INF333 2023-2024 Spring Semester

SEFILLER
Kerim Ayberk Çıtak 1234.ayberk@gmail.com
Onur Alp Gündüz onuralpgunduz@gmail.com

Homework 01 Design Document

The following is the sample design document for this homework. We will provide the general layout and the questions, you will provide answers in the `quote` environment.

1 Preliminaries

Q: If you have any preliminary comments on your submission, notes for the TAs, or extra credit, please give them here.

Q: Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, and lecture notes.

- https://www.scs.stanford.edu/24wi-cs212/pintos/pintos_10.html#SEC151
- https://www.scs.stanford.edu/24wi-cs212/pintos/pintos_2.html
- https://www.cse.iitd.ac.in/ sbansal/os/previous_years/2013/pintos/doc/pintos_html/files.html
- https://teams.microsoft.com/l/message/19:8624f6e6-2b14-4d7e-a4bd-a60fe485a21e_cefdce80-113a-41b3-a5a6-22b398851b05@unq.gbl.spaces/1712670444726?context=%7B%22contextType%22%3
- https://stackoverflow.com/questions/77047057/bochs-display-library-nogui-not-available

2 Join

2.1 Data Structures

Q: Copy here the declaration of each new or changed 'struct' or 'struct' member, global or static variable, 'typedef', or enumeration.

Identify the purpose of each in 25 words or less.
A:

- ```
  struct thread
  {
    //Implementations for alarm-clock

    int64_t waketick;

    //End of Implementations
  };
  ```

  .
  - `waketick`: Uyku modundan uyandırılma zamanını belirten işaretçi.
- ```
  struct list sleep_list;
  ```

  - `sleep_list`: Uyku halindeki processleri yönetmek için kullanılan bağlı liste. Bu liste, uyku modunda bekleyen processleri takip etmek için kullanılır.
- ```
  //thread.c dosyası içinde
  extern struct list sleep_list;
  ```

  - `extern struct list sleep_list;`: Uyku halindeki processleri yönetmek için kullanılan bağlı listeye dışarıdan erişimi sağlayan bir dış bağlantı deklarasyonu. Bu liste, uyku modunda bekleyen processleri takip etmek için kullanılır.

2.2 Algorithms

Q: Briefly describe your implementation of thread_join() and how it interacts with thread termination.
A:

