Betriebssysteme und Systemsoftware Übungsblatt 1
Übungsgruppe 7

Felix Kiunke, 357322
Philipp Hochmann, 356148
Daniel Schleiz, 356092

Aufgabe 1.1 done ☺

Aufgabe 1.2

a)

In heutigen Rechnersystemen ist der Von-Neumann-Flaschenhals nur noch abgeschwächt von Bedeutung, da mittlerweile Prozessoren eigene Cache-Systeme, wie zum Beispiel L1, L2 und L3 Caches, besitzen, welche es ermöglichen, Daten, die in geraumer Zeit benötigt werden, aus dem (Haupt-) Speicher vorzuladen. Diese Caches sind meist direkt an die Prozessoren angebunden und ermöglichen extrem schnelle Zugriffszeiten, wodurch zum Einen Speicherlatenzen durch das Bus-System vermieden werden können und zum anderen das Bus-System unter Umständen entlastet wird. Somit können die Daten, sofern sie korrekt in den Cache geladen wurden, schnell verarbeitet werden ohne den Bus zu belasten.

Außerdem gibt es die Möglichkeit, für Anwendungen, in denen die Geschwindigkeit von hoher Bedeutung ist, Parallelbussysteme zu verwenden, welche zwar teurer in der Anschaffung sind, jedoch einen höheren Datendurchsatz ermöglichen als bei einem normalen seriellen Bussystem, wodurch der von Neumannsche Flaschenhals relativiert wird.

b)

Für einen Programmierer kann es wichtig sein, zu wissen, ob eine Bibliotheksfunktion einen Systemaufruf verursacht, da dabei ein Kontextwechsel in den privilegierten Modus, also dem Kernel-Modus stattfindet. Prinzipiell sind die Bibliotheksfunktionen so angelegt, dass die damit verbundenen Systemaufrufe nicht allzu viel Schaden anrichten können, dennoch sind Systemaufrufe potenziell sehr mächtig und können bei falscher Handhabung zum Beispiel wichtige Datenstrukturen anderer Komponenten überschreiben oder ähnliches, weshalb es nützlich ist, dass sich der Programmierer bewusst ist, wenn ein Systemaufruf geschieht, da er dadurch eventuell vorsichtiger damit umgeht. Zudem können die Kontextwechsel viel Zeit kosten und damit die Performance beeinflussen, was als Programmierer mitbedacht werden sollte.

c)

Polling fragt periodisch den Status von zum Beispiel Prozessen, Ein-/Ausgabegeräten oder ähnlichem ab. Der Vorteil gegenüber Interrupts besteht darin, dass Polling prinzipiell relativ leicht implementierbar ist und meistens keine zusätzliche Hardware erfordert. Generell ist Polling vorteilhaft, wenn permanent eine zyklische Aktivität vorliegt, die es zu beobachten gilt, insofern man die Abfragefrequenz passend wählt und einem die periodische Abfrage genügt. Ebenfalls ist Polling sinnvoll zur Synchronisation in einer parallelen Anwendung, wodurch abhängig von den abgefragten Zuständen vorgegangen werden kann. Außerdem muss Polling oft fest in den Programmcode eingebunden werden, wodurch der Code vorhersehbarer ist als bei einer Interrupt Steuerung.

Treten allerdings die zu beobachtenden Ereignisse eher selten auf, so ist Polling sehr ineffizient und Interrupts sinnvoller, da so zum Einen auf ein Ereignis direkt reagiert werden kann, was unter Umständen bei kritischen Fehlern notwendig ist, und nicht erst auf die Abfrage durch das Polling gewartet werden muss. Zum Anderen verbraucht Polling durch die permanenten Abfragen somit auch immer Systemressourcen, was bei Interrupts nur der Fall ist, wenn tatsächlich das Ereignis vorliegt, wodurch die allgemeine Effizienz gegenüber Polling meist höher ist.

d)

Dies ist möglich durch sogenanntes Multitasking, welches durch das Betriebssystem ermöglicht wird. Zwar ist die CPU im Prinzip nur in der Lage, einen Prozess zur gleichen Zeit auszuführen, jedoch kann das Betriebssystem den Prozessor so ansprechen, dass zwischen Prozessen nach bestimmten Zeiteinheiten gewechselt wird, zum Beispiel wenn ein Prozess auf Daten aus dem Speicher warten muss wird währenddessen ein anderer Prozess abgearbeitet. Es entsteht so die Illusion, dass scheinbar mehrere Prozesse gleichzeitig verarbeitet werden.

