas från main till tråden när den skapas.

Spara koden från Uppgift 2 innan ni fortsätter med Uppgift 3 för redovisningen.

Uppgift 3: You got this sync'in feeling….

Dags att synkronisera de två trådarna.

a)

Modifiera programmet så att ”Hello World!” skrivs ut 10 gånger (från main-tråden) och `myMoon()` skriver sin text 10 gånger. Dessa utskrifter ska sedan fortsätta att skrivas ut i klumpar om 10 rader som inte får blandas. Det får alltså inte finnas ställen där t.ex. 5 hello world skrivs ut och sedan blir det 10 moon-texter.

Innan ni kör programmet, fundera på vad ni förväntar er för utdata.

Kör programmet i minst 10 sekunder!

Scrolla tillbaks i utskrifterna.

Fick ni det förväntade beteendet? Om inte, vad hände igentligen?

Om ni fick det, kan ni verkligen garantera att det alltid blir klumpar om 10??

b)

Gissningsvis så använde ni antingen korta sleep anrop eller någon delad variabel för att uppnå synkroniseringen.

Eller så har ni redan gått igenom föreläsningen kring processkommunikation och använt någon primitiv för att synkronisera trådar (typ Lock). I så fall är ni faktiskt klara nu.

Har ni använt sleep så kan ni faktiskt inte garantera att det fungerar. Skulle någon annan tung process stjäla CPUn ett tag så vet ni inte i vilken ordning trådarna kommer att köras när båda är i readykön.

Har ni använt en delad variabel (typ en int eller liknande) så är risken stor att den är felimplementerad, alternativt att koden nu innehåller loopar som väntar på trådens tur, så kallad *busy waiting*.

Precis som att ni använde sleep för att frysa trådar så kan man också använda `Lock` som också finns i modulen `threading` för att frysa trådar som väntar på någon annan tråd. Lock kan låsas och låsas upp och det är bara en tråd i taget som kan låsa. Försöker man låsa ett redan låst lås så fryser tråden tills låset blir upplåst igen.

Modifiera programmet så att det använder `Lock` istället!!

Bra jobbat. Dags att redovisa labben!!!
Glöm inte att skicka in koden när ni är godkända samt klaga hos labassen för de urusla ordvitsarna i labspecen!!!