BS U1 Ergebnisbericht

Karim Amri

02.12.2024

Aufgabe 1

Beim Aufruf von `read` wird der Code aus der dynamischen Bibliothek `/lib/x86_64-linux-gnu/libc.so.6` ausgeführt. Den C-Code dafür findet man in der GNU C Library in `sysdeps/unix/sysv/linux/read.c`:

```
#include <unistd.h>
#include <sysdep-cancel.h>

ssize_t __libc_read (int fd, void *buf, size_t nbytes) {
  return SYSCALL_CANCEL (read, fd, buf, nbytes);
}
```

Das `read` in `SYSCALL_CANCEL (read, fd, buf, nbytes)` ist der Index des `read`-Systemaufrufs in der Systemaufruftabelle. Das Aufrufen des `SYSCALL_CANCEL`-Makros führt dann letztendlich über die Funktion `syscall(read, fd, buf, count)` zum Gegenstück im Linux-Kernel in `fs/read_write.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;
}
```

Beim Aufruf von `syscall(...)` wird dann die Kontrolle an das System abgegeben, welches den Syscall im Supervisor-Modus ausführt.

Aufgabe 2

Man könnte einen „leeren“Syscall aufrufen und die Laufzeit messen. In Linux gibt es leider keinen solchen Null-Syscall, allerdings kommt `getpid()` dem sehr nah, da er kaum Overhead hat und nur einen Wert ausgibt. Das folgende Programm (`a2.c`) implementiert solch ein Experiment:

Listing 1: C-Programm zur Messung der Latenz eines System-Calls

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

#define ITERATIONS 100000000

int main() {
    struct timespec start, end;
    long min_time = 1e9;
    long total_time = 0;

    for (int i = 0; i < ITERATIONS; i++) {

        clock_gettime(CLOCK_MONOTONIC, &start);

        // Syscall
        getpid();

        clock_gettime(CLOCK_MONOTONIC, &end);

        long diff = (end.tv_sec - start.tv_sec) * 1e9
           + (end.tv_nsec - start.tv_nsec);
        if (diff < min_time) {
            min_time = diff;
        }
        total_time += diff;
    }

    printf("Minimale␣Latenz:␣%ld␣ns\n", min_time);
    printf("Durchschnittliche␣Latenz:␣%ld␣ns\n",
       total_time / ITERATIONS);
    return 0;
}
```

Das Programm misst 1000000-mal die Laufzeit von `getpid()` und gibt das Minimum und den Durchschnitt. Auf meinem System (Linux, x86_64) erhalte ich eine Minimallaufzeit von 160 ns und 171 ns im Durchschnitt. Man kann also davon ausgehen, dass die minimale Latenz für einen Syscall im Generellen auf meinem System bei etwas unter 160 ns liegt.

Aufgabe 3

Theorie und Annahmen

Ein Kontextwechsel kann nötig sein, wenn der Scheduler den aktuellen Thread unterbricht, um einen anderen fortzuführen. Man könnte also eine Reihe von Threads starten, die nichts anderes tun, als den aktuellen Zeitpunkt ihrer Ausführung zu messen und diesen in einem Array speichern. So kann man versuchen, die durchschnittliche Dauer eines Kontextwechsels einzugrenzen, wenn man garantieren kann, dass zwischen zwei Threads ein Kontextwechsel stattfindet.

In Abbildung 1 ist die Ausführung zweier identischer Threads A und B dargestellt, zwischen denen ein Kontextwechsel stattfindet. Es wird angenommen,

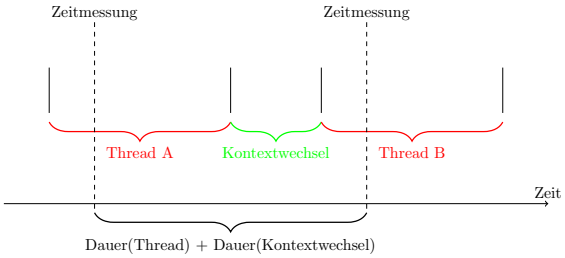

Abbildung 1: Abstand zwischen zwei Zeitmessungen = Dauer eines Threads + Dauer eines Kontextwechsels

dass die Ausführungen der Threads gleich lang dauern und die Zeitmessungen an der gleichen Stelle im Thread passieren. Weiß man jetzt noch, wieviel Zeit ein einzelner Thread beansprucht, kann man die Dauer des Kontextwechsels ermitteln.

Man muss jetzt also noch die Dauer der Ausführung eines Threads messen, es ergibt sich allerdings wieder eine ähnliche Situation: In Abbildung 2 ist die Zeitmessung eines Threads dargestellt. Da die Zeitmessung immer noch als

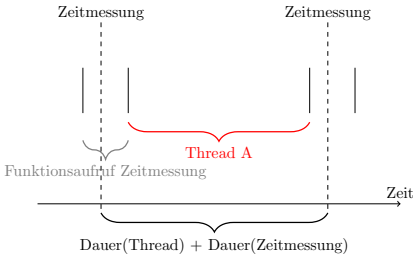

Abbildung 2: Abstand zwischen zwei Zeitmessungen = Dauer eines Threads + Dauer eine Zeitmessung

Funktion einer Programmiersprache ausgeführt wird und nicht magischerweise

augenblicklich mit infinitesimal geringer Dauer geschieht, muss auch die Dauer der Zeitmessung beachtet werden. Es ergibt sich bei dieser Messung also das folgende Verhältnis: Dauer eines Threads = Abstand der Zeitmessungen - Dauer einer Zeitmessung.

Bei der Messung der Dauer des Funktionsaufrufs der Zeitmessung haben wir wieder dasselbe Problem, wie in Abbildung 3 dargestellt. Diesmal ergibt sich

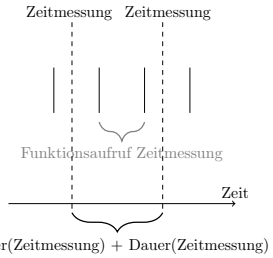

Abbildung 3: Abstand zwischen zwei Zeitmessungen = zweimal Dauer einer Zeitmessung

das folgende Verhältnis: Abstand zwischen zwei Zeitmessungen = Dauer der Funktionsaufrufe zweier Zeitmessungen.