Del 2: Teori

Uppgifter

1. **Notera:** För G på tentan behöver du få godkänt på denna uppgift. Ett välskrivet resonemang som visar god insikt kan få upp till två bonuspoäng (utöver de 4 från del 1). [2p]

   Tillsammans med det här dokumentet har du fått två lösningar på problemet i del 1 i filerna `a.c` och `b.c`. Dessa lösningar har olika styrkor och svagheter, och har i vissa fall inte ens lyckats synkronisera programmet korrekt.

   Välj en av de ovanstående lösningarna och jämför den med din lösning av del 1. Innan du börjar, fundera på vilken lösning som är mest lämplig att jämföra med. För att kunna svara bra på den här frågan vill du välja en lösning som är tillräckligt olik din lösning, annars kommer du inte kunna svara tillfredsställande på frågorna nedan. Om du inser att din lösning på del 1 innehåller fel, välj då en lösning som löser så många av felen i din lösning som möjligt och beskriv varför din lösning är fel och den givna lösningen är korrekt och vice versa. Gör du detta får du G på del 1 även om den inte var helt korrekt tidigare.

   Jämför nedanstående egenskaper i din lösning och i lösningen du har valt. Skriv sedan ner vad du kom fram till och hur du resonerade för att komma fram till ditt svar, gärna uppdelat i stycken eller rubriker som motsvarar punkterna nedan.

   (a) Innehåller antingen din lösning från del 1 eller den lösningen du har valt några synkroniseringsproblem? Beskriv (gärna med exempel) vad som kan gå fel i den ena, och varför det inte kan hända i den andra.

   (b) Välj en *busy-wait* som fanns i originalimplementationen och beskriv hur antingen din lösning eller den lösning du har valt har gjort för att undvika *busy-wait* problemet.

   (c) I originalimplementationen av `pool_destroy` väntade inte implementationen på att alla trådar hade hunnit stänga av sig efter att den hade skickat stängen `EMPTY` till dem. Beskriv varför det kan vara ett problem, och hur din implementation eller den implementation du har valt har löst problemet.

   (d) Vilken av din lösning eller den lösning du har valt tror du har bäst prestanda, både teoretisk och i praktiken? Motivera ditt svar.

   För godkänt på den här uppgiften vill vi se:

   - Att ditt svar är mellan 700 och 1500 ord (ca 1–2 sidor), exklusive eventuella kodexempel. (Du ska visa att du har förstått kursinnehållet, längden är en riktlinje för att du ska hinna göra det, men inte en hård gräns. Korta och koncisa lösningar är ofta bättre än långa utläggningar.)
   - Att du har diskuterat alla punkter som finns i uppgiften ovan.
   - Att du kan förklara varför din lösning är korrekt, eller med hjälp av en referenslösning kan förklara varför din lösning är felaktig och hur den kan korrigeras.
   - Att du kan förklara skillnaden mellan din lösning och den valda givna lösningen med avseende på korrekthet och möjlig parallellism.

2. I ett system körs tre processer, *P1, P2* och *P3*. I systemet finns det också tre typer av resurser, *A, B* och *C*. Tabell 1 visar hur många resurser som finns, hur systemets resurser är allokerade i nuläget och det maximala resursbehovet för varje process.

	A	B	C
P1	9	6	2
P2	6	1	9
P3	9	9	4

Maximal resursanvändning

	A	B	C
P1	6	4	1
P2	6	0	8
P3	6	5	3

Nuvarande resursanvändning

	A	B	C
P1	19	11	14

Totalt antal resurser

Tabell 1: Resurser i systemet.

(a) Är systemet i ett säkert läge? Redovisa dina beräkningar. [2p]

(b) Process *P3* begär en extra resurs av typ *B*. Ska begäran tillåtas enligt Banker's algorithm? Redovisa dina beräkningar. [1p]

(c) Vad är tabellen med nuvarande resursanvändning efter föregående uppgift? [1p]

3. Nedan finns en dålig implementation av en semafor (som lämpligt nog heter `bad_sema`). Kopiera koden nedan till det dokument du har svarat på övriga frågor i och använd atomiska operationer för att göra implementationen trådsäker (har du problem att kopiera från PDF:en finns koden i `atomics.c` också). Precis som i del 1 behöver den kod du skriver inte kompilera.

```
struct bad_sema {
    int current_value;
};

void bad_sema_init(struct bad_sema *sema, int value) {
    sema->current_value = value;
}

void bad_sema_up(struct bad_sema *sema) {
    sema->current_value++;
}

void bad_sema_down(struct bad_sema *sema) {
    while (sema->current_value <= 0)
        ;
    sema->current_value--;
}
```

(a) Synkronisera den dåliga semaforimplementationen med hjälp av de atomiska operationer som finns i filen `wrap/atomics.h` (det är samma operationer som vi har pratat om i kursen). [3p]

(b) Varför är det inte möjligt att eliminera busy-wait endast med hjälp av atomiska operationer? [1p]