- **Genel Bakış:**

  Yaptığımız implementationda thread_join() fonksiyonu bulunmamaktadır. processler arasındaki senkronizasyonları farklı fonksiyonların ve elementlerin birleşimi ile sağladık.
- **Thread Sonlandırma:**
  - **Waketick:** Bir thread `timer_sleep()` fonksiyonunu çağırdığında, waketick değeri, thread'in uyanması gereken gelecekteki zamana ayarlanır.
  ```

- **Uyku Listesi:** `sleep_list`, thread'leri waketick değerlerine göre sıralı bir şekilde tutar.
- **Zamanlayıcı Kesme İşleyicisi:** Zamanlayıcı kesme işleyicisi `timer_interrupt()` fonksiyonu:
  * Küresel ticks sayacını artırır.
  * Her thread'in `elapsed_ticks` sayacını güncellemek için `thread_tick()` fonksiyonunu çağırır.
  * `sleep_list`'te gezinir:
    - Bir thread'in `waketick` değeri mevcut `ticks` değerinden küçük veya ona eşitse, thread listeden kaldırılır ve engellenmesi kaldırılır.

Q: What steps are taken to minimize the amount of time spent in the timer interrupt handler?
A: Zamanlayıcı kesme işleyicisinde geçirilen süreyi minimize etmek için aşağıdaki adımlar alınır:

- Uyuyan processlerin listesi üzerinde işlemler yapılırken, mümkün olduğunca kısa sürelerde kalınmaya çalışılır.
- Uyku listesi üzerinde sıralı bir şekilde işlem yapılır ve sadece uyanması gereken processler uyandırılır.

2.3 Synchronization

Q: Consider parent thread `P` with child thread `C`. How do you ensure proper synchronization and avoid race conditions when `P` calls `wait(C)` before `C` exits? After `C` exits? How do you ensure that all resources are freed in each case? How about when `P` terminates without waiting, before `C` exits? After `C` exits? Are there any special cases?

Q: How are race conditions avoided when multiple threads call timer_sleep() simultaneously?
A: Birden fazla iş parçacığının aynı anda timer_sleep() çağrısında bulunması durumunda yarış koşulları, bir kilit mekanizması kullanılarak önlenebilir. İş parçacıkları timer_sleep() çağrısı yaptığında, ilgili kod bloğu kritik bir bölgeye alınarak, yalnızca bir iş parçacığının bu bölgeye erişmesine izin verilir. Diğer processler, kritik bölgede olan iş parçacığının işlemini bekler ve ardından sırayla işlemlerini gerçekleştirirler.
Q: How are race conditions avoided when a timer interrupt occurs during a call to timer_sleep
A: Bir timer_sleep() çağrısı sırasında zamanlayıcı kesintisi gerçekleştiğinde, yarış koşulları kesinti işlemi tarafından ele alınarak önlenebilir. Kesinti işlemi, uyku listesindeki processlerin uyku sürelerini kontrol eder ve uyandırılması gereken processleri belirler. Bu işlem sırasında, kesinti işlemi tarafından kullanılan veri yapılarına erişim sırasında kesinti

işlemi devam eder ve diğer processlerin müdahalesini engeller. Bu sayede, uyku sırasında olası yarış koşulları önlenebilir ve tutarlı bir işlem akışı sağlanır.

2.4 Rationale

Q: Critique your design, pointing out advantages and disadvantages in your design choices.
A:Tasarımımızın avantajları arasında, uyku ve uyanma işlemlerinin etkin bir şekilde yönetilmesi, zamanlayıcı kesmesi işleyicisinin hızlı bir şekilde çalışması ve processler arasında uygun senkronizasyonun sağlanması yer alır. Dezavantajları arasında ise, bazı durumlarda kesmelerin devre dışı bırakılması gerekebilir ve bu da sistem performansını etkileyebilir. Ayrıca, çok sayıda iş parçacığının aynı anda uyku listesine eklenmesi veya çıkarılması gerektiğinde, uyku listesi üzerindeki işlem yoğunluğu artabilir.

3 Priority Scheduling

3.1 Data Structures

Q: Copy here the declaration of each new or changed 'struct' or 'struct' member, global or static variable, 'typedef', or enumeration. Identify the purpose of each in 25 words or less
A:

- `struct thread`'e `int basepriority` eklendi: Bir iş parçacığının orijinal önceliğini takip eder.
- `struct thread`'e `struct list pot_donors` eklendi: Öncelik bağışı için potansiyel bağışçıların listesini tutar.
- `struct thread`'e `struct lock *blocked` eklendi: Bir iş parçacığının bloke olduğu kilidi gösterir, varsa.

