BLGM 343 – DENEY 3*
PROGRAM GEÇİŞİ VE DOSYA İŞLEMLERİ

Amaçlar

1. POSIX tabanlı işletim sistemlerinde program geçişi
2. POSIX tabanlı işletim sistemlerinde dosya işlemleri
3. Program geçişi esnasında programa parametere gönderme
4. Yeni dosya oluşturma veya varolan dosyadan veri okuma

Ön Bilgi

Bu deney iki kısımdan oluşmaktadır. İlk kısımda POSIX işletim sistemlerine özgü olan program geçişiyle ilgileneceğiz. İkinci kısımda ise dosya işlemleriyle ilgi örnek yapacağız. Burada göreceğimiz dosyalarla ilgili işlemleri standart C kütüphanesinde bulunan fonksiyonlarla değil de fcntl.h tanımlı sistem çağrıları aracılığıyla yapacağız.

Program geçişi

Program geçişi, çalışmakta olan bir programın, program akışını bir başka programa devretmesidir. Bu esnada yeni bir işlem yaratılmaz, varolan işlem üzerinden çalışma devam edilir. Genelde program geçişi alt işlem açıldıktan sonra alt işlemde gerçekleştirilir, ancak bu bir zorunluluk değildir. Program geçişi başarılı olursa, çağıran program bellekten atılır. Bundan dolayı program geçişini sağlayan sistem çağrısının ardındaki kodlar, eğer çağrı başarılıysa çalıştırılmazlar. Program geçişini sağlayan sistem çağrılarına **exec** sınıfı sistem çağrıları adı verilir. Bunun sebebi tüm bu fonksiyonların isimlerinin "exec"le başlamasıdır. Biz deneyimizde **execl()** ve **execlp()** sistem çağrılarını kullanacağız. Bu çağrılar ilk parametre olarak çalıştırılacak olan programın yolunu alır. **execlp()** yalnızca ana programın bulunduğu dizinde değil, sistemde tanımlı programların bulunduğu dizinlere bakacaktır. **execl()** ile çağıran programın bulunduğu dizindeki programlar çalıştırılabilir. İlk parametreden sonra verilen parametreler, çalıştırılacak olan programa aynen geçirilirler. Burada unutulmaması gereken, çalıştırılan programa her zaman program adının

* BLGM 343 dersi için Güz 2013/2014 döneminde Gürcü Öz ve Cem Kalyoncu tarafından hazırlanmıştır

parametre olarak geçirilmesi gerektiğidir. Programa geçirilecek parametreler yazıldıktan sonra, parametrelerin bittiğini belirtmek için **NULL** değeri de verilmelidir. Aşağıdaki kod parçası, kullanıcıya oluşturmak istediği dizinin adını sorarak, yaratılan alt işlemde verilen isimde bir dizin yaratacaktır. **perror()** sistem çağrısı, en son sistem çağrısından kaynaklanan hatayı ekrana yazırır. mkdir komutu ise yeni bir dizin oluşturmaya yarayan programdır.

```
char dizin[30];
scanf("%s", dizin);

if(fork()) {
     wait();
     printf("Dizin yaratıldı\n");
}
else {
     execlp("mkdir", "mkdir", dizin, NULL);
     printf("Hata oluştu\n");
}
```

Deney 1

Bir uygulama yazarak "**çık**" komutu verilene kadar kullanıcının istediği komutların parametrelerini sorarak çalıştırılmasını sağlayınız. Bu program aşağıdaki komutları desteklemelidir:

1. **dosyalar**: ls komutunu çalıştırmalı, her hangi bir ek soruya gerek yoktur.
2. **yenidizin**: yeni dizinin adını sorduktan sonra, mkdir komutunu çalıştırmalı, yukarıdaki örnekten faydalanabilirsiniz.
3. **merhaba**: ekrana merhaba yazan basit bir program yazarak derleyiniz. Bu seçenek istendiğinde, ana program bu basit programı çalıştırmalı. (Bu işlem için **execl()** sistem çağrısını kullanınız)
4. **çık**: programdan çıkışı sağlamalı

Ana işlem çalıştırılan komutların tamamlanmasını beklemeli ve komutun tamamlandığını ekrana yazmalı.

