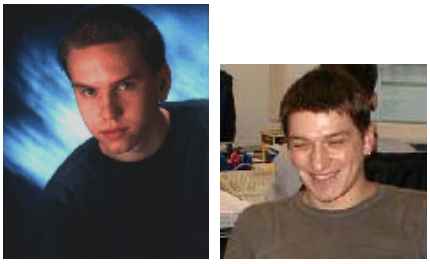
Eine Hand voll Unix-Umsteigen leicht gemacht

Sebastian Mahr

23. April 2001



David Metzger, Nico Fabian, Sebastian Mahr, Daniel Ortlepp

Dieses Dokument wurde in LaTeX erstellt.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Was soll das ganze Papier?

Dies ist nun der Unix-Einsteiger-Kurs, besser gesagt, die schriftlichen Aufzeichnungen dazu. Das Hauptaugenmerk liegt unsererseits auf dem Kurs der CLASS Firmengruppe, da nur dort genauer auf Fragen eingegangen werden kann und nur dort das direkte Nachvollziehen des Erlernten am Rechner möglich ist. Und genau das ist es was in unseren Augen einzig zum Erfolg führen kann: Das Arbeiten am Rechner. Dies kann sich natürlich nicht nur auf die kurze Zeit im Kurs vor Ort beschränken, vielmehr sollte man versuchen so viel Zeit wie möglich am heimischen oder schulischen (Unix-) Rechner zu verbringen um Neugelerntes aber auch und vor allem eigene Ideen auszuprobieren. Oben Genanntes soll natürlich keine Aufforderung sein eine eventuelle Familie und/oder Arbeit zu vernachlässigen, das "isses nich'" wert, vielmehr eine Anregung zum Weiterdenken und -handeln. Und genau hierbei sollten diese Unterlagen unterstützend helfen.

1.2 Themenauswahl

Der Bereich, der hier so salopp *Unix-Systemverwaltung* genannt wird, umfasst eine Vielzahl von Tätigkeiten, ein grosses Wissen im Bereich der verschiedensten Unix-Varianten[1] und Netzwerktechniken sowie -diensten. Wir werden versuchen einen allgemeinen Überblick über die Herkunft von Unix, die Unterschiede zu gängigen PC-Betriebssytemen und die Möglichkeiten, die uns dieses System bietet, zu geben und weiterhin auf die Einrichtung und Pflege einer 'typischen' Installation eingehen - letzteres am Beispiel der in der CLASS Firmengruppe verwendeten Hard- und Software.
Wie so vieles im Leben hat auch Unix in fast jedem Bereich verschiedene Lösungsansätze für einzelne Probleme. So gibt es allein z.B. mindestens sechs verschiedene *Shells*[2]und eine Vielzahl von Unix-Herstellern und somit -Varianten. In der Regel beziehen wir uns auf die hier verwendeten Utilities, versuchen aber auch, soweit nötig und sinnvoll, die "Konkurenten" dazu anzusprechen. Prinzipiell behandeln wir die *SystemV-Form* von Unix (im Gegensatz zu *BSD*), da wir diese hier und andernorts erfolgreich einsetzen und mit dem Betriebssystem *Linux* auch ein umfangreiches, kostengünstiges und leistungsfähiges Unix-Derivat für Intel-basierte Rechner zur Verfügung steht, das auch an SystemV angelehnt ist. Wer noch ein paar MegaBytes (ab 200, ohne Grenze nach oben) auf seiner Festplatte frei hat und auch sonst einen zeitgemäßen Rechner besitzt, sollte sich deshalb ruhig o.g. Betriebssystem installieren, um auch daheim, in Ruhe, eventuell in Zusammenhang mit einer schönen Tasse Kaffee oder Tee, die Vorzüge und vielleicht auch die Faszination dieses Systems kennen und schätzen zu lernen.

Der Erste Teil wird sich ausschließlich auf die Hardware und den sachgemäßen Umgang damit beschränken, der zweite Teil jedoch wird dann schon in die Tiefen von Unix einführen.

