CS360 Ders notları - Cat ve türevleri.Tampon.

- Jim Plank
- Dizin: / **blugreen/homes/plank/cs360/notes/Cat**
- Ders notları: **http://www.cs.utk.edu/ ~ plank/plank/classes/cs360/360/notes/Cat/lecture.html**

Bu ders unix sistem çağrıları ve C standart I / O kütüphanesi ile "cat" yazma ile ilgili daha fazla ayrıntı sunar. Ayrıca performans için tamponlanmayı motive eder.

Simpcat

Burada sadece standart girdiden okur ve standart çıktıya yazar basit bir **cat** programı, yazma üç eşdeğer yolu vardır.

simpcat1.c

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

main()
{
  char c;

  c = getchar();
  while(c != EOF) {
    putchar(c);
    c = getchar();
  }
}
```

simpcat2.c

```
main()
{
  char c;
  int i;

  i = read(0, &c, 1);
  while(i > 0) {
    write(1, &c, 1);
    i = read(0, &c, 1);
  }
}
```

simpcat3.c

```
#include <stdio.h>

main()
{
  char c[1];
  int i;

  i = fread(c, 1, 1, stdin);
  while(i > 0) {
    fwrite(c, 1, 1, stdout);
    i = fread(c, 1, 1, stdin);
  }
}
```

Hadi bunlara biraz daha yakından bakalım. ***.c** ve **makefile** sizin bir dizinize kopyalayın ve **"make"** yazın. Şimdi aşağıdakileri yapın:

```
UNIX> sh
sh-2.05b$ time simpcat1 < large > /dev/null

real    0m0.151s
user    0m0.137s
sys     0m0.012s
sh-2.05b$ time simpcat2 < large > /dev/null

real    0m34.675s
user    0m10.037s
sys     0m24.594s
sh-2.05b$ time simpcat3 < large > /dev/null

real    0m0.971s
user    0m0.543s
sys     0m0.014s
sh-2.05b$ exit
UNIX>
```

Kullanmakta olduğunuz makineye bağlı olarak, muhtemelen yukarıdaki zamanlardan farklı sonuç elde edilebilir – onlar benim 2.16 GHz MacBook Pro 2010'du. Ne olursa olsun elde ettiğiniz numaralar, **simpcat1, simpcat2** ve **simpcat3** arasındaki oranlar kabaca aynı olmalıdır.

Peki, ne oluyor? **/dev/null** sizin yazabileceğiniz özel bir Unix dosyasıdır, ama hiçbir diskte hiçbir zaman saklanmaz. Biz onu kullanıyoruz , bundan dolayı 7.5M dosyayı home dizinine açarak dizininde atık alanı oluşturma. "**Büyük**" bir 7.500.000 baytlık dosyadır. Bu **simpcat1.c** içinde, **getchar()** ve **putchar()**, **simpcat2.c**'de de **read()** ve **write()** ve **fread()** ve **fwrite()** **simpcat3**'te 7.500.00 defa çağrıldığı anlamanı gelir. Açıkçası, **simpcat2.c**'nin suçu gerçekte programın kütüphane çağrıları yerine sistem çağrıları yapmasıdır. Unutmayın ki bir sistem çağrısı işletim sistemine yapılmış bir istektir. Bu her bir read/write çağrısı, işletim sistemi CPU'yu yönetimine alır(Bu simcat2 programının kaydedilmesi durumuna gelir), işlem isteği ve geri dönüş(simcat2 programın durumuna geri yüklemesi anlamına gelir) anlamına gelir. Bu açıkçası **simpcat1.c** ve **simpcat3.c**'nın yaptıklarına göre çok daha pahalıdır. Şimdi, **simpcat4.c** ve **simpcat5.c** bakın:

simpcat4.c

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

main(int argc, char **argv)
{
  int bufsize;
  char *c;
  int i;

  bufsize = atoi(argv[1]);
  c = (char *) malloc(bufsize*sizeof(char));
  i = 1;
  while (i > 0) {
    i = read(0, c, bufsize);
    if (i > 0) write(1, c, i);
  }
}
```

simpcat5.c

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

main(int argc, char **argv)
{
  int bufsize;
  char *c;
  int i;

