병행 프로그래밍에서 상속 이상의 구현에 관한 연구

論 文
54P-1-9

A Study on the Implementation of Inheritance Anomaly in Concurrent Programming

曺 明 絃[†] · 李 明 彦[*]
(Myung-Hyun Cho · Myung-Un Lee)

Abstract – When concurrency is integrated to programming language, problem more than inheritance remains yet. Because more than inheritance happens by collision or cross fire between synchronization and inheritance of same time object,when synchronization code is not detached properly from method code about same time object, it makes expanded of code to make derivation class change synchronization code of super class and method code.
In this paper, minimize right of inheritance method, When subclass is introduced to new synchronization limitation condition, wish to solve problem more than inheritance of three types that happen in same time programming embodying C++ class that do so that can avoid alteration of method reed.

Key Words : synchronization code, first in first out, b-buf, lb-buf, BoundedBuf

1. Introduction

Inheritance inherits attribute or composition element of class that one class is different.

Relation between class can appear by class structure, and subclass can inherit attribute of super class, or add and define new attribute.

In case composition element name of super class is same with composition element name of subclass, collision can arise, andchange high position element name in other name this time, and can redefine in super class in other form in subclass about method.But, more than inheritance happens by conflict between concurrency and concurrency when wish to keep synchronization limitation condition of same time object in same time programming language including concurrency. So that method that is defined subclass and super class executes different work, to subclass finance of when define subclass because can not grasp easily method action of subclass by super class in class high a hierarchy structure if become in addition of new method limitation for conditionality.

Synchronization happen, and redefine because can not use method that is defined in super class just as it is in subclass must.

Maintenance uses easy C++ language in this paper, wish to solve problem more than inheritance by embodying class that do to offer advantage that can do to minimize re-right of inherited method.

† 교신저자, 正會員 : 瑞逸大學 電氣工學科 副敎授・工博
E-mail : cmhyun01@mail.seoil.ac.kr
* 正 會 員 : 瑞逸大學 電氣工學科 兼任敎授・工博
接受日字 : 2004年 11月 12日
最終完了 : 2005年 2月 19日

2. Theoretical ancient temple more than inheritance

2.1 Justice

When more than inheritance wishes to keep synchronization limitation condition of same time object, is phenomenon that happen by conflict between concurrency and concurrency, when method of super class is not inherited entirely, synchronization code of super class and method code happen when should be changed by extension of synchronization code that synchronization code for same time object creates derivation class in case was not segmentalized properly in method code.

Do to redefine method that method that is added to subclass by occurrence more than inheritance is inherited fromsuper class, and method that when inherit code of same time object by re-justice, capsulation of class can break, and is inherited can not reuse.

2.2 Type more than inheritance

Object is described to state set that presumption is available, and set of state is segmentalized to subset called accept-set according to synchronization limitation condition of object. When new method is added to subclass, necessity that accept-set divides accept-set of super class because synchronization limitation condition of new method can not express definitely state of super class occurs.

The following is first in first out(FIFO) border buffer that is embodied by action abstract picture way.

2.2.1 Division something wrong of permission state

Fig. 4.4 of [1] is x-buf2 class that more than division of permission state happens[1].

X-buf2 that is class b-buf's subclass get2 () that is get () put () instance variable in and out, and new method that is inherited from b-buf class compose.

Method get () is similar except that get2 () action removes oldest element two at the same time within buffer of x-buf2 class.If add get2 () that remove two element in X-buf2 class, because element more than only one must distinguish state of whether exist to buffer buf or do not exist, accept-set partial should be segmentalized in x-one and x-partial state. Therefore, get () put () because should be redefined and what of method except initialization is not inherited bysubclass x-buf2 in b-buf class more than division of permission state happen.

2.2.2 Past susceptibility something wrong of permission state

Fig. 4.8 of [1] is inherited from b-buf and it is method gget () added subclassgb-buf[1].

Get () is equal almost by method that gget () action has one exception that can not be permited immediately since put () call. Because exception condition for these method call can not discriminate set of instance variable method guard and class b-buf, extra instance variable 'After-put' To need, and establish or re-establish value of new variable because can not define in variable that is inherited from class b-buf 'After-put' Add, and get () put () everybody finance of do must .

More than past susceptibility of permission state happens thereby.

2.2.3 Alteration something wrong of permission state

Fig. 4.9 of [1] is lock and lb-buf class[1].

Lock's method practice changes state set that method that is inherited from parents can be called.

Lock class is abstraction mixing class, and direct instances are not created in lock class. But, should be mixed in class that purpose of lock class is different to add fastening ability of object. When Lock is mixed with b-buf for lb-buf's creation, because may not influence in justiceof method that state of object is different about method lock and unlock object present 'Locked' Is it state 'Unlocked' Instance variable that display whether is state 'Locked' Add must .

It is because distinction of lock and unlock that is two states of object is impossible, put () get () inherited method 'Locked'

For examination of state finance of because must become more than alteration of permission state happen.

