Mon 18 Mar 2015 02:20:57 AM EET

GEBZE TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
CSE 312 OPERATING SYSTEM
2014-2015 SPRİNG
PROJECT 1

@date 10/03/2015
@author Alican OZER-Selim AKSOY

Alican OZER - 111044070
Selim AKSOY - 111044072

/***** ALARM CLOCK **********/

EKLENDİ: static struct list uyuma_listesi;// uyumasi gereken threadlerin listelendigi kisimdir.

-->bool compare_timeto_wakeup_ticks (const struct list_elem *first,const struct list_elem *second);
threadlerin uyuma surelerinin karsilastirilmasi bu sayede uyuma_listesi uzerinde sirali bir sekilde tutulmasini sagladik.

Degistirilen fonksiyonlar

void timer_sleep (int64_t ticks);
// timer sleep fonksiyonu busy waiting yapiyordu.Her threadin
uyuma surelerinin hesaplanmasi ve uyuma_listesi icerisine sirali olarak eklenmesi
gerceklestirilmistir.

static void timer_interrupt (struct intr_frame *args UNUSED) function;

//Her bir timer tick interrupt oldugunda sistem uzerindeki threadlerin uyuma
suresi tamamlananlarin uyandirilmasi gerekmektedir.Uyandirilan threadlerin o
anda calisan current threadin prioritysi ile karsilastirilarak eger kendisinden
buyuk bir priority soz konusu olma durumunda o anki calisan threadin yield
olmasi soz konusudur.

RE-IMPLEMENTED: timer_sleep (int64_t ticks) function:

wake_up time i guncelledik ve sleep ettik.hemen yield etmek(ready liste koymak)
yerine uyanana kadar blockladik.Bu sekilde busy waitingi onledik.

RE-IMPLEMENTED: timer_interrupt (struct intr_frame *args UNUSED);

call thread_has_max_priority();(check the max priority)

Burda threadlerin wake_up_time gore uyandirma islemini ve BSD Schedular için
bazi degisikler yaptik.

Zaman harcamasini azaltma durumunu su sekilde minimize edebiliriz,
her timer_interrupt cagirildiginda sistem uzerinde timer_tick fonksiyonu
cagirilir bu sayede sistem uzerindeki o anki tick degerine ulasilir.Sleep
listesindeki threadler priority degerlerine gore siralanirsa o durumda ilk
bastaki thread en buyuk priority'e sahip olucaktir.O anki calisan threadin
priority'si uyuma_listesindekiler ile karsilastirilir en buyuk olan priority 'e
sahip olan thread current thread olarak devam eder.Total Cost : Liste uzerinde
gezinerek en buyuk priority 'e sahip olan threadin current thread ile
karsilastirilmasi durumu O(n) cost olucaktir.

/*** SYNCHRONIZATION ********/

RE-IMPLEMENTED: timer_sleep();

Interruptlarin enable-disable mekanizmasini ve thread_block fonksiyonunu(1 den
fazla thread oldugunda threadi uyutur ve baska bir threadin çalismasini saglar)
kullanarak race conditionu onledik.

Timer interrupt fonksiyonu thread_tick fonksiyonu çagrisindan sonra,ilgili
thread blocked durumundaysa thread uyumu zamaninin bitimine kadar uyutulacak.
Her thread kendi uyuma zamani kadar bekletildiginden threadler arasindaki race
condition durumu engellenmis olur.

