Del 1: Synkronisering

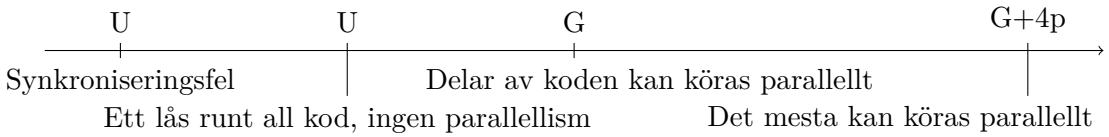
Bedömning

För att få godkänt på tentan krävs att den här uppgiften är godkänd. Uppgiften kan dessutom ge totalt 4 bonuspoäng.

För att uppgiften ska bedömas som godkänd ska koden fungera enligt beskrivningen nedan och enligt beskriven i kommentarerna i den givna koden. Busy-wait ska undvikas, och olägliga trådbyten ska inte kunna orsaka fel givet att koden används enligt specifikationen, oberoende av hur osannolikt detta kan vara. Om inte annat anges ska alla funktioner kunna anropas från flera trådar samtidigt utan att det leder till problem, oavsett om detta görs i exempelprogrammet eller inte. Om du inser att du har glömt något kan du fortfarande få G på uppgiften genom att i del 2 beskriva varför det är fel och beskriva hur du skulle lösa problemet.

För upp till 4 bonuspoäng ska det vara möjligt för kod som arbetar på orelaterad data att köras parallellt i så stor mån som möjligt. Detta innebär att kritiska sektioner ska vara så korta som möjligt, och att lås i den mån det är möjligt är placerade tillsammans med den data de skyddar. Alltså ska exempelvis kod som hanterar orelaterade datastrukturer kunna köras parallellt, och i den mån det är möjligt ska även operationer på samma datastruktur kunna köras parallellt. Detta gäller även om exempelprogrammet inte instansierar flera datastrukturer, eller kör funktioner parallellt från flera trådar.

Bedömningen kan sammanfattas i diagrammet nedan:



Notera: all kod under kommentaren *Huvudprogram* är inte en del av uppgiften och kommer inte att rättas, så lägg ingen synkronisering där. Huvudprogrammet finns för att programmet ska gå att köra, och för att visa hur resterande kod kan användas. Det är *inte* byggt för att illustrera alla potentiella problem i den givna koden. Du får gärna modifiera huvudprogrammet för att testa din lösning om du vill, det är dock inget krav.

Notera: I denna uppgift ska du använda lås, semaforer, och/eller condition variables. Atomiska operationer används i en senare uppgift.

Inlämning

Modifiera koden i filen `exams.c` och lämna in din lösning i Lisam senast klockan 15:30.

Ange följande text i textfältet där du lämnar in uppgiften för att visa att du har läst och förstått reglerna i regeldokumentet:

Jag har läst och förstått tentans regler, och jag lovar att jag har följt dem under tentan.

Uppgift

Din kompis, Kim, som arbetar som matematiklärare har under våren funderat mycket på hur examinationen bäst kan anpassas för att kunna ske på distans, och kom fram till att det är bäst att ge varje student unika uppgifter. Kim insåg däremot att det skulle bli riktigt tungt att skapa individuella uppgifter för alla studenter och att sedan hålla reda på vem som fick vilken uppgift för att sedan kunna rätta uppgifterna på ett bra sätt.

Därför har nu Kim bett dig att skriva ett program för att hantera alla uppgifterna. Kim har redan skrivit datatypen `task`, som representerar en uppgift. Den har följande funktioner:

`task_create` Skapar en uppgift. I och med att Kim vill att uppgifterna ska hålla hög kvalité och vara likvärdiga tar denna funktionen lång tid på sig att komma fram till en bra uppgift.

`task_destroy` Frigör det minne som allokerades av `task_create`.

`task_answer` Beräknar svaret till en uppgift.

`task_print` Skriver ut uppgiften till standard output.

Givet Kims kod för att hantera uppgifter har du börjat implementera resten av programmet i filen `exams.c`. Programmet ska i slutändan producera en lista med uppgifter, en till varje student, och en lista med svar till varje uppgift. Du har börjat implementera typen `students` för att hantera detta. I och med att det tar tid att generera uppgifter vill du utnyttja alla tillgängliga kärnor för att bli klar snabbare (det viktiga är alltså att kunna köra `task_create` parallellt). Du har därmed beslutat att tråda skapandet av uppgifter. För att ge Kim en uppfattning om hur lång tid det är kvar ska programmet också skriva ut en statustext som säger hur många uppgifter som har genererats. Datatypen har följande operationer:

`students_create` Skapar en instans av `students`-typen givet en lista av alla studenters namn.

`students_destroy` Frigör det minne som allokerades av `students_create`.

`students_generate` Skapa uppgifter till alla studenter genom att starta flera trådar och låta dem arbeta parallellt. Du kan anta att denna funktionen bara anropas en gång per `students`-instans.

`students_worker` Hjälpfunktion som körs från flera trådar samtidigt, och är ansvarig för att skapa uppgifterna. Vi antar att den bara körs från `student_print_tasks`.

`students_print_tasks` Skriver ut alla genererade uppgifter i en fin tabell. Du kan anta att denna funktion bara anropas från en tråd.

`students_print_answers` Skriver ut alla svar i en fin tabell. Du kan anta att denna funktion bara anropas från en tråd.

När du testkör programmet märker du dock att programmet kraschar för det mesta. På utskriften ser det ut som att uppgifterna genereras som de ska, men att utskriften av uppgifterna får programmet att krascha. Givet din tidigare erfarenhet med parallellprogrammering misstänker du att det är något synkroniseringsfel när uppgifterna skapas, trots att det ut att gå bra. För maximal poäng, se till att din lösning tillåter så många uppgifter som möjligt att genereras samtidigt.

Du kan kompilera koden med kommandot `make exams` i mappen där du har extraherat tentafilerna. Kompileringen ska fungera på det Linux-system du har använt för att göra labbserien. Din kod behöver *inte* kompilera för att ge poäng, så länge vi förstår vad du menar.