2015년 2학기 운영체제

프로젝트 1 리포트

5조

2009-11604 정태호

2009-11779 이동우

2013-11399 박병준

2013-11431 정현진

**1. Wait queue design**

- Data Structures

Struct thread에 int64\_t wait\_start, int64\_t wait\_length, bool wait\_flag 추가

- Algorithms

Timers.c에서 timer\_sleep 함수에 계속적으로 thread\_yield 함수를 호출하는 부분 삭제 및 thread\_sleep함수를 호출하도록 변경. Thread\_sleep함수와 sleep상태의 thread를 깨우는 wake\_thread함수는 다음과 같다. 이 때, wait\_list는 static struct list wait\_list; 와 같이 선언되었다. Thread\_init 함수에서 list\_init(&wait\_list); 명령어로 초기화시킨다. Wake\_thread 함수는 next\_thread\_to\_run 함수에서 초기에 호출한다.

void thread\_sleep (int64\_t start, int64\_t ticks){

struct thread \*cur = thread\_current ();

enum intr\_level old\_level;

ASSERT(!intr\_context());

old\_level = intr\_disable ();

cur->wait\_start = start;

cur->wait\_length = ticks;

cur->wait\_flag = true;

list\_insert\_ordered (&wait\_list, &(cur->elem), is\_less\_time, NULL);

thread\_block();

intr\_set\_level (old\_level);//return to the original interrupt level

}

Thread의 우선순위를 정하기 위해 두 threads의 우선순위를 비교하는 is\_less\_time()함수는 다음과 같다.

void wake\_thread (){

struct thread\* th;

while(!list\_empty(&wait\_list)) {

th = list\_entry(list\_front (&wait\_list), struct thread, elem);

if(timer\_elapsed(th->wait\_start) >= th->wait\_length){

list\_pop\_front(&wait\_list); th->wait\_length = 0;

th->wait\_start = 0; th->wait\_flag = false;

thread\_unblock(th);

} else break;

}

}

bool is\_less\_time (const struct list\_elem\* a, const struct list\_elem\* b, void \*aux UNUSED){

struct thread \*thread\_a = list\_entry (a, struct thread, elem);

struct thread \*thread\_b = list\_entry (b, struct thread, elem);

if ((thread\_a->wait\_start + thread\_a->wait\_length) < (thread\_b->wait\_start + thread\_b->wait\_length))

return true;

else if( (thread\_a->wait\_start + thread\_a->wait\_length) > (thread\_b->wait\_start + thread\_b-wait\_length))

return false;

else { // if two itmes have same waiting time, compare with priority.

if(thread\_a->priority > thread\_b->priority) return true;

else return false;

}

}

**2. Priority donation design**

- Data Structures

Struct lock에 struct list\_elem elem 멤버변수 추가, struct thread에 int old\_priority, int set\_priority, struct list lock\_list 멤버변수 추가

- Algorithms

Thread.c의 init\_thread함수에서 멤버변수 old\_priority, set\_priority와 lock\_list를 각각 초기화한다.

void thread\_set\_priority (int new\_priority) {

struct thread \*curr = thread\_current();

if(list\_empty(&curr->lock\_list)) { // Set priority if only there's no donated lock in current thread.

curr->priority = new\_priority;

thread\_yield(); // If priority were changed, change the running thread immediately.

}else { // otherwise, remember set-value to wait until lock release.

if(new\_priority >= curr->priority) {

curr->priority = new\_priority;

thread\_yield();

} else { curr->set\_priority = new\_priority; } } }

한편, wait\_list와 ready\_list에 thread의 elem을 넣을 때 기존 코드에서는 list\_push\_back으로 단순히 FCFS방식으로 실행했다면, 수정된 코드에서는 각 쓰레드의 priority를 기준으로 리스트를 정렬하도록 하였다. 이 과정에서 기본 라이브러리의 list\_insert\_ordered() 함수를 사용하고, 두 쓰레드의 priority를 비교하는 함수로 아래와 같은 is\_higher\_priority() 함수를 사용하였다. 또한 priority inversion 현상을 제거하기 위하여 synch.c파일에 다음과 같은 priority\_donation(),priority\_rollback() 함수를 정의하여 사용하도록 하였다.

bool is\_higher\_priority (const struct list\_elem \*a, const struct list\_elem \*b, void \*aux UNUSED) {

struct thread \*thread\_a = list\_entry(a, struct thread, elem);

struct thread \*thread\_b = list\_entry(b, struct thread, elem);

if(thread\_a->priority > thread\_b->priority) return true;

else return false; }

void priority\_donation(struct lock \*lock) {

struct thread \*holder = lock->holder; struct thread \*curr = thread\_current();

if( holder != NULL && curr != NULL ) {

if ( holder->priority < curr -> priority ) {

if(holder->old\_priority == -1) holder->old\_priority = holder->priority; // Save priority.

holder->priority = curr->priority; // donate

if(!is\_in\_list(&holder->lock\_list, &lock->elem)) list\_push\_front (&holder->lock\_list, &lock->elem);

// Add a lock to donated list of holder. ( holder reveices donation )

if(holder->wait\_lock != NULL) priority\_donation(holder->wait\_lock);