Dosya işlemleri

Dosya işlemlerini C dilinde bize verilen kütüphane fonksiyonlarıyla yapabileceğimiz gibi, sistem çağrılarıyla da yapabiliriz. Aradaki en önemli fark, sistem çağrıları dosya dışında özel giriş/çıkış meknizmalarına da veri yazabilir, yada okuyabilir. Sistem çağrılarıyla açılmış olan dosya tanımlayıcıları bir işleme özgüdür. Ancak bir işlem alt işlem açtığında, alt işlem ana işleme ait olan dosya tanımlayıcılarını miras alır.

creat() sistem çağrısı yeni bir dosya yaratır yada varolan dosyayı silerek yeni bir dosya olarak

açılmasını sağlar. Bu çağrıya ilk parametre olarak dosyanın adını, ikinci parametre olarak dosyanın erişim haklarını vermek gerekir. **creat()** sistem çağrısı eğer hata varsa -1 değerini, eğer hata yoksa açılan dosya tanımlayıcısını döndürür. Aşağıda yeni.txt dosyasını, sahibinin değiştirip, diğer herkesin okuyabileceği şekilde yaratan bir örnek mevcuttur, dt dosya tanımlayıcısının kısaltmasıdır.

```
int dt=creat("yeni.txt", 0644);
```

Yazılmak için açılmış bir dosya tanımlayıcısına, **write()** sistem çağrısıyla veri yazılabilir. Bu sistem çağrısı ilk parametre olarak dosya tanımlayıcısını, ikinci parametre olarak yazılacak verinin işaretcisini, üçüncü parametre olarak ise yazılacak verinin uzunluğunu alır. Eğer yazma esnasında bir hata oluşursa, **write()** sistem çağrısı -1 değerini döndürür. Aşağıda bir dosyaya bir tam sayı değişkeni ve bir dizgi değişkeninin kaydedilmesi verilmiştir.

```
write(dt, &tamsayi, sizeof(int));
write(dt, dizgi, strlen(dizgi));
```

open() sistem çağrısı varolan bir dosyayı açar. Bu çağrı ilk parametre olarak dosya adını, ikinci parametre olaraksa dosyanın hangi modda açılacağını alır. Dosyadan okumak için **O_RDONLY** modunu kullanmak gerekmektedir. **open()** sistem çağrısı eğer hata varsa -1 değerini, eğer hata yoksa açılan dosya tanımlayıcısını döndürür. Okunmak için açılmış bir dosyadan veriler **read()** sistem çağrısıyla okunabilir. Bu sistem çağrısı ilk parametre olarak okunacak dosyanın tanımlayıcısını, ikinci parametre olarak okunacak verinin yerleştireceği değişkeni, son olarak da, okunması gereken uzunluğu alır. Hata çıkması durumunda **read()** sistem çağrısı -1 değerini döndürür, eğer hata yoksa, okunan veri uzunluğunu verir. Dosyanın sonuna ulaşıldıysa, okunan veri miktarı istenen veri miktarından az olabilir. Bu bir hata durumu oluşturmaz. Aşağıda bir dosyadan bir tam sayı değeri ile 10 karakter uzunluğunda bir verinin okunma örneği verilmiştir.

```
char veri[11];
int uzunluk;
int dt=open("dosya.txt", O_RDONLY);
read(dt, &tamsayi, sizeof(int));
uzunluk=read(dt, veri, 10);
veri[uzunluk]=0;
```

Son kısımda, okunan verinin sonuna boş karakter konarak, dizgi fonksiyonlarıyla kullanılabilmesi sağlanmaktadır. Yeni açılan bir işlemde, 0 numaralı dosya tanımlayıcısı standart girdi, 1 numaralı dosya tanımlayıcısı ise standart çıktıyı temsil eder. Bu dosya tanımlayıcılarını kullanarak, ekrana veri yazabilir, yada kullanıcıdan veri okunabilir. Aşağıda bir dosyayı sonuna kadar 100 bytelık bloklar halinde okuyup ekrana yazan kod verilmiştir.

```
char veri[100];
int uzunluk;
...
while( (uzunluk=read(dt, veri, 100) > 0) {
     write(1, veri, uzunluk);
}
```

Deney 2

Bu deneyde, sizden aynı anda birden fazla dosyayı kopyalayabilen bir sistem yapmanız istenmektedir. Bu sistemde, ana işlem aracılığıyla kullanıcıdan iki dosya okuyunuz. İlk dosya kaynak dosyası, ikinci dosya ise hedef dosyasıdır. Her kopyalama işlemi için ana işlem yeni alt işlem açmalıdır. Oluşturulan bu alt işlem, verilen kaynak ve hedef dosyasını açıp, kaynaktan okunan verileri hedef dosyasına yazdırmalıdır. Bu işlemi yaparken, alt işlem, her adımda 100 byte okumalı, ve her yazma işleminden sonra 1 saniye beklemelidir. Böylece, kopyalama işlemi süre alacağından programınızı test etme şansınız olacaktır. Kopyalama işlemini bitiren alt işlem ekrana kopyalama işleminin tamamlandığını yazmalıdır. Eğer ilk dosya ismi olarak çık yazıldıysa, ana işlem programı sonlandırmalıdır, yoksa, tekrar iki dosya ismi sorarak çalışmaya devam etmelidir. Ana program çıkmadan önce, alt işlemler bekleniyor yazdıktan sonra, tüm alt işlemleri beklemeli, ve ondan sonra çıkmalıdır.