CS360 Lecture notes -- Error Checking

- Jian Huang
- CS360
- Directory: **~huangj/cs360/notes**
- Lecture notes: **http://www.cs.utk.edu/~huangj/cs360/360/notes/errorchecking.html**

Error Checking

Sistem programlama ve uygulama programlama arasındaki önemli farklardan biri iyi bir hata denetimine sahip olmamasıdır.**but so essential that one cannot live without**
Çok yaygın olarak insanların consola hata mesajlarını çıktısını "printf " ile yaptıklarını görebilirsiniz. **Although better than not having any error checking, however, this is not enough even when not doing systems programming**. Her program çalışır bir işlem olduğunda varsayılan olarak, stdin, stdout ve stderr tarafından açılan üç dosya vardır. İlk ikisi giriş ve çıkış konsolları *stder, hata mesajlarında gidilen yerdir. Stder çalışmadığında ,Stdin* ve *stdout a* yönlendirilebilirsiniz. Bu nedenle hata mesajlarını kullanarak en iyi çıkış:

```
fprintf(stderr,"your error message");
```

Ayrıca *printf* ve *fprintf(stdout,"...")* aynı görevi yaptığını bilmeniz gerekir.fprintf in yanısıra hata kontrolünde kullanmak için diğer iki önemli arç şunlardır: **perror** ve **assert,** ANSI C standardı ve ANSI C'1987 desteklendiği tüm işletim sistemlerinde mevcuttur.

Perror

Bilgisayar ilk açıldığında yürütülen ilk programa *" İşletim Sistemi*" denir. İşletim sistemi bilgisayardaki birçok faaliyeti kontrol eder. Bu, disklerin nasıl kullanıldığını, hafızanın nasıl kullanıldığını, CPU'in nasıl kullanıldığını, kimin içeride günlüğe ne kaydettiği, ve diğer bilgisayarlarla nasıl konuştuğunu kapsar. Biz " *Unix*" işletim sistemini kullanırız ve onu çağırırız. Programların işletim sistemiyle konuştuğu yol " *sistem çağrıları*" yoluyladır. Bir sistem çağrısı, bir prosedür çağrısı gibi görünür (aşağıda bakınız), ama farklıdır**-- bazı faaliyeti yapması için işletim sistemine bir istektir**. Sistem çağrıları pahalıdır. Bir prosedür çağrısı genellikle, birkaç makine talimatında yapılabilirken, bir sistem çağrısı, bilgisayarda onun durumunu kaydetmesi için, CPU'nun kontrolü işletim sistemine verilir, işletim sisteminin, bazı fonksiyonları gerçekleştirme, işletim sisteminin kendi durumunu kaydetme ve CPU' nun bize geri dönmesi için kontrolü sağlar.Bu kavram önemlidir ve sınıfta tekrar tekrar görülecektir.

Genellikle bir hata bir sistem veya kütüphane çağrısı olduğunda, özel bir dönüş değeri geri gelir ve global bir değişken "hata numarası(errno)" hata olarak ne söylemesi gerektiği ayarlanır. Örneğin, var olmayan bir dosyayı açmaya çalıştığınızı varsayalım.

```
#include < stdio.h >
```

```
#include < errno.h >

main()
{
  int i;
  FILE *f;

  f = fopen("~huangj/nonexist", "r");
  if (f == NULL) {
    printf("f = null.  errno = %d\n", errno);
    perror("f1");
  }
}
```

ch1a.c ~huangj/nonexist okuma için dosyayı açmaya çalışır. Bu, dosya yok gibi bir ifade çıkar. Böylece, **fopen NULL** (**fopen** için man sayfasına) döndürür ve hata bayrağı hata numarası (**errno**) ayarlar. Programı çalıştırdığınızda, hata numarası (errno) 2 olarak ayarlandığını göreceksiniz. Bunu görmek için, iki şeyden birini yapabilirsiniz:

• 1. **/ Usr / include / errno.h errno** değeri (UNIX makinelerde / usr / include / sys / errono.h uzantısına bakmak zorunda kalacaktır*."#include<sys / errno.h> içerir".*). Bu sonucu görürsünüz:

```
#define ENOENT          2               /* No such file or directory */
```

• 2.Prosedürü **"perror ()"** kullanın - yine man sayfası okuyun. Buna errno aracı yazdırın.Bu nedenle, f1'in çıktısı;

```
 f = null. errno = 2

 f1: No such file or directory
```

Bunlar standart ara yüz errorudur. **perror** kullanmayı öğrenin.

---

assert

Çoğu zaman, kod yazarken varsayımlar yapmaya ihtiyaç vardır. Varsayım yaptığımız zaman hata yaptığımızda yanlışı bulmamızın en iyi yolu *assert* kullanmaktır. *assert*'in en tipik kullanımı, bir programı geliştirme süresince program errorlarını belirlemektir. Assert'e verilen argüman seçilmiş olmalı böylece yalnızca program istenildiği gibi çalıştığında doğruyu tutacaktır.

Eğer argüman deyimi false(0) ise – *assert* argümanını ölçme argümanı- program uygulama durur ve kullanıcı uyarılır. Eğer argüman true ise uyarı gelmez.

Bir onaylama işlemi başarısız olunca aşağıdaki şekilde bir hata mesajı oluşturulur.

```
assertion failed in file name in line num
```

kaynak dosyanın ismi buradadır ve num adı başarısız satır numarasıdır. Programınız boyunca savunmaların kullanımının gelişmesi sırasında hatalar yakalayabilirsiniz. Yaptığınız herhangi bir varsayım için iddialar yazmak gerektiği pratik bir kuraldır. Örneğin, bir argümanın NULL olmadığını varsayalım, bu durumu kontrol etmek için bir iddia deyimi kullanın.

```
void checkerror_strcpy(char * src, char *dst)
{
assert(src!=dst);
assert(src!=NULL);
assert(dst!=NULL);
}
```

Burada strcpy yaptığımız bazı varsayımların doğru olup olmadığını kontrol eder. Yazılımda hata olmadığından emin olduktan sonra, biz kolayca tüm iddiaları devre dışı bırakabiliriz "#define NDEBUG" denetimini eklemeden önce burada *"#include<assert.h>"* kaynak kodu görünür.

Özet

Diğer insanlara yararlı olabilir kod yazmak için, hata denetimi önemli küçük bir adımdır. Gelecekte kariyeriniz için ve CS360 için ödevlerinizi yaptığınızda zamandan tasarruf gibi bir geri dönüş sağlar, şiddetle yazdığınız tüm kodlara bolca hata denetimi eklemenizi öneririm.