  bufsize = atoi(argv[1]);
  c = (char *) malloc(bufsize*sizeof(char));
  i = 1;
  while (i > 0) {
    i = fread(c, 1, bufsize, stdin);
    if (i > 0) fwrite(c, 1, i, stdout);
  }
}
```

Bunları bir kez daha bayt zamanlı okuyalım. Buna *buffering (tamponlama)* denir. Şeyleri saklamak için bir bölge tahsis edersiniz, böylece daha az sistem/prosedür çağrısı yaparsınız. Bunu not edin **fread()** ve **fwrite()** aynı **read()** **write()** gibidir, beklenenin dışında, onlar işletim sisteminin yerine standart I/O kütüphanesine giderler . Aşağıda farklı *bufsize değerleri için* "**büyük**" dosyasının **simpcat4** ve **simpcat5** çalıştırmasının ne kadar hızlı olduğunu gösterir(bu eski sayılar 1990'dan -- bugün ki sayılar daha hızlı olacak ama bağıl performansı aynı olacak).

simpcat4.c (the first column is bufsize):

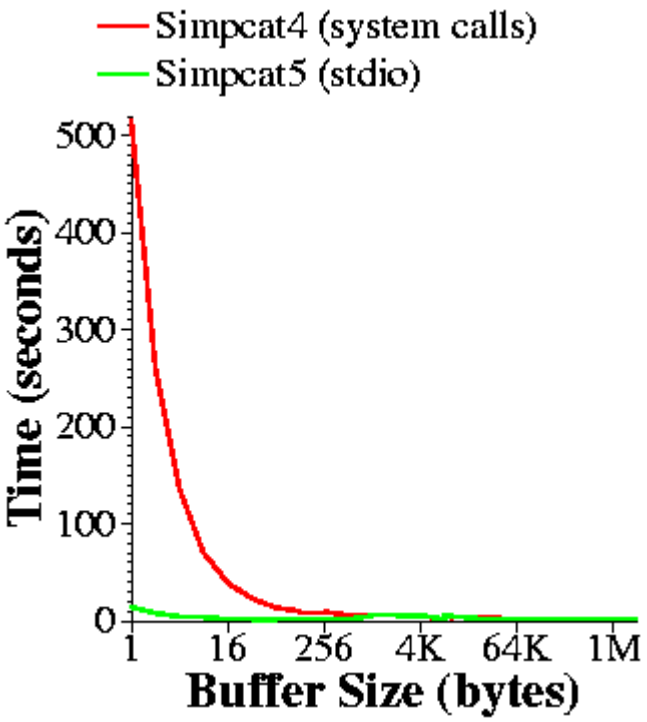
1	516.1 real	25.4 user	481.0 sys
2	261.1 real	12.3 user	241.9 sys
4	135.9 real	6.2 user	123.8 sys
8	70.1 real	3.3 user	61.5 sys
16	39.2 real	1.5 user	32.5 sys
32	22.9 real	0.8 user	16.6 sys
64	13.6 real	0.5 user	8.5 sys
128	10.0 real	0.2 user	4.7 sys
129	8.6 real	0.1 user	4.9 sys
256	7.8 real	0.1 user	2.8 sys
267	9.3 real	0.1 user	3.0 sys
512	5.6 real	0.0 user	1.9 sys
513	5.6 real	0.0 user	2.0 sys
1024	5.5 real	0.0 user	1.3 sys
1025	5.5 real	0.0 user	1.5 sys
2048	5.8 real	0.0 user	0.9 sys
2049	5.6 real	0.0 user	1.3 sys
4096	5.2 real	0.0 user	0.9 sys
4097	5.4 real	0.0 user	1.2 sys
8192	1.5 real	0.0 user	0.6 sys
8193	5.4 real	0.0 user	1.1 sys
10000	3.2 real	0.0 user	0.7 sys
20000	4.1 real	0.0 user	0.7 sys
50000	2.1 real	0.0 user	0.6 sys
100000	2.0 real	0.0 user	0.6 sys
200000	1.9 real	0.0 user	0.6 sys
500000	1.9 real	0.0 user	0.6 sys
1000000	1.9 real	0.0 user	0.7 sys
2000000	2.0 real	0.0 user	0.8 sys

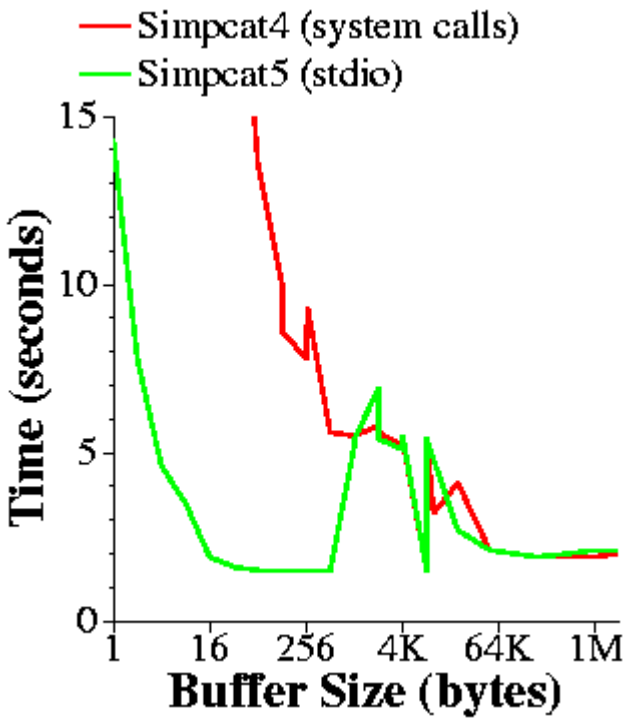
simpcat5:

1	14.3 real	13.2 user	0.8 sys
2	7.8 real	6.9 user	0.7 sys
4	4.6 real	3.8 user	0.7 sys
8	3.5 real	2.3 user	0.7 sys
16	1.9 real	1.0 user	0.6 sys
32	1.6 real	0.6 user	0.6 sys
64	1.5 real	0.5 user	0.5 sys
128	1.5 real	0.3 user	0.6 sys
129	1.5 real	0.4 user	0.7 sys
256	1.5 real	0.2 user	0.7 sys
267	1.5 real	0.3 user	0.6 sys
512	1.5 real	0.2 user	0.6 sys
513	1.5 real	0.3 user	0.7 sys
1024	5.2 real	0.2 user	1.1 sys
1025	5.4 real	0.3 user	1.1 sys
2048	6.9 real	0.1 user	0.9 sys
2049	5.4 real	0.1 user	1.1 sys
4096	5.1 real	0.1 user	1.0 sys
4097	5.5 real	0.1 user	1.0 sys
