TDIU16 Exam

Klas Arvidsson

2015-06-02 14-18
TER1, TERD

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 7 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 (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 $P3$ begär nu ytterligare en resurs C. Beräkna med hjälp av Banker's algoritm om begäran ska tillåtas.

A	B	C	D
37	39	29	31

Tabell 1: Totalt antal resurser

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

Tabell 2: Maximalt resursbehov

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

Tabell 3: Nuvarande resursanvändning

Uppgift 2 (6p)

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

(a) Två av villkoren kan illustreras med en resursallokeringsgraf. Välj ett av de två och exemplifiera det med en korrekt graf över vilka processer som håller och väntar på varje resurs. (2p)

(b) Namnge och förklara vart och ett av de fyra villkoren. (4p)

Uppgift 3 (2p)

I ett stort system som ofta behöver hålla flera lås samtidigt har man numrerat alla lås och skapat en låsfunktion som låser inte bara ett lås, utan alla lås i en lista:

```
1  lock_all(vector<Lock*> const& v)
2  {
3    for (Lock* l : v)
4      lock_acquire(l);
5  }
```

Samantha föreslår att låsvektorn bör sorteras enligt låsens nummerordning innan låsen tas. Kim protesterar att det kostar prestanda och att ordningen inte spelar någon roll, alla lås blir ju ändå låsta. Förklara varför Samantha har rätt.

Bakgrund till uppgift 4,5,6

Ett skrivarsystem använder en skrivarkö enligt koden på sista sidan. Det finns ett antal skrivare som automatiskt plockar jobb från kön (*get_job_to_print*), och ännu fler användare som lägger till jobb i kön (*add_print_job*).

Uppgift 4 (3p)

Visa ett konkret exempel på hur ett olägligt trådbyte kan göra att utskriftsjobb försvinner spårlöst.

Uppgift 5 (5p)

Förklara begreppet busy-wait och visa hur du löser alla eventuella förekomster i koden.

Uppgift 6 (6p)

Utgå från din kod som den ser ut efter uppgift 5 om du löst den.

(a) Förklara vilka kritiska sektioner som finns i koden och visa hur du synkroniserar dessa korrekt med ett lås. (4p)

(b) Förbättra din lösning i (a) så funktionerna blockerar varandra så lite som möjligt. (2p)

Uppgift 7 (4p)

Visa hur reservationskoden nedan kan synkroniseras korrekt genom att använda funktionen *__atomic_swap*. Vi antar att vår kompilator automatiskt konverterar funktionen till korrekt hårdvaruinstruktion. [1]

```
1  int i = 0;
2  while ( reservation[i] == 1 )
3    i = i + 1;
4  reservation[i] = 1;
```

Kodexempel att synkronisera med *__atomic_swap* i uppgift 7.

```
1  __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 7.

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

```cpp
template<int SIZE>
class Print_Queue
{
public:
  Print_Queue() : work_buffer() {}

  void add_print_job(Print_Job* task);
  Print_Job* get_job_to_print();

private:
  std::array<Print_Job*, SIZE> work_buffer;
  int free_queue_slots = SIZE;
  int read_pos = 0;
  int write_pos = 0;
};

template<int SIZE>
void Print_Queue<SIZE>::add_print_job(Print_Job* task)
{
  while ( free_queue_slots == 0 )
    ; // Wait for a free slot

  work_buffer[write_pos] = task;
  write_pos = (write_pos + 1) % SIZE;

  free_queue_slots = free_queue_slots - 1;
}

template<int SIZE>
Print_Job* Print_Queue<SIZE>::get_job_to_print()
{
  while ( free_queue_slots == SIZE )
    ; // Wait for a print job

  Print_Job* job = work_buffer[read_pos];
  work_buffer[read_pos] = nullptr;
  read_pos = (read_pos + 1) % SIZE;

  free_queue_slots = free_queue_slots + 1;

  return job;
}
```

C++ kodexempel för uppgift 4, 5 och 6. SIZE är konstant.

```cpp
struct semaphore;
void sema_init(struct semaphore*, int);
void sema_down(struct semaphore*);
void sema_up(struct semaphore*);

struct lock;
void lock_init(struct lock*);
void lock_acquire(struct lock*);
void lock_release(struct lock*);

struct condition;
void cond_init(struct condition*);
void cond_wait(struct condition*, struct lock*);
void cond_signal(struct condition*, struct lock*);
```

Tillgängliga synkroniseringsmekanismer för uppgift 4-6 enligt Pintos implementation.