[1]und davon gibt es unwahrscheinlich viele, wie wir sehen werden.
[2]Kommandointerpreter, vergleichbar mit command.com bei DOS

Kapitel 2

Hardware

2.1 Komponenten einer Sun

Eine Sun besteht wie jeder Computer aus einem oder mehreren Mainboards, Arbeitsspeicher und aus diversen Laufwerken, wie z.B. Festplatte, Diskettenlaufwerk, CD Rom, usw. Allerdings unterscheiden sich die Komponenten teilweise erheblich voneinander.

Mainboard

Als erstes ist das *Mainboard*[1] zu nennen. Die verschiedenen Hauptplatinen unterscheiden sich je nach System voneinander.
Hier sind einige Systeme aufgeführt und die entsprechenden Schemata:

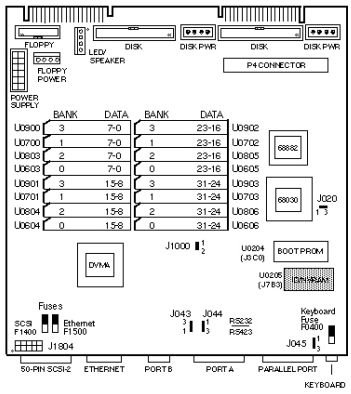


Abbildung 2.1: Mainboard einer SUN 3/80

[1] deutsch: Hauptplatine - Hierauf sind alle Bausteine und Schaltkreise integriert, um ein skalierbares System mit Bussen und Erweiterungsmöglichkeiten zu schaffen.

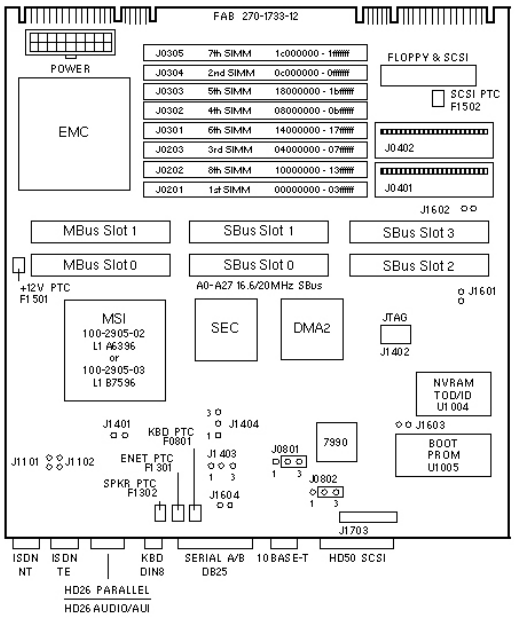


Abbildung 2.2: Mainboard einer SparcStation 10

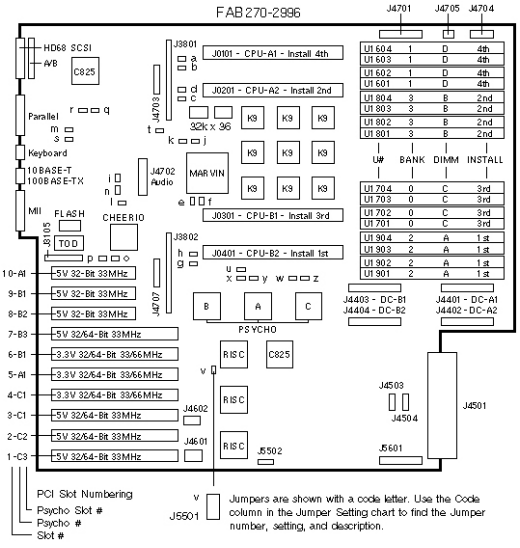


Abbildung 2.3: Mainboard einer Enterprise E450

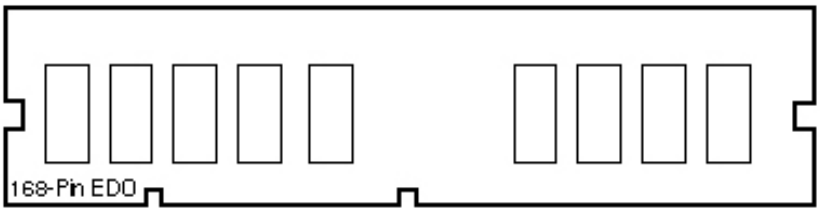


Abbildung 2.5: Speicher 168 Pin

Speicher:

Die verschiedenen Speicherbausteine haben, sofern sie von SUN sind, eine Nummer an der man erkennt, wo diese verwendet werden können. Achtung!! Einige Speicherbausteine sehen so aus, wie SDRAM oder DDRRAM, diese dürfen aber nicht mit den normalen PC-Speicher vertauscht werden, da bei einigen die Speicherspannung anders ist.

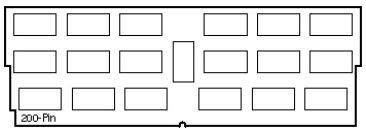


Abbildung 2.4: Speicher 200 Pin

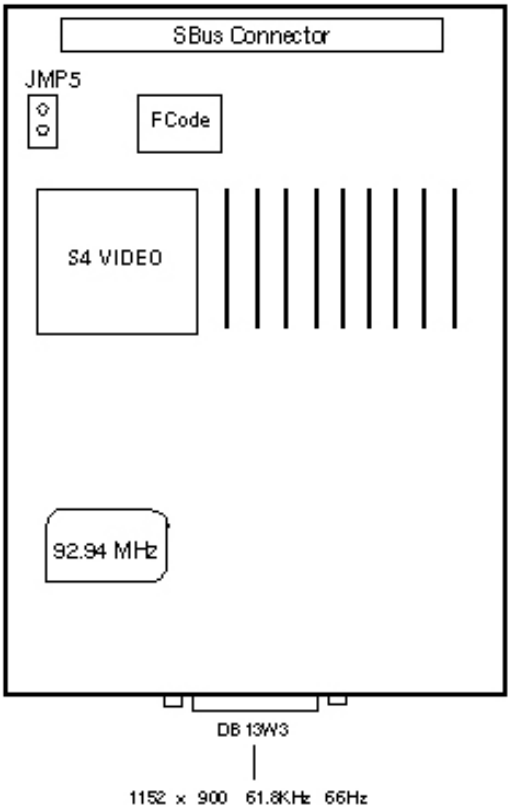


Abbildung 2.6: SBus Grafikkarte

Grafikkarte

Bei einigen Mainboards ist sie bereits onboard[2], benötigt manchmal aber extra Speicher.
Achtung!! Die 8 bit Grafikkarten haben eine Standardfestfrequenz von 66Hz bzw 76Hz.

[2]deutsch: auf Der Hauptplatine - Es muss keine Grafikkarte mehr integriert werden, da diese Funktion vom Mainboard übernommen wird.

Es gibt drei verschiedene Stecksysteme für Grafikkarten, welche nicht kompatibel sind:

- SBus Grafikkarten
- UPA Bus Grafikkarten
- AFX Bus Grafikkarten

Nun wird nur noch eine *Tastatur* und eine *Maus* benötigt und der SUN-Rechner kann ohne weitere SUN-Komponenten betrieben werden. Die Tastatur wird seriell angesteuert.

Achtung!! Der Stecker paßt nicht An PC's

Als Anzeigegerät kann jeder handelsübliche *Monitor* verwendet werden, der die Auflösungen und die Frequenzen der Grafikkarte schafft.

Mit welchen Komponenten kann eine Sun aufgerüstet werden?

Bei älteren Sun-Rechnern konnten nur Komonenten von Sun verwendet werden, da der SBus eine Spezifische Schnittstelle ist und keine PC-Hardware in diesen Steckplatz passt.
Für den SBus gibt es folgende Komponenten:

- Grafikkarten 8Bit 24 Bit
- SCSI wide single end-, differential- und Narrow-karten
- Netzwerkkarten 10MBit und 100Mbit auch mehrfach Netzwerkkarten
- Prozessorkarten mit Intel Prozessoren

Die neueren Systeme haben neben den Sun-Spezifischen Schnittstellen auch PCI Steckplätze. In die kann jede PC PCI-Karte eingesetzt werden.