8192	1.5 real	0.1 user	0.6 sys
8193	5.4 real	0.2 user	1.0 sys
10000	4.8 real	0.1 user	0.7 sys
20000	2.7 real	0.1 user	0.8 sys
50000	2.1 real	0.2 user	0.6 sys
100000	2.0 real	0.1 user	0.6 sys
200000	1.9 real	0.1 user	0.6 sys
500000	2.0 real	0.1 user	0.6 sys
1000000	2.1 real	0.1 user	0.7 sys
2000000	2.1 real	0.1 user	0.7 sys

Buffer(arabellek) boyutunu artırdığın zaman, kullanıcı ve sistem zamanının ikisinin de nasıl büyük ölçüde azaldığını not ediniz. Bu **simpcat4.c'**da daha az doğrudan bir sistem çağrısı yapma sonucudur. **Simpcat5.c'**de daha az yordam çağrıları yapmanın bir sonucudur. Bir kere arabellek yeterince büyük olursa, iki program aşağı yukarı aynı davranışı sergiler.

İki programın grafikleri de burada. Her ikisi grafikte aynı veri grafiğini çizer -- en sağdaki grafik basitçe en soldaki grafiğin altında ki bölgeyi büyütür. Ayrıca, x – ekseni bir günlük ölçekte olduğunu not edin.





Açıkçası, verilerin arasında bir grup gürültü var, ama bazı sonuçların çizimi için onu kullanabiliriz.

İlk olarak, standart I/O kütüphanesi hakkında şimdi ne sonuç çıkartabiliriz ? Tamponlama kullanır! Başka bir deyişle, ilk **getchar()** veya **fread()** çağırdığın zaman, tamponun içine bir read() baytlarının geniş sayılarını gerçekleştirir. Böylece, sonraki **getchar()** ve **fread()** çağrıları hızlı olacaktır. **fread()** hafızanın geniş bölümlerine girişimde bulunduğu zaman, **fread()** tampona ihtiyacı yoktur gibi iki aynı davranışı sergiler – bunu alt program için yapıyorsun.
Öyleyse **getchar() , fread(c, 1, 1, stdin)** neden daha hızlı? Çünkü **getchar()** bir karakter okumak için optimize edilmiştir ve **fread()** ise edilmemiştir.

Bundan sonra ki ders nedir?

1. Tamponlama çok fazla sistem çağrılarını azaltmak için iyi bir yöntemdir.
2. Eğer küçük bayt yığını okuyorsan, o zaman **getcahr()** ve **fread()** kullan. Onlar sizin için tamponlamayı yapar.
3. Eğer tek bir karakter I/O karakter yapıyorsanız, **getchar()** kullan (veya **fgetc()**).
4. Eğer büyük bayt yığını okuyorsan, o zaman **fread()** ve **read()** yaklaşık aynı çalışır. Ancak, **fread()** kullanmalısınız, sizin programlamanızı daha tutarlı yapar, ve çünkü

sizin birazcık daha fazla hata kontrolü yapar.

Sınıfta detaylı olarak onları incelemedik bile, yazar içinde aynısı geçerlidir.

Standart I/O vs Sistem çağrıları

Her sistem çağrısı standart I/O kütüphanesinden benzer yordam çağrıları vardır.

