Jerry Jason Herrmann
1422017

Betriebssysteme, Übungsblatt 1

Aufgabe 1: System-Call

Der Ablauf des Aufrufs eines System-Calls anhand des Beispiels `__libc_read()` aus der glibc-Bibliothek, ein System-Call-Wrapper für den `read` System-Call:

Der Prozess beginnt mit dem Aufruf der Funktion `__libc_read()` (Z.b. durch eine beliebige Applikation, die die glibc-Bibliothek verwendet). Für den Aufruf dieser Funktion werden zwei Header-Einschlüsse benötigt. Der erste, `unistd.h`, ist ein Standard-Header, der verschieden Funktionen, Konstanten und Makros des POSIX.1-Standards enthält. Der zweite Einschluss sysdep-cancel.h bezieht sich auf die interne Verarbeitung für Thread-Terminierungspunkte.

```
1   #include unistd.h
2   #include sysdep-cancel.h
3
4    Read NBYTES into BUF from FD.  Return the number read or -1.
5   ssize_t
6   __libc_read (int fd, void buf, size_t nbytes)
7   {
8     return SYSCALL_CANCEL (read, fd, buf, nbytes);
9   }
10  libc_hidden_def (__libc_read)
11
12  libc_hidden_def (__read)
13  weak_alias (__libc_read, __read)
14  libc_hidden_def (read)
15  weak_alias (__libc_read, read)
```

Die eigentliche Funktion besteht lediglich aus einem Aufruf der Funktion SYSCALL_CANCEL. Anders als ein direktes System-Call, werden hier intern in der glibc-Bibliothek die Parameter erst intern überprüft und bei Fehlern eine -1 ausgeben. Sollten keine Fehler auftreten, werden die Parameter an die eigentlich Syscall-Funktion übergeben. Das ausführen dieser Funktion wechselt nun in den Kernel-Modus. Innerhalb dieses Modus wird nun anhand der mitgegebenen System-Call-Zahl die korrespondierende Handlerfunktion lokalisiert, in diesem Fall ksys_read() innerhalb des Kernels.

Jerry Jason Herrmann
1422017

```c
ssize_t ksys_read(unsigned int fd, char __user *buf, size_t count)
{
    CLASS(fd_pos, f)(fd);
    ssize_t ret = -EBADF;

    if (!fd_empty(f)) {
        loff_t pos, *ppos = file_ppos(fd_file(f));
        if (ppos) {
            pos = *ppos;
            ppos = &pos;
        }
        ret = vfs_read(fd_file(f), buf, count, ppos);
        if (ret >= 0 && ppos)
            fd_file(f)->f_pos = pos;
    }
    return ret;
}

SYSCALL_DEFINE3(read, unsigned int, fd, char __user *, buf, size_t, count)
{
    return ksys_read(fd, buf, count);
}
```

Das eigentlich Ausführen der Operation erfolgt anschließend innerhalb des Virtual File Systems (anhand der vfs_read()-Funktion). Sobald die Operation abgeschlossen wird, wird das Ergebnis zurück an den Nutzerbereich übermittelt. Die Applikation erhält nun die Kontrolle zurück, zusammen mit dem Ergebnis der Operation. In diesem Fall die Daten der ausgelesenen Datei oder eine Fehlermeldung.

Aufgabe 2: System-Call-Latenz

Die Latenz eines System-Calls lässt sich im Wesentlichen auf die Differenz zwischen dem Aufruf des System-Calls via einer Applikation oder ähnlichem und die Rückgabe der Kontrolle and den Nutzerbereich seitens des Kernels definieren. Im Wesentlichen muss also die Zeit zwischen diesen Zeitpunkten gemessen werden. Da in der Aufgabe die minimale Latenz gesucht ist, ist weiterhin zu beachten, das die eigentlich Operation des Kernels dementsprechend minimal ausfallen sollte. Für diese Aufgabe habe ich mich für ein Testen anhand eines read()-System Calls entschieden, da ich diesen auch schon bereits in Aufgabe 1 behandelt habe. Das Ergebnis ist eine Zeit von ca. 346 Nanosekunden für einen read() System-Call.

```
jason@DESKTOP-NHUS8SF:~$ cd A2
jason@DESKTOP-NHUS8SF:~/A2$ ./program
Average latency of read() system call: 0.000000346 seconds
jason@DESKTOP-NHUS8SF:~/A2$
```

Jerry Jason Herrmann
1422017

Aufgabe 3:

Der Ansatz den ich gewählt habe um die durchschnittliche Dauer eines Kontextwechsels zu bestimmen ist via eines Abwechselns von 2 Threads über einen Mutex. Das Wechseln von einem Thread zu dem anderen entspricht hierbei einem Kontextwechsel. Das Ganze wird eine gewisse Anzahl lang wiederholt (in meinem Code 1000000x). Die Endzeit wird anschließend durch die Anzahl Wiederholungen geteilt um eine durschnittlichen Dauter zu erhalten.

```
jason@DESKTOP-NHUS8SF:~$ cd A3
jason@DESKTOP-NHUS8SF:~/A3$ ./program
Average context switch time: 27.01 ns
jason@DESKTOP-NHUS8SF:~/A3$
```

Die gemessene Dauer wirkt auf den ersten Blick etwas zu schnell, was sich womöglich darauf zurückführen lässt, dass ich WSL verwenden habe, um das Programm zu testen. Dies könnte zu unterschieden in dem Verhalten von Threads, sowie der Präzision der Messungen im Vergleich zu einer nativen Linux-Umgebung geführt haben.

