Die Vorlesung fokussierte auf den Abstraktionen des POSIX-Standards. Diese haben aber durchaus einige Defizite und Probleme, die von diversen UNIX-Varianten bei ihrer Weiterentwicklung unterschiedlich gelöst worden sind und bislang nicht Teil des POSIX-Standards wurden.

Wir fokussieren uns hier insbesondere auf die Schwächen der *select*- und *poll*-Schnittstelle.

Die Schnittstellen *select* und *poll* haben zahlreiche Nachteile:

- Bei einer großen Zahl von n Dateideskriptoren, gibt es bei jedem Aufruf einen Aufwand von $O(n)$, um die Datenstruktur im Kern aufzubauen und kurz danach wieder abzubauen, obwohl bei vielen Anwendungen die Menge der Dateideskriptoren und der für uns relevanten Ereignisse sich nicht sehr rasch ändert. Auch die Anwendung hat einen Aufwand von $O(n)$ um herauszufinden, welches Ereignis eingetreten ist.
- Sie sind beschränkt auf I/O-Ereignisse. Ereignisse im Kontext der Prozesse (z.B. Terminierung eines Kindprozesses) oder im Dateisystem sind nicht integriert.
- Probleme bei der Parallelisierung zur Skalierung: *C10k problem* in Verbindung mit dem *thundering herd problem*.

Einige Begriffsdefinitionen für die folgenden Diskussionen:

- **Ereignis**: Beliebige Zustandsveränderungen im Kernel, die erfasst werden können, z. B. Eintreffen eines Netzwerkpakets, Eröffnen einer Verbindung, Terminierung eines Prozesses, Zustellung eines Signals usw.
- **Filter**: Spezifikation einer Klasse von Ereignissen, z.B. ein Dateideskriptor und das Eintreffen einer Eingabe.
- **Filterliste**: Menge von beliebig vielen Filtern, die beschreiben, an welchen Ereignissen Interesse besteht.
- **Aggregation**: Mehrere Ereignisse der gleichen Art (z.B. aufeinanderfolgende Pakete bei der gleichen Netzwerkverbindung) können u.U. zu einem Ereignis aggregiert werden.

- Die Filter bestehen jeweils nur aus Dateideskriptoren und den zugehörigen Ereignismasken (z.B. *POLLIN* und *POLLOUT*).
- Die möglicherweise sehr umfangreiche Filterliste wird bei jedem Aufruf von *poll* und *select* übergeben.
- Nach dem Aufruf beginnt die umfangreiche Suche nach den Ereignissen, die stattgefunden haben.
- Sowohl beim Kernel als auch bei der Anwendung haben wir einen Aufwand von $O(n)$, wenn n der Umfang der Filterliste ist.

Idee: Wir verlagern die Verwaltung der Filterliste in den Kernel.
Vorgehensweise:

- Zunächst muss eine Filterliste im Kernel angelegt werden.
- Es gibt Systemaufrufe, die das Hinzufügen und Löschen von Filtern in der Filterliste erlauben.
- Es ist möglich, auf das Eintreffen eines von der Filterliste erfassten Ereignisses zu warten.

poll und *select* sind zustandslos (*stateless*), d.h. die Filterlisten sind nur während eines Aufrufs von *poll* oder *select* relevant.

Wenn jetzt eine im Kernel existierende Filterliste unabhängig von einem aktuell wartenden Systemaufruf ein Ereignis erfasst, was soll dann geschehen?

- Soll das Ereignis aufgezeichnet werden?
- Können Ereignisse, die den gleichen Filter betreffen, aggregiert werden?
- Was passiert, wenn mehrere Filterlisten sich für das gleiche Ereignis interessieren?
- Was passiert bei *fork*?

Aus der Elektrotechnik werden hier gerne die Begriffe pegelgesteuert (*level-triggered*) und flankengesteuert (*edge-triggered*) verwendet.

Angenommen, wir haben einen Eingangssignal, bei dem entweder Spannung anliegt oder nicht. Was soll dann als Ereignis betrachtet werden?

- Pegelgesteuert (*level-triggered*): Das Ereignis ist immer dann zu erfassen, wenn wir nachsehen und die Spannung zu diesem Zeitpunkt anliegt.
- Flankengesteuert (*edge-triggered*): Nur der Vorgang des Hochgehens der Spannung wird als Ereignis betrachtet. Das Ereignis wird somit nur ein einziges Mal erfasst.

