TDIU16 Exam

Klas Arvidsson

2015-06-01 14-18
TER3, TER4, G33

Tillåtna hjälpmedel

Inga hjälpmedel.

Jour

Klas Arvidsson (013-282146) besöker tentamenssalen efter ungefär en timme.

Instruktioner

- Fyll i tentamensomslaget och läs dess instruktioner innan du börjar.
  Läs instruktionerna och alla uppgifter innan du börjar.
- Ange din tolkning av frågan och alla antaganden du gör.
- Skriv tydligt. Oläsliga svar bedöms som noll poäng.
- Var precis i dina påståenden. Bevisa ditt resonemang när möjligt.
  Svävande eller tvetydiga formuleringar leder till poängavdrag.
- Motivera tydligt och djupgående alla påståenden och resonemang.
  Förklara uträkningar och lösningsmetoder.
- Tentamen har 5 uppgifter på 4 sidor (inklusive denna sida).
- Tentamen är på 30 poäng och betygsätts U, 3, 4, 5 (prel. gränser: 16p, 21p, 26p).
  Poäng ges för motiveringar, förklaringar och resonemang. Enbart rätt svar ger inte (full) poäng.
- Lycka till!

Uppgift 1 (5p)

Det finns fyra villkor som måste vara uppfyllda för att deadlock ska kunna uppstå.

(a) Visa och förklara villkoren “hold and wait” samt “circular wait” med en korrekt resursallokeringsgraf. (3p)

(b) Namnge och förklara de övriga båda villkoren. (2p)

Uppgift 2 (4p)

I tabell 1 visas ett systems totala antal av fyra resurser. Tabell 2 visar fyra processers maximala behov för varje resurs. Tabell 3 visar slutligen varje process nuvarande användning av varje resurs. Systemet är i ett säkert läge.
Process $P4$ begär nu ytterligare en resurs D. Beräkna med hjälp av Banker's algoritm om begäran ska tillåtas. (4p)

A	B	C	D
39	37	31	29

Tabell 1: Totalt antal resurser

	A	B	C	D
P1	12	14	14	28
P2	2	9	2	9
P3	10	14	5	12
P4	39	37	31	3

Tabell 2: Maximalt resursbehov

	A	B	C	D
P1	6	3	4	13
P2	1	5	1	5
P3	2	2	1	3
P4	18	20	13	1

Tabell 3: Nuvarande resursanvändning

Uppgift 3 (2p)

Antag att varken allokeringsgraf eller Banker's algoritm är gångbara alternativ för att undvika deadlock. Beskriv en annan metod att organisera lås och låsning som ändå förhindrar deadlock.

Uppgift 4 (4p)

Koden nedan är tänkt att leta upp en ledig plats (0) och markera den som upptagen (1). Synkronisera funktionen korrekt genom att använda *___atomic_swap*. Vi antar att vår kompilator automatiskt konverterar funktionen till korrekt hårdvaruinstruktion. [1]

```
1  int i = 0;
2  while ( seat[i] == 1 ) // skip any occupied seats
3    i = i + 1;
4  seat[i] = 1;           // mark found free seat as occupied
```

Kodexempel att synkronisera med *___atomic_swap* i uppgift 4.

```
1  void ___atomic_swap(int* a, int* b) // compiled to one atomic instruction
2  {
3    int save = *a;
4    *a = *b;
5    *b = save;
6  }
```

Ekvivalent C-kod för instruktionen *___atomic_swap* för uppgift 4.

Uppgift 5 (15p)

I kodexemplet på nästa sida finns en parallell algoritm för att hitta den första positionen som har högsta värdet i ett fält. N trådar söker i en S stor bit vardera.
Därefter väntar varje tråd på en signal om att nästa tråd är klar med sin sökning, sparar sin funna maxposition om den var större än vad som hittats hittills, och skickar slutligen en signal om att tråden är klar.
Sökningen sker alltså parallellt, medan sökresultaten sparas sekvensiellt med start på sista tråden, sedan näst sista osv. Huvudprogrammet skickar direkt signal till sista tråden att sätta igång, och väntar därefter på att första tråden ska bli klar med slutresultatet.

(a) Förklara vad busy-wait innebär och ange det exempel i den givna koden som kan förväntas vänta längst. (2p)

(b) Använd semaforer för att lösa alla problem med busy-wait och låta trådarna slutföra i samma ordning som givet. (3p)

(c) Använd conditions för att lösa alla problem med busy-wait och låta trådarna slutföra i samma ordning som givet. Du kan denna gång anta att din villkorsvariabel enbart ändras en gång av en tråd. (3p)

(d) Modifiera givna koden så att tråden som först är klar med sökningen lägger in sitt max först (se kommentarer för kod som utgår). Förklara steg för steg med hjälp av radnummer hur ett olägligt trådbyte kan göra att slutresultatet ger en position som inte innehåller maxvärdet. (3p)

(e) Baserat på deluppgift d), ange alla kritiska sektioner och använd lås för att skydda dem. (2p)

(f) Baserat på deluppgift d), ange varför utskriften i de flesta fall anger att max finns på position 0 i arrayen och föreslå en lösning. (2p)

[1] Intresserad? https://gcc.gnu.org/onlinedocs/gcc-5.1.0/gcc/_005f_005fatomic-Builtins.html

```
1
2  void find_max(int* start, int* end, int*& global_max,
3                int& my_turn, int& done)
4  {
5    int* max = start;
6    for (int* i = start + 1; i != end; ++i)
7    {
8      if ( *max < *i )
9        max = i;
10   }
11
12   while ( ! my_turn )                          // remove in d), e), f)
13     ;                                          // remove in d), e), f)
14
15   if (*max >= *global_max)
16     global_max = max;
17
18   done = true;                                 // remove in d), e), f)
19 }
20
21 int main()
22 {
23   int array[N*S] = { /* random values */ };
24
25   int* g_max = &array[0];
26   int turn[ N+1 ] = { false... };
27   for ( int i = 0; i < N; ++i )
28   {
29     thread_create(find_max, // function to execute
30                   // and arguments to pass in
31                   &array[i*S], &array[(i+1)*S],
32                   g_max, turn[i+1], turn[i]);
33   }
34
35   turn[ N ] = true;                             // remove in d), e), f)
36   while ( ! turn[0] )                           // remove in d), e), f)
37     ;                                           // remove in d), e), f)
38
39   cout << "Max value was " << *g_max
40        << " found at index " << (g_max - array) << endl;
41 }
```

C++ kodexempel för uppgift 5.

```
1  struct semaphore;
2  void sema_init(struct semaphore*, int);
3  void sema_down(struct semaphore*);
4  void sema_up(struct semaphore*);
5
6  struct lock;
7  void lock_init(struct lock*);
8  void lock_acquire(struct lock*);
9  void lock_release(struct lock*);
10
11 struct condition;
12 void cond_init(struct condition*);
13 void cond_wait(struct condition*, struct lock*);
14 void cond_signal(struct condition*, struct lock*);
```

Tillgängliga synkroniseringsmekanismer för uppgift 5 enligt Pintos implementation.