2 Übungsaufgabe #2: Signale als Unterbrechungen

In der Vorlesung wurde die Behandlung von Unterbrechungen auf realen Prozessoren besprochen. In dieser Aufgabe soll die technische Basis für die Unterbrechungsbehandlung durch den Einsatz von Signalhandlern nachgebildet werden.

2.1 Signale in mehrfädigen Prozessen

Das Unix-Signalmodell sieht eine Menge von Signalen pro **Prozess** vor. Dies gilt auch für mehrfädige Prozesse, d.h. die Zuordnung von Signalen zu Signalbehandlern ist in einem Unix-Prozess für alle virtuellen Prozessoren (Threads) gleich. Eine eigene Funktion zur Signalbehandlung kann man mit `sigaction(2)` installieren. Diese gilt dann für alle Fäden innerhalb eines Prozesses.
Die Signalmaske hingegen ist in einem mehrfädigen Prozess spezifisch für den jeweiligen Faden, so dass man auf den virtuellen Prozessoren unterschiedliche Mengen von Signalen blockieren und zulassen kann. Die Signalmaske eines einzelnen Fadens kann man mit `rt_sigprocmask(2)` verändern, das ein zu `sigprocmask(2)` kompatibles Interface hat.

2.2 Schnittstelle für die Unterbrechungsbehandlung

In dieser Aufgabe sollt ihr eine einfache (einstufige) Unterbrechungsbehandlung implementieren. Dabei sollt ihr die notwendigen Systemaufrufe hinter einer vom Wirtssystem unabhängigen Schnittstelle kapseln. Folgende Funktionalität soll unterstützt werden:

- Zuweisen von Unterbrechungsbehandlungen zu Signalen
- Sperren/Zulassen von einzelnen Signalen
- Senden von „*Interprozessorinterrupts*“ zu den anderen virtuellen Prozessoren mittels `tgkill(2)`

Es sollen drei unterschiedliche Varianten der Unterbrechungszustellung implementiert werden:

1. Signale immer auf dem virtuellen Prozessor behandeln, auf dem das Signal ankommt.
2. Signale immer auf einem vorbestimmten Prozessor behandeln.
3. Signale über alle Prozessoren gleichverteilen.

Im Rahmen von Aufgabe 2 sollt ihr euch zwei unterschiedliche Signale heraussuchen und einmal Variante 1 und Variante 2 direkt in der Unterbrechungsbehandlung implementieren.

2.3 Zeitgeber

Die dritte Interruptquelle soll ein Zeitgeber sein, der alle 100 Millisekunden eine Unterbrechung erzeugt. Mit Hilfe von `setitimer(2)` kann man ein entsprechendes Signal erzeugen, so dass man regelmäßige `SIGALRM`s erhält. Diese sollen nun über alle Prozessoren gleichverteilt werden (Variante 3).

2.4 Nebenläufig benutzbare Ausgabeoperationen

Zum Testen der Implementierung könnt ihr Bildschirmausgaben mit den C++ Stream-Operatoren nur eingeschränkt benutzen, da die nebenläufige Benutzung des Terminals in unterschiedlichen Fäden eine reentrante Implementierung der Ausgabeprimitiven erfordert. Jedoch ist gerade dies bei den Typkonvertierungsfunktionen aus der Standardbibliothek, die z.B. für die Ausgabe von Ganzzahlen verwendet werden, nicht gegeben.
Implementiert daher einen eigenen, reentranten Ausgabe-Stream und passende Konverter zur Ausgabe von Integerzahlen, die von mehreren Fäden gleichzeitig benutzt werden können. Dazu könnt ihr Teile euerer Lösung (nämlich genau die Zahlenkonvertierungsoperatoren) aus der Veranstaltung *Betriebssysteme* wiederverwenden. Beachtet auch, dass verwendete Puffer innerhalb der Ausgabeprimitiven nicht von mehreren Fäden verwendet werden, um Race Conditions beim Beschreiben und Leeren der Puffer zu vermeiden. Zur Ausgabe der Daten auf der Konsole könnt ihr den Systemaufruf `write` verweden, der auch problemlos von mehreren Threads parallel aufgerufen werden kann.

Aufgaben:

→ Implementierung einer Schnittstelle zur Registrierung und Verwaltung von Unterbrechungsbehandlungen als Signalhandler.

→ Implementieren der drei in 2.2 beschriebenen Varianten der Unterbrechungsbehandlung.

→ Bereitstellung einer regelmäßigen Zeitgeberunterbrechung

→ Implementierung eines reentranten Ausgabestromes

Hinweise:

- Zustellung von Signalen an einen einzelnen Faden kann unter Linux mit dem Systemaufruf `tgkill` erfolgen. Die Threadgruppen-ID, die `tgkill` als Parameter erwartet, ist die `PID` des Prozesses, die ihr über `getpid` abfragen könnt.
- Jeder Faden braucht eine eigene Signalmaske. Diese kann unter Linux mit dem Systemaufruf `rt_sigprocmask` gesetzt und manipuliert werden. Diese Funktion ist bist auf einen vierten Parameter interfacekompatibel zu `sigprocmask`. Dieser Parameter gibt die Größe der als Parameter verwendeten Signalmasken (`sizeof(size_t)`) an. Jedoch unterscheidet sich `sizeof(size_t)` innerhalb des Linuxkerns von `sizeof(size_t)` im Userspace. Hier muss man den Wert, den der Linuxkern annimt, als vierten Parameter übergeben, und der ist `8`.
- Für den Systemaufruf `rt_sigprocmask` existiert in der LibC keine Wrapperfunktion, so dass ihr ihn direkt per `syscall`-Macro aufrufen müsst.
- Signale kann man mit dem Werkzeug **kill(1)** an *Prozesse* schicken.

2.5 Abgabe: am 01.06.2011