Zudem besteht für einen Prozessor die Möglichkeit, insofern die passende Hardware vorhanden ist, Hyper-Threading zu betreiben, welche mehrere komplette Registersätze und ein komplexes Steuerwerk erfordern. Dies ermöglicht die simultane Abarbeitung mehrerer Threads.

Aufgabe 1.3

a)
sed 's/5/6/'

b)
Gibt für jede Datei im aktuellen Verzeichnis, die mit einem kleinen d beginnt,
jeweils zeilenweise alle Zeichen vor dem ersten Leerzeichen aus.

c)
grep -E '^[[:digit:]]{5} \w+(\w+)?$' -B 2 emails

Aufgabe 1.4

a)

Ein Syscall ist der von einem Anwendungsprogramm erzeugte Aufruf von vom Betriebssystem bereitgestellten Methoden zum Ansprechen der Hardware, wodurch die Hardware in vollem Umfang genutzt werden kann. Dabei findet ein Kontextwechsel vom User-Mode in den Kernel-Mode statt, nach Abarbeitung des Systemaufrufs wird wieder in den User-Mode gewechselt und das zu bearbeitende Programm wird fortgeführt.

b)

exec: Führt ein Programm aus und ersetzt dabei den ursprünglichen Prozess durch einen neuen Prozess, wodurch eine Rückkehr zum aufrufenden Prozess nicht mehr möglich ist.

stat: Gibt verschiedene Informationen über eine Datei oder über das Dateisystem dieser wieder.

ioctl: Ein Input/Output Control, welches in der Lage ist, verschiedene Parameter eines Geräts zu manipulieren.

mmap: Lädt Dateien oder Geräte in den Speicher des Systems.

brk: Ist in der Lage, einem Prozess zusätzlichen Speicher zu allozieren beziehungsweise Speicher zu abzunehmen.

c)

strace beobachtet den Aufruf und die Abarbeitung eines Kommandos und speichert dabei in einer Datei nützliche Informationen über getätigte Systemaufrufe und Signale von Prozessen. Somit eignet sich das Programm strace sehr gut zum Debugging und auch sonst als Diagnosetool.

d)

Zunächst die Analyse der Kommandos:

ls /etc listet die Dateinamen aller (nicht-versteckten) Dateien im Ordner /etc

ls -l /etc listet die Dateinamen aller (nicht-versteckten) Dateien im Ordner /etc und gibt zu jeder Datei folgende zusätzliche Informationen:
Zugriffsrechte, Besitzer, Dateigröße, Erstellungsdatum

ls -la /etc wie ls -l /etc, aber mit versteckten Dateien (also Einträge, die mit . starten)

Der erste Befehl verwendet lediglich ein paar wenige Male open und fstat, um gewisse Informationen über Rechte und sonstige Eiegnschaften abzufragen und verwendet dann schließlich das einzige mal stat auf /etc, um die Ordnerinhalte aufzulisten. Insgesamt werden nur 20 Syscalls getätigt.

Beim zweiten Befehl sieht das sehr anders aus: Aufgrund der Tatsache, dass dieser zu jeder aufgelisteten Datei die oben genannten zusätzlichen Informationen ausgibt, müssen viel mehr Syscalls gemacht werden, um an die benötigten Informationen ranzukommen. Während die Anzahl der Calls von open und fstat nur leicht angestiegen ist, werden im Vergleich zum vorigen Befehl sehr viel mehr stat Calls durchgeführt werden, um eben Sachen wie zum Beispiel die Zugriffsrechte zu erfahren, zudem müssen nun auch ziemlich viele lstat Calls benutzt werden, um auf die Dateien im /etc Ordner zuzugreifen. In Summe sind es nun 519 calls, also ein Vielfaches vom ersten Befehl.

Im Vergleich zum zweiten Befehl kommen beim dritten Befehl lediglich ein paar wenige stat und lstat Calls dazu. Dies lässt sich dadurch erklären, dass sich im Ordner /etc ein paar wenige "versteckte" Dateien befinden, auf die auf gewohnte Weise mit lstat zugegriffen wird und mithilfe von stat zusätzliche Informationen abgefragt werden wie z.B. Zugriffsrechte.