3. Design of C++ class

These chapter design C++ class to solve problem more than inheritance of three types.

Designed C++ class is consisted of guard, body, and transition part.

Body part is code that achieve method and approaches directly in interior state of object, guard part is examined before execute body part by Boolean declarer who depend on instance variable value and method benevolent person aboutstate of object.

Transition part is set that is made by Boolean expression and list of two interface methods, and transition declarer is evaluated after method body's practice.

3.1 BoundedBuf class

Fig. 1 is justice of BoundedBuf class that is FIFO border buffer that design.

Each method of class defined in class to handle instance variable. Become means to evaluate GetGuard () putGuard () guard declarer, and is method to achieve work that handle state of getBody () putBody () object.

Two instance variable rear and front are pointer variable that store cell to buffer buf and use when remove.

Public part declared interface method get () put () of BoundedBuf class.

In case is very Get () getGuard (), achieving getBody (), put () is similar.

```
 class BoundedBuf {
   protected:
     const int SIZE;
     int buf[SIZE];
     int rear, front;
     bool getGuard() {
          return front != rear;         }
     int getBody(bool) {
          int res = buf[front++];
          front %= SIZE;
          return res;         }
     bool putGuard() {
           return ((rear - front) % SIZE) < (SIZE -
1));}
     bool putBody(int args) {
          buf[rear++] = args;
          rear %= SIZE;
          return true;        }
   public: void get() {
           if (getGuard()) getBody(bool);   }
           void put() {
           if (putGuard()) putBody(args);   }
   }
```

Fig. 1 BoundedBuf class

3.2 Get2Buf class

Fig. 2 is Get2Buf class that add get2 () that remove the oldest two cell in buffer buf to solvemore than division of permission state.

This is that add get2 () because it is impossible that call and achieves series get () of pair.

Two body and guard part defined in get2Guard () subclass that confirm that cell more than at least two exists to buffer buf before permit get2Body () practice. If getGuard () result bears, can remove two element consecutively achieving get2Body () because cell more than at least two exists within buffer.

```
  class Get2Buf : BoundedBuf {
    bool get2Guard() {
          return ((rear - front) % SIZE) >
1;          }
    pair(int, int) get2Body(bool) {
         int temp1 = front;
         front++;
         front %= SIZE;
         int temp2 = front;
         front++;
         front %= SIZE;
         return create_pair(buf[temp1],
      buf[temp2]); }
  public:
    void get2() {
      if (get2Guard()) get2Body(bool);
}
   }
```

Fig. 2 Get2Buf class

3.3 GgetBuf class

Justice of GgetBuf class to solve more than past susceptibility of permission state is same with Fig. 3. Added get () gget () that gouge GgetBuf.

Achieve according to state offered after get () put () practice to enable () disable () transition part that is GgetBuf's member function. After put () practice according to Gget () transition disable do, and after get () practiceenable because is done gget () achievement control can. Therefore, need not to redefine put () get (), for past state 'After-put' More than past susceptibility of permission state does not happen without adding instance variable that is.

```
  class GgetBuf : BoundedBuf {
   public:
     void gget() {
       if (getGuard())
           getBody(bool);
       put.disable(gget);
       get.enable(gget);        }
  }
```

Fig. 3 GgetBuf class

3.4 LockBuf class

Justice of LockBuf class to solve more than alteration of permission state is same with Fig. 4.

Lock () and unlock () of new LockBuf class added.

Achieving lockBody () according to Lock () lockGuard () achievement result relevant method to do lock.

Lock () transition part all methods that except unlock () to do disable method be not achieved make. And unlockGuard () lock disturbs concurrency of done object, and unlock () transitions part methods to do enable method in state that achievement is possible make.

```
 class LockBuf: BoundedBuf {
  protedted:
    bool lockGuard() const {
         return true;      }
    bool lockBody(bool) {
         return true;      }
    bool unlockGuard() const {
         return true;      }
    bool unlockBody(bool) {
         return true;      }
  public:
    bool locked(false);
    bool lock() {
          if (lockGuard()) lockBody(bool); }
    bool unlock() {
          if (unlockGuard()) { unlockBody(bool);
                                lock.disableAll();
                               lock.enable(unlock);
                               unlock.enableAll(); }
          }
  }
```

Fig. 4 LockBuf class

4. Simulation and assay

Simulation for C++ class that design to solve problem more than inheritance achieved using standard C++ language in NT, embodied following substance to foundation.

First, for same time achievement of simulation, created thread that request achievement of method that is definedin each class. thread did achievement request order of method as is different whenever achieve simulation using function that generate random number.

Second, achieved several reconsideration simulations to request method and produce various result for visual point that achieve. Because simulation result is same with table 1 and request about each method achievement is random from table 1, request order of method appears as is different whenever achieve simulation.

Attached serial numbers via request time of method, and request expresses thread's method request time.

Exe_start and Exe_end express achievement beginning time of method and achievement end time each, and unit is tic counter value (1/1000) during standard time. Buf_state each method achieve that display state since and buffer buf defined by circulation system queue that can store 10 cell . And cost of each element stored within buffer is wave and value (proper move) after have achieved method.

