
Signale

By rac

Published: 13.01.2008 - 17:54

- Signale sind asynchron ("überraschend")
- ein Signal, welches den Programmfluss unterbricht kann an jeder Stelle im Code eintreffen
- Division by Zero
- Segmentation Fault
- Timer I/O
- I/O Operation fertig
- SIGUSR
- Suspend
- SIGTERM

- Bedingung für Thread Signalhandling Mechanismus
- muss kompatibel bleiben
- es muss festgelegt werden können, wer ein Signal bekommt/sieht
- was ist einem Thread im Signalhandler erlaubt, sodass dies die sonstige Arbeit des Threads nicht beeinflusst?

- ein Signal geht immer an einen Prozess, und nicht an einen Thread
- Maske, welche die Reaktion eines Prozesses auf ein Signal festlegt (sigaction) wird von allen Threads geteilt
- jeder Thread hat Maske, welche definiert, welche ankommenden Signale er sieht
- Verhalten der Pthread Werkzeuge selbst innerhalb eines Signalhandlers ist nicht definiert!

-
- keine Synchronization mit pthread Mitteln innerhalb eines Signalhandlers möglich

Traditionelle Signalverarbeitung

- sigaction erlaubt es einem Signal eine Aktion zuzuordnen
- SIG_IGN Signal ignorieren
- SIG_DFL Default. Reaktion ist abhängig vom empfangenen Signal. Die meisten Signale beenden den Prozess, einige werden jedoch ignoriert. SIGSTOP bzw. SIGTSTP unterbrechen und SIGCONT nimmt den Prozess wieder auf.
- bearbeiten Signal wird empfangen und eine spezifische Handling Routine ausgeführt
- die Reaktionen auf SIGSTOP und SIGKILL können nicht verändert werden
- bei Signal wird Prozess unterbrochen und Handler ausgeführt. Danach wird die Ausführung am Unterbrechungsort fortgesetzt
- SIGCONT setzt einen Prozess fort welcher durch wait, sigsuspend, sleep oder pause unterbrochen wurde

Signalhandling und Threads

Beim Empfang, werden verschiedene Signale unterschieden:

- Synchrone Signale
- ausgelöst durch Operation des Programms
- SIGFPE - Teilung durch Null
- SIGSEGV - illegaler Speicherzugriff
- SIGPIPE - Pipe ist kaputt
- ...

-
- Signal wird an den Thread ausgeliefert, der das Problem ausgelöst hat
 - explizit mit `pthread_kill` innerhalb eines Prozesses an einen anderen Thread gesendete Signale (kein IPC!)

 - Asynchrone Signale
 - können nicht einfach auf einen bestimmten Thread zurückgeführt werden
 - asynchron dahingehend, dass sie unabhängig von den Aktivitäten bzw. dem ausgeführten Code sind
 - üblicherweise Job Kontroll Signale
 - SIGALRM
 - SIGHUP
 - SIGINT
 - SIGKILL
 - SIGUSRx

 - diese können, sofern nicht maskiert, von einem beliebigen Thread bearbeitet werden

Wie wurde Signal ausgelöst	Was hat's ausgelöst	Ziel des Signals	Bearbeitung des Signals
synchron	System aufgrund einer Exception	spezifischer Thread	durch auslösenden Thread
synchron	<code>pthread_kill</code>	spezifischer Thread	durch Ziel-Thread
asynchron	externer Prozess mittels <code>kill</code>	Prozess	durch Thread mit entsprechender Maske

Per Thread Signalmasken

- wie Prozesse auch besitzen Threads Signalmasken, welche anzeigen, welche Signale sie entgegennehmen (deblockiert) und welche nicht (blockiert)

- Masken werden geerbt
- vom Prozess
- vom Thread, der fork ausgeführt hat
- vom Thread, welcher pthread_create ausgeführt hat

- pthread_sigmask
- ein ankommendes, asynchrones Signal wird an genau einen Thread ausgeliefert

Threads in Signal Handlern

- Aufrufe, welche aus Signal Handlern gemacht werden können werden als "asynchronous signalsafe" bezeichnet
- sind "reentrant", da mehrere Ausführungsstränge gleichzeitig in ihnen aktiv sein können
- siehe Liste von Systemfunktionen p170 im PThreads Buch
- Pthread Funktionen nicht in der Liste
- Synchronization über sem_post
- sigwait verwenden, um auf Signal zu warten
- Signale maskieren
- Thread erstellen, welcher sigwait durchführt
- demaskieren

Probleme mit Signalen

-
- manche POSIX Funktionen geben EINTR zurück, wenn sie von einem Signal gestört werden
 - sie müssen darauf noch einmal ausgeführt werden

[< PThreads und Unix up Threadssichere Bibliotheks- und Systemfunktionen >](#)