Bu bölüm stat, Istat ve opendir çağrılar ailesi ile yapabileceklerimizi içeriyor

STAT

Stat için ana sayfayı okuyun(yada kitabın 4. bölümünü).Stat,dosyalar hakkında bilgi(dosya düğümünde bulunan bilgiler) alabileceğimiz bir komuttur.

Motive edici basit bir örnek üzerinden gidecek olursak. Stat sistem çağrılarına sahip olmadığınızı varsayalım ve siz her argümanı bir dosya olan ve bu

dosyanın boyutunu ve adını listeleyen bir program yazmak istiyorsunuz.Bu da aşağıdaki gibi bir şey olur:

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

main(int argc, char **argv)
{
  int i;
  int fd;
  off_t size;

  for (i = 1; i < argc; i++) {
    fd = open(argv[i], O_RDONLY);
    if (fd < 0) {
      printf("Couldn't open %s\n", argv[i]);
    } else {
```

```
    size = lseek(fd, (off_t) 0, SEEK_END);
    printf("%10ld %s\n", size, argv[i]);
    close(fd);
   }
  }
}
```

Burda herbir dosyayı açıyor,dosyanın sonuna gidiyor ve dosyanın boyutunu hesaplıyor.

Bu çalışma tamam, şimdi bunu kopyala ve sonra derle aşağıdakinde:

```
UNIX> ls1 /home/plank/cs360/notes/Stat/m*
```

Aşağıdakinin karşınıza çıkacagını göreceksiniz:

```
584 /home/plank/cs360/notes/Stat/makefile
Couldn't open /home/plank/cs360/notes/Stat/myfile
```

Henüz, normal(sıradan) ls-l yi denediğinizde, karşılaşacağınız şey:

```
UNIX> ls -l /home/plank/cs360/notes/Stat/m*
-rw-r--r-- 1 plank guest 584 2010-02-01 18:45 /home/plank/cs360/notes/Stat/makefile
---------- 1 plank guest   3 1996-09-17 15:05 /home/plank/cs360/notes/Stat/myfile
UNIX>
```

Eğer Isl "myfile"ı açamasaydı, boyutu yazdıramacaktı. Bu da kötü olurdu, işte bu noktada "stat" fonksiyonuna neden ihtiyacımız olduğunu görüyoruz. Bu dosya hakkında bilinmesi iyi olacak şeyler var, bu dosyanın erişimine iznimiz olmamasına rağmen.

Özellikle, stat fonksiyonu bir dosyanın düğümü hakkında bilgi verir.Stat fonksiyonu, kullanıcının, dosyanın içinde bulunduğu yolun iznini alıncaya kadar bunu yapabilir.

Stat yapısı kabaca aşağıdaki gibi tanımlanır:

```
struct stat {
        mode_t   st_mode;    /* File mode (see mknod(2)) */
        ino_t    st_ino;     /* Inode number */
        dev_t    st_dev;     /* ID of device containing */
                             /* a directory entry for this file */
        dev_t    st_rdev;    /* ID of device */
                             /* This entry is defined only for */
                             /* char special or block special files */
        nlink_t  st_nlink;   /* Number of links */
        uid_t    st_uid;     /* User ID of the file's owner */
        gid_t    st_gid;     /* Group ID of the file's group */
        off_t    st_size;    /* File size in bytes */
        time_t   st_atime;   /* Time of last access */
        time_t   st_mtime;   /* Time of last data modification */
```

```
        time_t   st_ctime;    /* Time of last file status change */
                        /* Times measured in seconds since */
        long     st_blksize;  /* Preferred I/O block size */
        long     st_blocks;   /* Number of 512 byte blocks allocated*/
};
```

Kafa karıştırıcı tipler genellikle int, long ve short tiplerdir.

```
        typedef unsigned long   ino_t;
        typedef short          dev_t;
        typedef long           off_t;
        typedef unsigned short  uid_t;
        typedef unsigned short  gid_t;

        typedef unsigned short  mode_t;        /* file mode bits */
        typedef short          nlink_t;       /* links to a file */
        typedef long           time_t;        /* value = secs since epoch */
```

İlk önce ana sayfayı okumalısın, open/lseek yerine kullanılan stat ın doğru bir şekılde çalışabilmesi için ls1.c nın değişimi gereksiz olacaktır.

```
                #include <stdio.h>
                #include <sys/types.h>
                #include <sys/stat.h>
```

```
main(int argc, char **argv)
{
 int i;
 struct stat buf;
 int exists;

 for (i = 1; i < argc; i++) {
 exists = stat(argv[i], &buf);
 if (exists < 0) {
 fprintf(stderr, "%s not found\n", argv[i]);
  } else {
 printf("%10ld %s\n", buf.st_size, argv[i]);
  }
 }
}
```

```
UNIX> ls2 /home/plank/cs360/notes/Stat/m*
      584 /home/plank/cs360/notes/Stat/makefile
        3 /home/plank/cs360/notes/Stat/myfile
UNIX>
```

Daha sonra, biz de çalışan, gerçek "ls" gibi bir "ls" ye sahip olmak

isteriz--argüman kabul etmeyen ve geçerli dizindeki dosyaları listeleyen--.

Bunu yapmak için "opendir/readdir/writedir" çağrılarına ihtiyacımız var.

Not: bunlar C kütüphanesi çağrılarıdır, sistem çağrıları değil.

Bunun anlamı,bizim için "open/close/read/write" çagrılarını yapabılır ve

dizin dosyalarını format şekline dönüştürebilir.Bu kullanışlıdır. "struct dirent

yapısı "/usr/include/sys/dirent.h" de tanımlıdır:

```
struct  dirent {
off_t          d_off;         /* offset of next disk dir entry */
unsigned long   d_fileno;      /* file number of entry */
unsigned short  d_reclen;      /* length of this record */
char           *d_name;       /* name */
};
```

Ls3.c, ls2.c yi geçerli dizinden okur(".") ve tüm dosyaları ve boyutlarını yazdırır:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <dirent.h>
```

```
main(int argc, char **argv)
{
  struct stat buf;
  int exists;
  DIR *d;
  struct dirent *de;

  d = opendir(".");
  if (d == NULL) {
    fprintf(stderr, "Couldn't open \".\"\n");
    exit(1);
  }

  for (de = readdir(d); de != NULL; de = readdir(d)) {
    exists = stat(de->d_name, &buf);
    if (exists < 0) {
      fprintf(stderr, "%s not found\n", de->d_name);
    } else {
      printf("%s %ld\n", de->d_name, buf.st_size);
          }

 }
  closedir(d);
}

UNIX> ls3
lsc2 32700
```

ls5 23731

ls2.c 625

lsc2.c 860

ls5a.o 10680

ls4.o 10024

ls1.c 628

ls6 23724

ls6.c 1303

ls5a 23833

...

Şimdi,iki şeye dikkat edeceksiniz ls3 ü çalıştırırken-ilk olarak, çıktı biçimlendirilemedi. ikincisi, dosyalar sırlanamadı. Çünkü readdir() bir dizindeki dosyaların sıralanması hakkında hiçbir garanti vermez.Bundan sonraki iki program bu problemleri çözecektir. Önce çıktı biçimlendirilememe problemi.Ne görmek istiyoruz gibi bir şey:

lsc2 32700

ls5 23731

ls2.c 625

lsc2.c 860

ls5a.o 10680

ls4.o 10024

ls1.c 628

ls6 23724

ls6.c 1303

ls5a 23833

...

Bunu yapmak için,herhangi bir dosya ismini yazdırmadan önce en büyük dosya boyutuna sahip dosyanın boyutunu

bilememiz gerekiyor.Peki, ne yaparsak tüm dizinin bir linked listen okunmasını ve tek bir yoldan max. boyutun

hesaplanmasını sağlarız.Bu işlemin ardından, listeyi çaprazlarız ve çıkışı güzel bir biçimde yazdırırız.

"printf" komutuna yakından bakın ve "printf"in neden çalıstığını göreceksiniz. ls4.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include "dllist.h"

main(int argc, char **argv)
{
  struct stat buf;
  int exists;
  DIR *d;
  struct dirent *de;
  Dllist files, tmp;
  int maxlen;
```

```
 d = opendir(".");
 if (d == NULL) {
  fprintf(stderr, "Couldn't open \".\"\n");
  exit(1);
 }

 maxlen = 0;
 files = new_dllist();

 for (de = readdir(d); de != NULL; de = readdir(d)) {
  dll_append(files, new_jval_s(strdup(de->d_name)));
  if (strlen(de->d_name) > maxlen) maxlen = strlen(de->d_name);
 }
 closedir(d);

 dll_traverse(tmp, files) {
  exists = stat(tmp->val.s, &buf);
  if (exists < 0) {
   fprintf(stderr, "%s not found\n", tmp->val.s);
  } else {
   printf("%*s %10ld\n", -maxlen, tmp->val.s, buf.st_size);
  }
 }
}
```

Neden strdup ı dll_append() çagrısında kullanırız de->d_name in yerine? Cevabı ise zekice. Ana sayfa bize

readdir() yapısının(struct) nasıl dödürülüp tahsil edildiği hakkında hiçbirşey söylemiyor. Tum bunları bir sonraki çağrıya

kadar(readdir()veya closedir()) tahmin edebiliriz. Eğer readdir() mallocs alanını "struct dirent" için döndüreceklerini

ve kullanıcı free() fonk. çağırana kadar space in serbest olmadıgını bilseydik , biz de okunaklı bir şekilde >d_name ı dllist

mize koyardık.Herneyse, ana sayfada bu tür bir güvence olmadığından "strdup" ı çağırmalıyız.Örnek olarak "opendir/readdir/closedir" aşağıdaki gibi

uygulanmalı:

* opendir() dizin dosyasını açar ve mallocs one "struct dirent".

* readdir () ise "struct dirent" haline gelecek dizin girdisini okur ve bir gösterici ile döner.

* closedir() dizin dosyasını kapatır ve "struct dirent" serbest kalır.

Biz neden mantıklı bir sebepten dolayı dll_append() deyiminde strdup() ı çağıracağımız bir uygulama gerektiriyor. Eğer sadece "de->d_name" i oraya koyarsak

hafıza ile ilgili tüm sorunları almış olacağız.burası önemli.Anladıgınızdan emin olun.

Şimdi, Sıralanmış dizin dosyalarını yazdırmak için,sadece dllist yerine bir red-black yerleştirmeye ihtiyacımız var? ls5.c. de bunu kolayca değiştirebiliriz.

Sırada, printf deyimindeki %10d den kurtulmak var. Başka bir deyişle dosya adlarının başında ve sonunda boşluklar olmasını istiyorum. Biz bunu dizin geçişinde max. boyuta

sahip dosyanın boyutunu bularak yapacagız ve printf deyimini kullacağız.sprintf and a strlen -- see ls5a.c.

---

ls6.c, "ls -F" ile aynı performansa sahiptir.Yani, bir sembolik sonunda "/", (yumuşak) bir "@" ile bağlantılar ve bir "*" ile çalıştırılabilir dosyalar ile dizin yazdırır.

Biz "struct tampon" nin "st_mode" alanına yorumlayarak bunu edebiliyoruz.Kod üzerinde bak.