------- ACIKLAMALAR ----------
Bu dizayn yapisini kullanmamizin sebebi oncelikle ,uyuyacak olan threadlerin
nerede tutulmasi gerektigi konusundaydi daha sonrasinda thread status icerisinde
sleep olduguna dair bir durum (THREAD_SLEEP) olusturulmasi konusunda fikir
danisikligina basladik.En sonunda zaten pointerlar uzerinden yaptigimiz icin
uyuma_listesi tutulmasi yeterli bir durum olucaktir diye dusunduk daha sonrasin da
bu pointer uzantilariyla ilgili yerlere gecis sagladik.uyuma_listesiesinde tutulan
thread yapilari ayni sekilde ready_listesinde de tutulmaktadir.Bu sayede
THREAD_SLEEP gibi bir status tanimlanmasina gerek kalmadan thread_block()
fonksiyonu ile ready_listesi uzerinde blocklanarak devam etmesi bir problem
cikarmayacagini anladik.Ayrica timer_interrupt fonksiyonu uzerinde uyuma suresi
tamamlanan artik uyanmasi gereken threadlerin sadece uyuma_listesi uzerinden
silinmesini ve ayrica o anki calisan threadin priority'si ile uyandirilan

threadlerin prorityleri karsilastirmasi yapildi ve o anki threadten daha buyuk priority'e sahip olan thread current thread olarak (thread_yield) set edilmesi saglandi.

/***** PRIORITY SCHEDULING **********/

EKLENDİ: bool compare_priority (const struct list_elem *first,const struct list_elem *second);

void thread_has_max_priority (void); // current thread ile ready_listesindeki threadleri karsilastirir.bu durumda daha buyuk bulunursa thread_yield() edilir.

void priority_donation_solve (void);

void remove_donation_list_waiting_lock (struct lock *lock);// o anki calisan threadin lock listesinde parametre olarak verilen lock'u bekleyenler silinir.

void priority_check_from_donation_list (void);// donation listesindekilere gore priority update edilir

RE-IMPLEMENTED:

tid_t thread_create (const char *name, int priority,thread_func *function, void *aux);
void thread_unblock (struct thread *t);
void thread_yield (void);
void thread_set_priority (int new_priority);

--------ACIKLAMALAR----------
sema_up fonksiyonunda semaforu bekleyen listeyi priority e gore sirala, en yuksek olani listeden al ve o thread i unblock hale getir.Ready liste eklenen thread current threadten daha yuksek prioritye sahipmi diye kontrol et.Condition variable lar ve locklar içinde bu fonksiyon kullaniliyor.Bu sekilde en yuksek prioritye sahip thread ilk once uyandiriliyor.

schedule() fonksiyonunda next_thread_to_run() fonksiyonu sayesinde en yüksek priorty thread seçilirve uyandırılır.

lock_acquire fonksiyonunda,

Interruptlar disable edilir.
Current threadin waitlock u bu lock yapilir.
Donation_liste ,donationa sebeb olan eleman eklenir
sema_down (&lock->semaphore);
Interruptlar enable edilir.

sema_down fonksiyonunda çagrilan priority_donation_solve() fonksiyonu sayesinde nested donation çozulur

interruptlar disable edilir
Bu lock u bekleyenler donation listten silinir.
Priority donation liste gore update edilir.
sema_up fonksiyonu çagirilarak en yuksek priorityli thread uyandirilir.
interruptlar enable edilir.

Priority set edildiginde donation a sebeb olabilir,ready listte kendinden daha yuksek priortye sahip bir thread olabilir.Bu durumlar kontrol edilmeli ve duruma gore priority_donation_solve() ya da thread_has_max_priority() fonksiyonlari çagirilarak bu durum handle edilmeli.

---ACIKLAMALAR----------
Lock'lar icin liste olusturduk.scheduler durumda icinde priority donation tamamlandiginda context switch artisi olucaktir.Bunun sebebi lock_listesi icerisinde cok fazla lock olmasi durumunda.Eger thread lock'i release etmeden olmesi durumunda problem ortaya cikicaktir.Durum her bir threadin bir lock tutmasi ve o lock bekleyen threadlerin bilgilerinin tutulmasi gerekmektedir.Bu sekilde donation olma durumu ve lock release edilmesi,Nested donation durumlari kontrol altina alinacaktir.Bu durumdan dolayi bu sekilde bir dizayn tercih ettik.