Table 1 Achievement result of each methods (Unit : tick)

Requst method		Req	Exe_ start	Exe_ end	Buf_state									
					0	1	2	3	4	5	6	7	8	9
1	put	406	406	406	10									
2	get2	408	-	-	10									
3	put	408	416	416	10	20								
4	gget	412	-	-	10	20								
5	get2	414	426	426										
6	get	422	-	-										
7	gget	424	-	-										
8	put	426	456	456			30							
9	get	428	456	456										
10	gget	434	-	-										
11	get2	434	-	-										
12	get	436	-	-										

Before early state and achievement of each method that is defined in class that design / state transition about great kindness state relation with table 2, Fig. 5 same. table 2 relationship 'State' being state after have run early situation and each method, ' _ ' relevant method displays that was not achieved.

State transition for table 2 cover of Fig. 5 is relation diagram.

Table 2 State transition table of between state and method

Method State	Unlock()	Lock()	Put()	Get()	Get2()	Gg et()
S_0	S_0	S_1	S_3	S_2	-	-
S_1	S_0	S_1	-	-	-	-
S_2	S_0	S_1	S_3	S_2	S_2	S_2
S_3	S_0	S_1	S_3	S_2	S_2	-

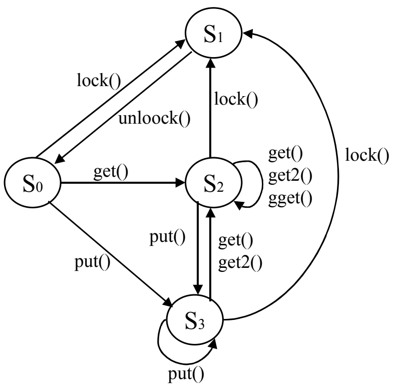


Fig. 5 Method and state transition diagram

When cell more than two exists to buffer from table 1, by get2 () achievement is possible, and without redefining put () get () when achieve get2 () in Get2Buf class that add get2 () by this achieve reusing, more than divisionof permission state does not happen. After Put () practice, more than past susceptibility of permission state does not happen by doing so that may do gget () disable and prevent achievement, and after get () achievement gget () achievement to do enable. And put () get () get2 () gget () achievement according to lock () unlock () achievement is possible or impossible from table 1.

That is, by practice about request of each method is impossible and begins self-discipline after get into unlock state while wait for an opportunity until get into unlock's state in case is lock state, more than alteration of permission state does not happen.

5. Conclusion

More than inheritance happens by conflict between concurrency and concurrency when re-justice of inherited method is essential to keep synchronization limitation condition of same time object, and much researches to solve these problem are held.

Designed C++ class that guard method is added lest more than inheritance of this paper should happen. So that, processed problem more than inheritance as minimize re-right of method code and avoid method body's correction. Also, offer advantage that can make use of class unartificially, and do so that minimize re-right of inherited method. So that can use easily making classes that design hereafter to library, wish to do, and study about solution of problem. More than inheritance that happen in other object intention language.

In this paper, studied by 2004 flues seoil college school ground research expense supports

6. Reference

[1] C. Barry, L. Leung, P. Peter, K. Chiu, ″Behaviour equation as solution of inheritance anomaly in concurrent object-oriented programming languages,″ IEEE′96, Proceedings of PDP′96, pp. 360-366, 1996.

[2] G. Agha, P. Wegner, A. Yonezawa, ′Research Directions in Concurrent Object-Oriented Programming′, Massachusetts, MIT Press, 1993.

[3] S. E. Mitchell, A. J. Wellings, ″Synchronization, concurrent object-oriented programming and the inheritance anomaly,″ Computer Language, Vol. 22, No. 1, pp. 15-26, 1996.

[4] J. Meseguer, ″Solving the inheritance anomaly in concurrent object-oriented programming,″ ECOOP′93, Object-Oriented.

[5] L. Thomas, ″An Object-Oriented Concurrent Language for Extensibility and Reuse of Synchronization Components,″ Computers and Artificial Intelligence, Vol. 15, No. 5, pp. 437-457, 1996.

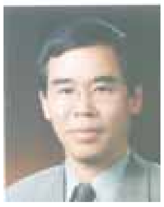
저 자 소 개

조 명 현(曺 明 絃)

1961년 5월 22일 1992년 조선대학교 대학원 전기공학과 박사 현재 서일대학 전기과 부교수 주관심분야 : 제어및계측 로봇, 프로그램

Tel : 018-789-1083

E-mail : cmhyun01@mail.seoil.ac.kr



이 명 언(李 明 彦)

1960년 6월 11일 2004년 건국대학교 대학원 전기공학과 박사 현재 서일대학 전기과 겸임교수 주관심분야 : 기기및전력전자, 전기설비설계, 프로그램

Tel : 011-795-1993

E-mail : emyungun@hanmail.net