Q: Describe the sequence of events when a call to lock_acquire() causes a priority donation. How is nested donation handled?

- `lock_acquire()` çağrıldığında ve kilit düşük öncelikli bir iş parçacığı tarafından tutuluyorsa, öncelik bağışı gerçekleşir.
- Sıra:
  - Düşük öncelikli iş parçacığı önceliğini kiliti tutan yüksek öncelikli iş parçacığına bağışlar.

  - Yüksek öncelikli iş parçacığının temel önceliği bağışlanan öncelikle güncellenir.
  - İç içe bağış varsa, bağışlanan öncelik kilidin serbest bırakılmasına kadar zincir boyunca yayılır.

- İç içe bağış otomatik olarak işlenir çünkü bir iş parçacığı önceliğini başka bir iş parçacığına bağışladığında ve bu iş parçacığı da başka bir kilidi bekliyorsa, önceliğini zincirdeki sonraki kilidi tutan iş parçacığına daha da bağışlar.

Q: Describe the sequence of events when lock_release() is called on a lock that a higher-priority thread is waiting for.

- `lock_release()` bir kilidi serbest bırakıldığında, potansiyel bağışçılar varsa öncelik geri yükleme gerçekleşir.
- Sıra:
  - Kilidi serbest bırakılır ve eğer `pot_donors` listesinde processler varsa, öncelikleri temel önceliklerine geri yüklenir.

3.2 Synchronization

Q: Describe a potential race in thread_set_priority() and explain how your implementation avoids it. Can you use a lock to avoid this race?
A: thread_set_priority() fonksiyonunda olası bir yarış durumu, aynı anda birden fazla iş parçacığının aynı önceliği ayarlamaya çalışması durumunda ortaya çıkabilir. Örneğin, processler A ve B aynı anda thread_set_priority() fonksiyonunu çağırarak önceliklerini değiştirmeye çalışabilirler. Bu durumda, hangi iş parçacığının önceliği daha önce ayarlanacak belirsizdir ve yanlış sonuçlar elde edilebilir.

Bizim implementasyonumuzda bu yarış durumunu engellemek için kilit mekanizması kullanır. thread_set_priority() fonksiyonu çağrıldığında, öncelik değerini ayarlamak için bir kilit edinilir. Bu kilit, sadece bir iş parçacığının öncelik değerini değiştirmesine izin verir ve diğer processlerin aynı anda müdahale etmesini engeller. Bu sayede, yarış durumları ve tutarsız sonuçlar önlenir.

Kilit mekanizması, herhangi bir zaman diliminde yalnızca bir iş parçacığının thread_set_priority() fonksiyonunu çağırarak öncelik değerini güncellemesini sağlar. Bu şekilde, doğru sonuçlar elde edilir ve yarış durumları önlenebilir.

3.3 Rationale

Q: Why did you choose this design? In what ways is it superior to another design you considered?

- Bu tasarım, öncelik bağışını etkili bir şekilde uygular.
- Üstünlük:
  - Öncelik bağışını etkili bir şekilde yönetir, böylece yüksek öncelikli processler kaynaklara hızlı erişim sağlanır.

4 Advanced Scheduler

4.1 Data Structures

Q: Copy here the declaration of each new or changed 'struct' or 'struct' member, global or static variable, 'typedef', or enumeration. Identify the purpose of each in 25 words or less.
A:

- thread.h dosyasında, struct thread yapısına nice ve recent_cpu üyelerini tanıttık. Bu üyeler, her bir iş parçacığı için güzel değer ve son CPU kullanımını temsil eder. Bunlar gelişmiş zamanlama algoritmaları için kullanılır.
- thread.c dosyasında, load_avg değişkenini global bir değişken olarak tanımladık. Bu değişken, zamanlama kararları için periyodik olarak güncellenen sistem yük ortalamasını tutar.

4.2 Algorithms

Q: Suppose threads A, B, and C have nice values 0, 1, and 2. Each has a recent_cpu value of 0. Fill in the table below showing the scheduling decision and the priority and recent_cpu values for each thread after each given number of timer ticks:

timer	recent_cpu			priority			thread
ticks	A	B	C	A	B	C	to run
0	0	0	0	63	61	59	A
4	4	0	0	62	61	59	A
8	8	0	0	61	61	59	B
12	8	4	0	61	60	59	A
16	12	4	0	60	60	59	B
20	12	8	0	60	59	59	A
24	16	8	0	59	59	59	C
28	16	8	4	59	59	58	B
32	16	12	4	59	58	58	A
36	20	12	4	58	58	58	C

Q: Did any ambiguities in the scheduler specification make values in the table uncertain? If so, what rule did you use to resolve them? Does this match the behavior of your scheduler?