```
Sistem Çağrıları              Standart I/O Çağrıalrı
-----------                   -----------------
open                          fopen
close                         fclose
read/write                    getchar/putchar
                              getc/putc
                              fgetc/fputc
                              fread/fwrite
                              gets/puts
                              fgets/fputs
                              scanf/printf
                              fscanf/fprintf
lseek                         fseek
```

Sistem çağrıları tamsayı dosya tanımlayıcılarıyla çalışır. Standart I/O çağrıları **FILE** adlı bir yapı(struct) tanımlar, ve bu yapılar için işaretçilerle çalışır.
Örnek vermek gerekirse, aşağıdaki cat programı versiyonları, **filename** onların argümanı olarak çağırmalı. **Cat1.c** sistem çağrılarını kullanır ve **cat2.c** standart I / O kütüphanesini kullanır. **man** sayfasını **open**("man 2v open") ve **fopen**("man 3s fopen") onların argümanlarını anlamak için oku.

Deneyin:

```
UNIX> sh
$ time cat1 large > /dev/null
        0.9 real         0.0 user         0.3 sys
$ time cat2 large > /dev/null
        1.2 real         0.1 user         0.4 sys
$ exit
UNIX>
```

Bunları ilk sayılarla nasıl karşılaştırırsınız?
Son olarak, **fullcat.c** gerçek versiyonu gibi çalışan **cat** bir sürümü içerir -- eğer dosya adı atarsanız, o zaman standart girdiden standart çıktıyı yazdırır. Aksi takdirde, komut satırı argümanlarını belirtilen her dosya yazdırır. **simpcat1.c** ve **cat2.c** her ikisinin nasıl benzediğini unutmayın.
Disk alanı kazanmak ve herhangi bir geçişi dosyayı kaldıracağın zaman '**make clean**' yazın. Bu derste oluşturulan tüm dosyaları silebilirsiniz, benim dizinimden de yeniden kopyalabilirsiniz.

Karakterler vs Tamsayılar

getchar()'ın geri dönüş değeri bir karakter değil ve bir tamsayı olarak tanımlanmıştır. Buna bağlı olarak, **simpcat1a.c** bakın:

```
#include < stdio.h >

main()
{
  int c;

  c = getchar();
  while(c != EOF) {
    putchar(c);
    c = getchar();
  }
}
```

Simpcat1a.c ve **simpcat1.c** arasındaki tek fark **c**'de bir **char** yerine **int**'dir. Şimdi, neden bu önemli mi? Aşağıdakine bakın:

```
UNIX> ls -l simpcat1.c simpcat1
-rwxr-xr-x   1 plank       10864 Sep  8 14:03 simpcat1
-rw-r--r--   1 plank         526 Sep 13  1996 simpcat1.c
UNIX> simpcat1 < simpcat1 > tmp1
UNIX> simpcat1 < simpcat1.c > tmp2
UNIX> ls -l tmp1 tmp2
-rw-r--r--   1 plank        1746 Sep  8 14:10 tmp1
-rw-r--r--   1 plank         526 Sep  8 14:10 tmp2
UNIX>
```

Garip bir şey fark ettiniz mi? Şimdi:

```
UNIX> simpcat1a < simpcat1 > tmp3
UNIX> ls -l tmp3
-rw-r--r--   1 plank       10864 Sep  8 14:12 tmp3
UNIX>
```

getchar() 255 karakter okunduğundav,bunun ne ilgisi var. Sınıfta bunun hakkında konuşacağız.

Burada getchar hatası için basitleştirilmiş bir kod var(değil çünkü basitleştirilmiş). Lütfen nerde olduğunu anlamaya çalışın.

```
int getchar (void)
{
    static char buf[BUFSIZE];
    static char *bufp;
    static int  n = 0;

    if (n == 0) {
       n = read(0, buf, BUFSIZE);
       bufp = buf;
    }

    if (n > 0) {
        n -= 1;
        return *bufp++;
    } else
        return EOF;
}
```