

---

## Del 1: Synkronisering

---

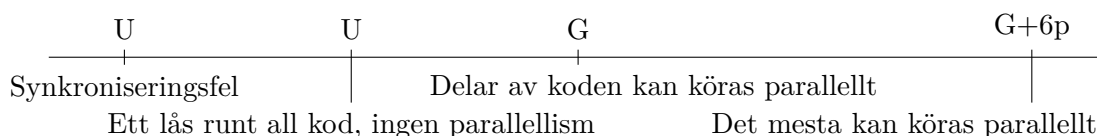
### Bedömning

För att få godkänt på tentan krävs att den här uppgiften är godkänd. Uppgiften kan dessutom ge 6 poäng mot högre betyg.

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.

För 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. För full poäng ska så mycket som möjligt kunna köras parallellt.

Bedömningen kan sammanfattas i diagrammet nedan:



Notera att all kod under kommentaren *Huvudprogram* inte är en del av uppgiften. Allt nedanför den kommer därmed inte 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. Detta är dock, som sagt, inte en del av uppgiften.

### Inlämning

Modifiera koden i filen `parallelize.c` och lämna in din lösning i Lisam senast klockan 9:45.

## Uppgift

Du håller på att implementera ett program som ritar grafer som du hoppas att du kan få ha som hjälpmedel på en kommande mattetenta. I och med att tentatiden är begränsad är varje sekund av tentatiden dyrbar, så du har verkligen inte tid att vänta på ditt program i onödan! Efter att ha funderat lite inser du att det som tar mest tid är att beräkna funktionen du vill rita ut för alla värden på  $x$ -axeln, och du har därmed beslutat dig för att parallellisera beräkningen på dina 4 CPU-kärnor.

Din lösning så här långt finns i filen `parallelize.c`. Du har refaktorert ditt program så att alla tunga beräkningar sker i funktionen `run_in_serial`. Den funktionen får en funktion och en array av  $x$ -värden som parametrar, beräknar  $f(x)$  för alla värden som finns i `input`-arrayen och lägger resultaten `output`-arrayen. För att göra alla beräkningar parallellt har du också implementerat `run_in_parallel`, som ska göra samma sak med skillnaden att den fördelar arbetet på flera trådar.

Tyvärr märker du att `run_in_parallel` inte fungerar som den ska. Ibland verkar det inte som att alla värden beräknas, och ibland kraschar till och med programmet helt och hållet. Efter ett tag märker du också att implementationen ibland inte beräknar alla  $x$ -värden som finns med i `input`-arrayen.

Använd lämpliga synkroniseringsprimitiver (dvs. lås, semaforer eller condition variables) för att lösa synkroniseringsproblemen som finns i koden. Atomiska operationer används inte här, det kommer i del 2.

**Notera:** Funktionen `run_in_parallel` ska kunna anropas från flera trådar samtidigt.

Du kan kompilera koden med kommandot `make parallelize`. Se till att du har kopierat `Makefile` om det inte fungerar.