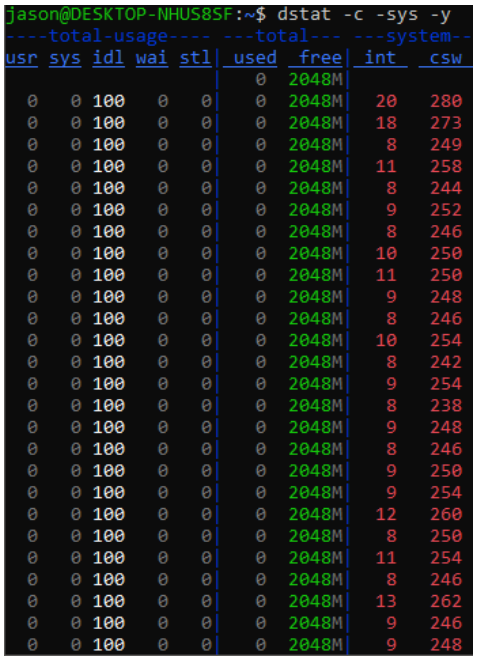
Für das Verwenden von Standardwerkzeugen zur Messung von Kontextwechselzeiten stehen in Linux eine Anzahl an Möglichkeiten zur Verfügung. Hierzu gehören unteranderem Performance Counters, Virtual Memory Statistics, der Task Manager, Process Statistics, Latency Analysis, Resource Statistics, sowie viele weitere Tools. Leider sind die Funktionalitäten vieler dieser Tools innerhalb einer WSL-Umgebung begrenzt, weshalb ich nur wenige selber Testen konnte.

Die Folgende Beiden Tools geben die aktuellen Kontextwechsel pro Sekunde an, anhand wessen sich die durschnittlichen Zeit pro Kontextwechsel mit der aktuellen Auslastung berechnen lässt. Da die Auslastung der CPU hier sehr gering ist, fällt die Anzahl an Kontextwechsel dementsprechend auch niedrig aus. Bei höheren CPU-Auslastungen wäre hier also mit deutlich höheren Werten zu rechnen. Bei ca. 250 Kontextwechseln pro Sekunde wäre die durschnittlichen Zeit bei in etwa 4 Millisekunden. Bei einer höheren Auslastung wäre vermutlich eher mit Zeiten im Mikrosekundenbereich zu rechnen.

Kontextwechselzeiten können negativ von verschiedensten Faktoren beeinflusst werden. Ein Faktor kann die Virtualisierung sein (Wie es vermutlich bei meinem Beispiel der Fall war). Der erhöhte Overhead von dem Managen von virtuellen CPUs könnte zu zusätzlichen Kosten bei Kontextwechseln führen. Weitere Kosten könnte durch Cache Misses oder TLB Misses entstehen, Speicherzugangsfehler bei denen auf langsamere Alternativen zurückgegriffen werden muss. Weitere möglichen Probleme könnten bei komplexeren Abläufen auftreten, wie zum Beispiel Ressourcenkonflikte oder Scheduling-Entscheidungen.