A: Belirsizlik yoktu. Zamanlayıcı özelliklerindeki belirtilen formül ve kuralları kullandık ve bu, zamanlayıcının beklenen davranışıyla uyumluydu.

Q: How is the way you divided the cost of scheduling between code inside and outside interrupt context likely to affect performance?
A:Kesinti bağlamındaki kod mümkün olduğunca kısa olmalıdır çünkü kesinti sırasında işlemci diğer işlemlerden uzaklaşır. Dışındaki kod ise daha uzun süre çalışabilir çünkü işlemciyi kesinti sırasında engellemez.

4.3 Rationale

Q: Briefly critique your design, pointing out advantages and disadvantages in your design choices. If you were to have extra time to work on this part of the project, how might you choose to refine or improve your design?
A:Tasarımın genel olarak iyi olduğunu düşünüyoruz, ancak daha fazla zamanımız olsaydı, kodun daha temiz ve modüler olması için daha fazla dikkat gösterirdik. Ayrıca, performansı daha da iyileştirmek için bazı optimizasyonlar ekleyebilirdik.

Q: The assignment explains arithmetic for fixed-point math in detail, but it leaves it open to you to implement it. Why did you decide to implement it the way you did? If you created an abstraction layer for fixed-point math, that is, an abstract data type and/or a set of functions or macros to manipulate fixed-point numbers, why did you do so? If not, why not?

A: Sabit noktalı matematik işlemlerini bu şekilde uygulamaya karar vermemizin nedenleri şunlardır:

- Okunabilirlik ve Bakım Kolaylığı: Sabit noktalı matematik işlemlerini bir kütüphane olarak soyutlama, kodun okunabilirliğini artırır ve bakımını kolaylaştırır. Bu şekilde, sabit noktalı aritmetiği kullanan herhangi bir kod parçası, bu işlemlere doğrudan erişebilir ve anlayabilir.
- Modülerlik: Sabit noktalı matematik işlemleri için ayrı bir kütüphane oluşturmak, kodu modüler hale getirir. Bu sayede, bu işlemleri başka projelerde veya bileşenlerde de kullanmak daha kolay olur.
- Daha Az Hata: Ayrı bir kütüphane, sabit noktalı matematik işlemlerinin ayrıntılarını gizler ve doğru bir şekilde uygulandığından emin olmak için test edilebilir. Bu, hataları tespit etmeyi ve düzeltmeyi kolaylaştırır.
- Gelecek Uyarlanabilirlik: Ayrı bir sabit noktalı matematik kütüphanesi, gelecekteki gereksinimlere ve değişikliklere daha iyi uyum sağlar. Bu kütüphane, gerektiğinde güncellenebilir veya değiştirilebilir, böylece kodun daha esnek ve uyarlanabilir olmasını sağlar.

Bu nedenlerle, sabit noktalı matematik işlemlerini bir kütüphane olarak uygulamaya karar verdik. Bu şekilde, kodun okunabilirliği, bakımı ve uyarlanabilirliği artarken, hata yapma olasılığını azaltmış oluyoruz.

5 Survey Questions

Answering these questions is optional, but it will help us improve the course in future quarters. Feel free to tell us anything you want–these questions are just to spur your thoughts. You may also choose to respond anonymously in the course evaluations at the end of the quarter.

Q: In your opinion, was this assignment, or any one of the three problems in it, too easy or too hard? Did it take too long or too little time?
A: Şimdiye kadar karşılaşmadığımız zorlukta bir ödevdi. 3 hafta süre ödevi yapmak için yeterli bir süre lakin deneyim olarak vites arttırmakta zorluk çektik. **Q:** Did you find that working on a particular part of the assignment gave you greater insight into some aspect of OS design?
A: Bir işletim sisteminin donanımla nasıl iletişime geçerek görevleri sıraladığını öğrenmek gerçekten ilginçti. Genel olarak her bölümünde yeni şeyler öğrendiğimizi hissettik. **Q:** Is there some particular fact or hint we should give students in future quarters to help them solve the problems? Conversely, did you find any of our guidance to be misleading?
A: Lab derslerinde ödevin bazı bölümlerinin hocayla beraber yapılması öğrenciler açısından işleri büyük ölçüde kolaylaştıracaktır. Dokümantasyon okumanın yanında eşlikle Pintos'u keşfetmek çok daha faydalı olabilir.

Q: Do you have any suggestions for the TAs to more effectively assist students, either for future quarters or the remaining projects?

Q: Any other comments?