- Eine flankengesteuerte Ereignisbehandlung setzt einen entsprechenden Zustand im Kernel voraus.
- Da sowohl *poll* als auch *select* zustandslos sind, können sie nur pegelgesteuert erfolgen:
  - Beim Aufruf sehen wir nach, ob bei einem der Filter der „Pegel anliegt“ (z.B. vorhandene Eingabe).
  - Falls ja, enden *poll* bzw. *select* sofort und melden dies als Ereignisse.
  - Falls nein, warten wir darauf, dass bei einem der Filter der Pegel hochgeht.

Wenn die Filterliste in den Kernel wandert, haben wir folgende Möglichkeiten:

- Wir können die Filterliste vollkommen ignorieren, wenn wir uns nicht in einem Systemaufruf befinden, der auf das Eintreffen eines Ereignisses dieser Filterliste wartet.
- Oder wir können von einer im Kernel existierenden Filterliste erfasste Ereignisse laufend aufzeichnen und in eine Warteschlange einreihen, auf die zugegriffen wird, sobald die nächste Abfrage- oder Warte-Operation stattfindet.

Es ergeben sich daraus unterschiedliche Ansätze in der Programmierung:

- pegelgesteuert: Nachdem wir auf das Eintreffen der Ereignisse gewartet haben, müssen wir diese verkonsumieren (d.h. den Pegel wegnehmen), bevor wir erneut warten. Sonst erhalten wir die gleichen Ereignisse erneut.
- flankengesteuert: Wenn wir gewartet haben, müssen wir darauf achten, alle eingetroffenen Ereignisse vollständig abzuarbeiten, weil diese sonst verlorengehen können. Eine Warteoperation mit einem Filter, der auf Eingabe wartet, kann dann auch trotz existierender Eingabe blockieren, weil es schon eine Gelegenheit gab, diese abzugreifen.

Bei konkurrierenden Filterlisten im Kernel gibt es je nach Ansatz unterschiedliche Probleme bzw. Fragestellungen:

- pegelgesteuert: Nach dem Eintreffen eines Ereignisses werden möglicherweise sehr viel mehr Prozesse bzw. Threads mit der Abarbeitung beschäftigt, obwohl einer das hätte alleine behandeln können (*thundering herd problem*).
- flankengesteuert: Wird es von allen Filterlisten erfasst oder nur exklusiv von einer einzigen?

Was passiert, wenn mehrere konkurrierende Filterlisten das gleiche Ereignis erfassen?

- Bei Exklusivität erreicht das Ereignis (im Idealfall) nur einen einzigen.
- Die Entscheidung, wer es erhält, fällt typischerweise beim Warten.
- Implementiert wird es dadurch, dass das Ereignis nach dem ersten Abruf sofort für alle anderen entfernt wird.

Exklusivität ist orthogonal zur Frage, ob es pegel- oder flankengesteuert ist.

Einige der Lösungen bieten eine Oneshot-Variante an.

- Bei dieser Option wird ein Filter aus der Filterliste entfernt, sobald ein entsprechendes Ereignis eingetreten ist.
- Für konkurrierende Situationen ist dies nur relevant, wenn eine Filterliste z.B. von konkurrierenden Threads gemeinsam genutzt wird.

Folgende Erweiterungen gibt es, die allesamt nicht zu POSIX gehören:

- */dev/poll* wurde von Sun für Solaris 7 bzw. 8 um 1999 eingeführt. Ist ausschließlich pegelgesteuert und beschränkt sich auf die Reduktion des Aufwands von $O(n)$ auf $O(1)$ im Vergleich zu *poll* oder *select*.
- *kqueue* wurde für FreeBSD entwickelt und stand ab FreeBSD 4.1 im Juli 2000 zur Verfügung. Das verbreitete sich auf andere BSD-Varianten und auf MacOS. Zahlreiche weitere Ereignisklassen wurden hinzugefügt. Ist überwiegend pegelgesteuert, Exklusivität und Oneshot werden unterstützt. Mehrere Threads können sich eine kqueue teilen, über *fork* kann es nicht vererbt werden.
- *epoll* kam zuerst mit dem Linux-Kernel 2.5.44 im Oktober 2002 und operiert per Voreinstellung pegelgesteuert, kann aber auch flankengesteuert arbeiten. Mehrere Threads oder Prozesse können sich eine Filterliste teilen, Oneshot ist dann möglich und sinnvoll. Exklusivität wird seit kurzem unterstützt.