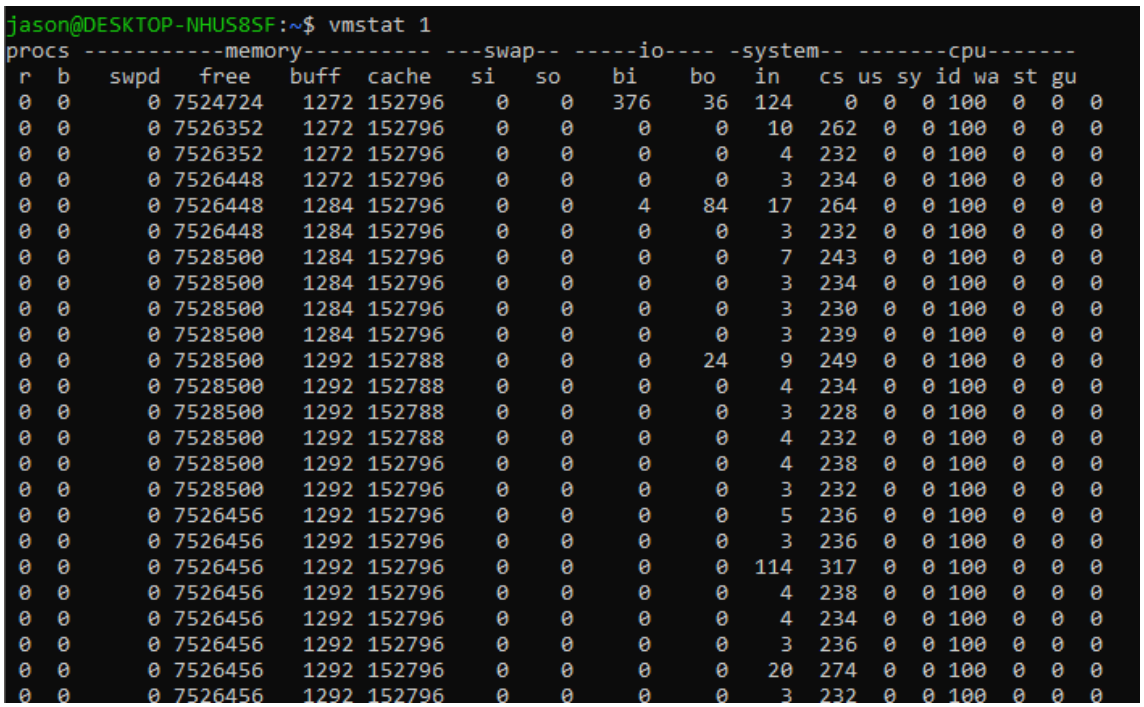
Jerry Jason Herrmann
1422017

Resource Statistics:



```
jason@DESKTOP-NHUS8SF:~$ dstat -c -sys -y
----total-usage---- ---total--- ---system--
usr sys idl wai stl|  used  free|  int   csw
                   |     0  2048M|
  0   0 100   0   0|     0  2048M|   20   280
  0   0 100   0   0|     0  2048M|   18   273
  0   0 100   0   0|     0  2048M|    8   249
  0   0 100   0   0|     0  2048M|   11   258
  0   0 100   0   0|     0  2048M|    8   244
  0   0 100   0   0|     0  2048M|    9   252
  0   0 100   0   0|     0  2048M|    8   246
  0   0 100   0   0|     0  2048M|   10   250
  0   0 100   0   0|     0  2048M|   11   250
  0   0 100   0   0|     0  2048M|    9   248
  0   0 100   0   0|     0  2048M|    8   246
  0   0 100   0   0|     0  2048M|   10   254
  0   0 100   0   0|     0  2048M|    8   242
  0   0 100   0   0|     0  2048M|    9   254
  0   0 100   0   0|     0  2048M|    8   238
  0   0 100   0   0|     0  2048M|    9   248
  0   0 100   0   0|     0  2048M|    8   246
  0   0 100   0   0|     0  2048M|    9   250
  0   0 100   0   0|     0  2048M|    9   254
  0   0 100   0   0|     0  2048M|   12   260
  0   0 100   0   0|     0  2048M|    8   250
  0   0 100   0   0|     0  2048M|   11   254
  0   0 100   0   0|     0  2048M|    8   246
  0   0 100   0   0|     0  2048M|   13   262
  0   0 100   0   0|     0  2048M|    9   246
  0   0 100   0   0|     0  2048M|    9   248
```

Virtual Memory Statistics:



```
jason@DESKTOP-NHUS8SF:~$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- -------cpu-------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st gu
 0  0      0 7524724   1272 152796    0    0   376    36  124    0  0  0 100  0  0  0
 0  0      0 7526352   1272 152796    0    0     0     0   10  262  0  0 100  0  0  0
 0  0      0 7526352   1272 152796    0    0     0     0    4  232  0  0 100  0  0  0
 0  0      0 7526448   1272 152796    0    0     0     0    3  234  0  0 100  0  0  0
 0  0      0 7526448   1284 152796    0    0     4    84   17  264  0  0 100  0  0  0
 0  0      0 7526448   1284 152796    0    0     0     0    3  232  0  0 100  0  0  0
 0  0      0 7528500   1284 152796    0    0     0     0    7  243  0  0 100  0  0  0
 0  0      0 7528500   1284 152796    0    0     0     0    3  234  0  0 100  0  0  0
 0  0      0 7528500   1284 152796    0    0     0     0    3  230  0  0 100  0  0  0
 0  0      0 7528500   1284 152796    0    0     0     0    3  239  0  0 100  0  0  0
 0  0      0 7528500   1292 152788    0    0     0    24    9  249  0  0 100  0  0  0
 0  0      0 7528500   1292 152788    0    0     0     0    4  234  0  0 100  0  0  0
 0  0      0 7528500   1292 152788    0    0     0     0    3  228  0  0 100  0  0  0
 0  0      0 7528500   1292 152788    0    0     0     0    4  232  0  0 100  0  0  0
 0  0      0 7528500   1292 152796    0    0     0     0    4  238  0  0 100  0  0  0
 0  0      0 7528500   1292 152796    0    0     0     0    3  232  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0    5  236  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0    3  236  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0  114  317  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0    4  238  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0    4  234  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0    3  236  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0   20  274  0  0 100  0  0  0
 0  0      0 7526456   1292 152796    0    0     0     0    3  232  0  0 100  0  0  0
```