} } }

void priority\_rollback(struct lock \*lock) {

struct thread \*curr = thread\_current();

if(is\_in\_list(&curr->lock\_list, &lock->elem)) {

list\_remove(&lock->elem); // remove from lock\_list(donated lock list) of current thread

int highest\_priority = curr->old\_priority; int origin\_priority = curr->old\_priority;

struct list\_elem \*e; struct list \*locks = &curr->lock\_list;

for(e = list\_begin(locks); e != list\_end(locks); e = list\_next(e)) {

struct semaphore \*sema = &list\_entry(e, struct lock, elem) -> semaphore;

struct list \*waiters\_list = &sema->waiters;

struct thread \*max\_thread = list\_entry(list\_front(waiters\_list), struct thread, elem);

if(max\_thread != NULL) {

if(max\_thread->priority > highest\_priority) highest\_priority = max\_thread->priority;

}

}

curr->priority = highest\_priority; // priority roll-back

if(origin\_priority == highest\_priority) curr->old\_priority = -1;

if(curr->set\_priority != -1) {

int tmp = curr->set\_priority; curr->set\_priority = -1;

thread\_set\_priority(tmp);

} } }

**3. Testing results of ‘alarm-multiple’**

PiLo hda1

Loading..........

Kernel command line: -q run alarm-multiple

Pintos booting with 4,096 kB RAM...

383 pages available in kernel pool.

383 pages available in user pool.

Calibrating timer... 204,600 loops/s.

Boot complete.

Executing 'alarm-multiple':

(alarm-multiple) begin

(alarm-multiple) Creating 5 threads to sleep 7 times each.

(alarm-multiple) Thread 0 sleeps 10 ticks each time,

(alarm-multiple) thread 1 sleeps 20 ticks each time, and so on.

(alarm-multiple) If successful, product of iteration count and

(alarm-multiple) sleep duration will appear in nondescending order.

(alarm-multiple) thread 0: duration=10, iteration=1, product=10

(alarm-multiple) thread 1: duration=20, iteration=1, product=20

(alarm-multiple) thread 0: duration=10, iteration=2, product=20

(alarm-multiple) thread 2: duration=30, iteration=1, product=30

(alarm-multiple) thread 0: duration=10, iteration=3, product=30

(alarm-multiple) thread 3: duration=40, iteration=1, product=40

(alarm-multiple) thread 1: duration=20, iteration=2, product=40

(alarm-multiple) thread 0: duration=10, iteration=4, product=40

(alarm-multiple) thread 4: duration=50, iteration=1, product=50

(alarm-multiple) thread 0: duration=10, iteration=5, product=50

(alarm-multiple) thread 2: duration=30, iteration=2, product=60

(alarm-multiple) thread 1: duration=20, iteration=3, product=60

(alarm-multiple) thread 0: duration=10, iteration=6, product=60

(alarm-multiple) thread 0: duration=10, iteration=7, product=70

(alarm-multiple) thread 3: duration=40, iteration=2, product=80

(alarm-multiple) thread 1: duration=20, iteration=4, product=80

(alarm-multiple) thread 2: duration=30, iteration=3, product=90

(alarm-multiple) thread 4: duration=50, iteration=2, product=100

(alarm-multiple) thread 1: duration=20, iteration=5, product=100

(alarm-multiple) thread 3: duration=40, iteration=3, product=120

(alarm-multiple) thread 2: duration=30, iteration=4, product=120

(alarm-multiple) thread 1: duration=20, iteration=6, product=120

(alarm-multiple) thread 1: duration=20, iteration=7, product=140

(alarm-multiple) thread 4: duration=50, iteration=3, product=150

(alarm-multiple) thread 2: duration=30, iteration=5, product=150

(alarm-multiple) thread 3: duration=40, iteration=4, product=160

(alarm-multiple) thread 2: duration=30, iteration=6, product=180

(alarm-multiple) thread 4: duration=50, iteration=4, product=200

(alarm-multiple) thread 3: duration=40, iteration=5, product=200

(alarm-multiple) thread 2: duration=30, iteration=7, product=210

(alarm-multiple) thread 3: duration=40, iteration=6, product=240

(alarm-multiple) thread 4: duration=50, iteration=5, product=250

(alarm-multiple) thread 3: duration=40, iteration=7, product=280

(alarm-multiple) thread 4: duration=50, iteration=6, product=300

(alarm-multiple) thread 4: duration=50, iteration=7, product=350

(alarm-multiple) end

Execution of 'alarm-multiple' complete.

Timer: 890 ticks

Thread: 599 idle ticks, 293 kernel ticks, 0 user ticks

Console: 2952 characters output

Keyboard: 0 keys pressed

Powering off..

**4. Testing results of ‘alarm-priority’**

PiLo hda1

Loading..........

Kernel command line: -q run alarm-priority

Pintos booting with 4,096 kB RAM...

383 pages available in kernel pool.

383 pages available in user pool.

Calibrating timer... 204,600 loops/s.

Boot complete.

Executing 'alarm-priority':

(alarm-priority) begin

(alarm-priority) Thread priority 30 woke up.

(alarm-priority) Thread priority 29 woke up.

(alarm-priority) Thread priority 28 woke up.

(alarm-priority) Thread priority 27 woke up.

(alarm-priority) Thread priority 26 woke up.

(alarm-priority) Thread priority 25 woke up.

(alarm-priority) Thread priority 24 woke up.

(alarm-priority) Thread priority 23 woke up.

(alarm-priority) Thread priority 22 woke up.

(alarm-priority) Thread priority 21 woke up.

(alarm-priority) end

Execution of 'alarm-priority' complete.

Timer: 588 ticks

Thread: 474 idle ticks, 116 kernel ticks, 0 user ticks

Console: 837 characters output

Keyboard: 0 keys pressed

Powering off..