SİSTEM PROGRAMLAMA

UNIX te İleri Seviye Dosya ve Process Uygulaması

GİRİŞ

Bu deney üç kısımdan oluşuyor:

1. İlk kısımda öğrencinin verilen açıklamalara gore yeni bir program yazması bekleniyor.
2. İkinci kısımda öğrencinin ilk bölümde geliştirilen kodu açıklamalara gore değiştirmesi bekleniyor.
3. üçüncü kısımda, system çağrıları **fork**, **exec**, **sleep** ve **wait** detaylı bir şekilde inceleniyor, kullanılan programın başlangıç versiyonu Appendiks te verilmiştir.
4.

Verilen Kodlar da hatalar çıkarsa düzeltiniz(düzeltemezseniz bilgi veriniz).
Geliştirdiğiniz kodları, sorulara olan cevapları ve ekran görüntülerini **bir pdf uzantılı Rapor olarak hazırlayıp, yükleyiniz**.

Bölüm-1 : Yeni bir program yazma

Bu kısımda öğrenciden verilen şartlara uygun bir şekilde bir program yazması beklenmektedir. Programın akış şeması aşağıda verilmiştir. Parent dosya1' i yazmak için açar ve sonra child process yaratır. Sonra iki process paralel bir şekilde dosya1 'den bloklar şeklinde okuyup, biri (parent) dosya2 'ye ve diğeri (child) dosya3' e kopya etmeye çalışır. Herbir process dosya1 i kopyaladıktan sonra sonlanırlar. Komut satırından program 3 parametreli olarak aşağıdaki gibi çalışacaktır.

% progname dosya1 dosya2 dosya3

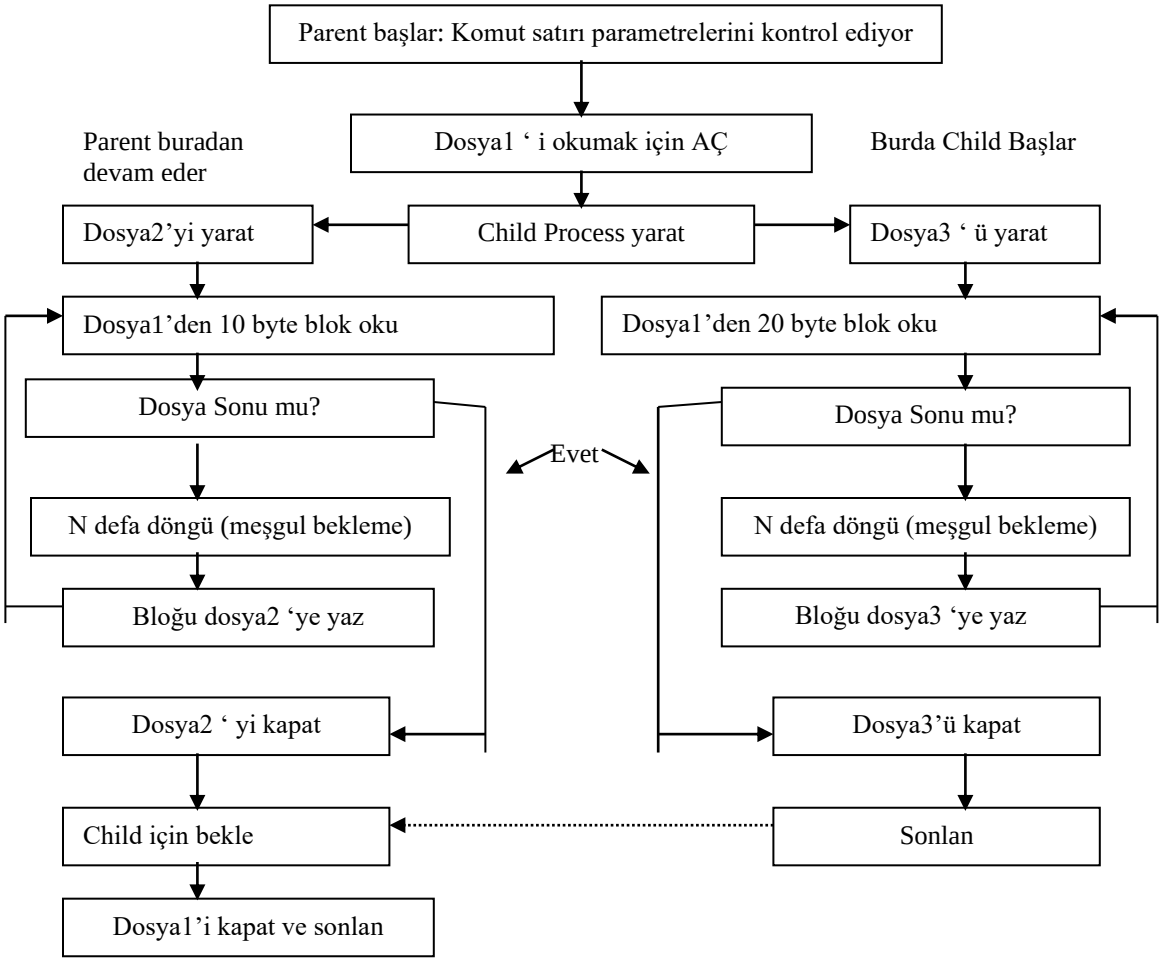
Deneyin bu bölümü için öğrenci aşağıdaki adımları gerçekleştirmelidir:

1. Yukarda anlatılan (akış diagramında detaylı verilen) programı home dizininizde lab3 diye bir dizin altında yazıp kaydediniz.
2. Programı verilen N değeri(akış diagramına bakınız) için derleyip birleştirin

3. Programı birçok kez aynı parametrelerle çalıştırın. Dosya1, Dosya2 ve Dosya3 birbirlerinin kopyasi oldu mu? Sonuçları açıklayın.

4. Adım 2 ve Adım3 'ü N = 10, 5000, 10000, ve 100000 değerleri için tekrar deneyin. Düşük ve Yüksek N değerlerinin etkisi ne oldu ? Dosya1, Dosya2 ve Dosya3 ne zaman birbirlerinin kopyasi oldu mu? Sonuçları açıklayın.

Bölüm-1 için Program Akış Şeması



Bölüm-2 : Bölüm-1 deki programın güncellenmesi

Deneyin bu bölümü için öğrenci aşağıdaki adımları gerçekleştirmelidir:

1. Bölüm-1 de geliştirilen programı farklı bir isimde kopya edip, bu dosya üzerinde çalışınız.
2. Programı, parent proses child yarattıktan hemen sonra text dosyasını okuması için güncelleyin. Benzer şekilde, child prosesin aynı dosya ismini okuması için güncelleme yapın.
3. programdaki meşgul beklemeleri kaldırın.
4. Dosya1, Dosya2 ve Dosya3 birbirlerinin kopyasi oldu mu? Sonuçları açıklayın.

Bölüm-3 : Sistem çağrıları **fork**, **exec**, **wait**, **sleep, system detaylı incelenmesi**

1. Appendix te verilen kodu yazınız.
2. Bu kodu derleyip, lab3 dizini altında oluşturunuz.
3. Kodun ne yaptığını bir iki cümle ile yazıp, Programı başlatınız. İki process (parent ve child) gönderdiği mesajların çıktı sırası herzaman doğru mu? bir çok kez test ediniz.
4. Mesajların doğru bir sırasını sağlamak için (önce child, sonra parent) programı uygun bir şekilde değiştiriniz.
5. system çağrısı **execl** (child process) çağrısını, system çağrısı **system** ile yer değiştirin ve gerekli değişiklikleri yapınız. Programın çalışmasında nasıl bir değişiklik oldu? execl ve system arasındaki temel farkı bir cümle ile yazınız.
6. Exec ailesinden execv 'nin kullanım şeklini anlamaya çalışınız, execl den farkını bir cümle ile ifade ediniz.

APPENDIKS

```
#include <sys/types.h>
#include <unistd.h>

int main(void) {
   pid_t pid;
   char *args[4];

   if ((pid = fork()) < 0) {
      perror("fork failed");
      exit(1);
   }

   if (pid == 0) {
      execl("/bin/echo", "echo", "Today's", "date", "is:", 0);

      /*
       * eğer exec başarılı ise buraya gelmeyecektir.
       */
      perror("exec failed in child");
      exit(1);
   }

   /*
    * Bu kodu sadece parent işletecek
    */
/*  burada execv için parametreleri hazırlıyor ..execv exec ailesinin başka bir üyesi*/
   args[0] = "date";
   args[1] = "+Date = %D  Time = %H:%M";
   args[2] = NULL;

   execv("/bin/date", args);
   /*
    * eğer exec başarılı ise buraya gelmeyecektir..
    */
   perror("exec failed in parent");
   exit(1);
}
```