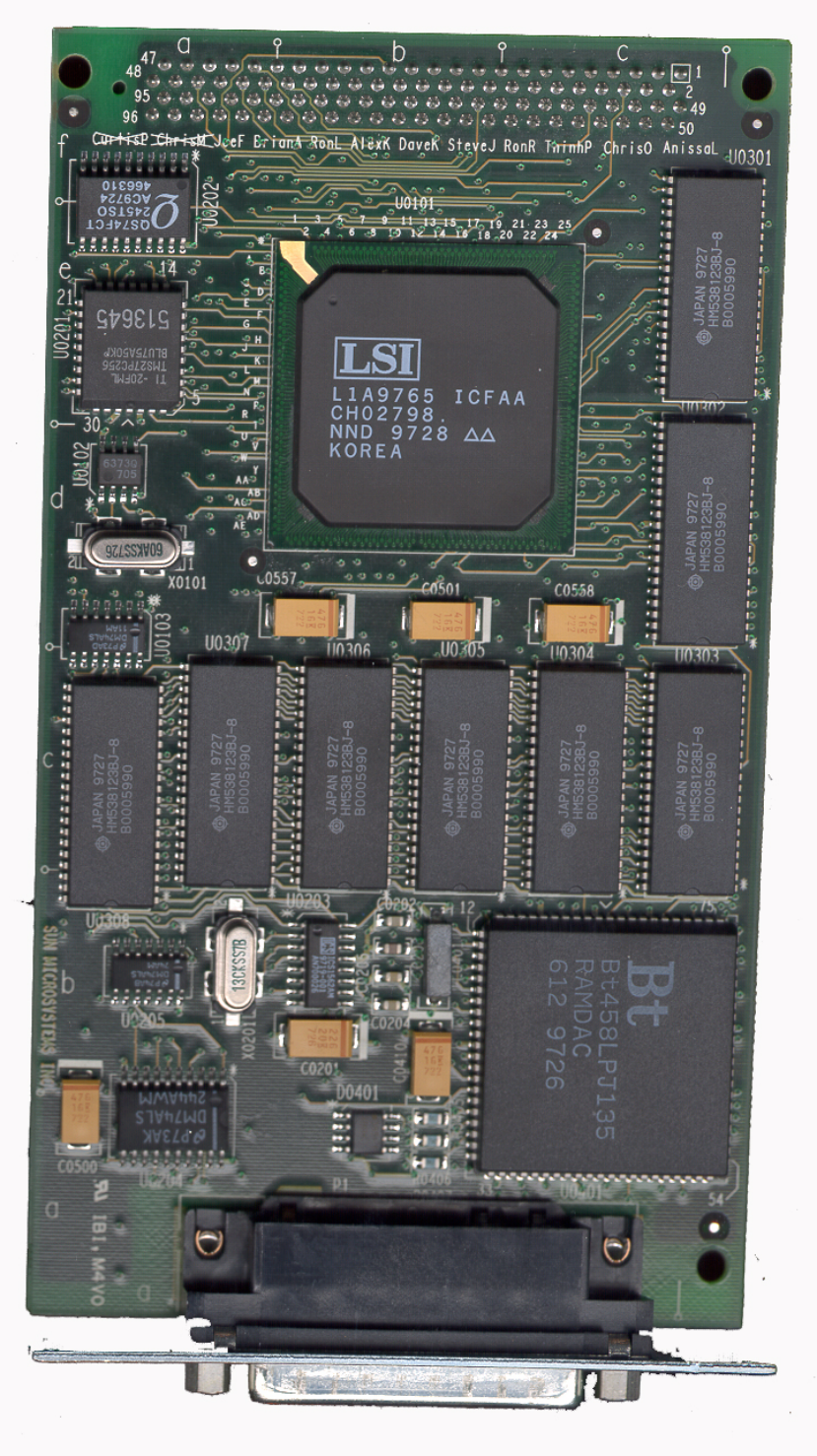


Abbildung 2.7: SBus Grafikkarte

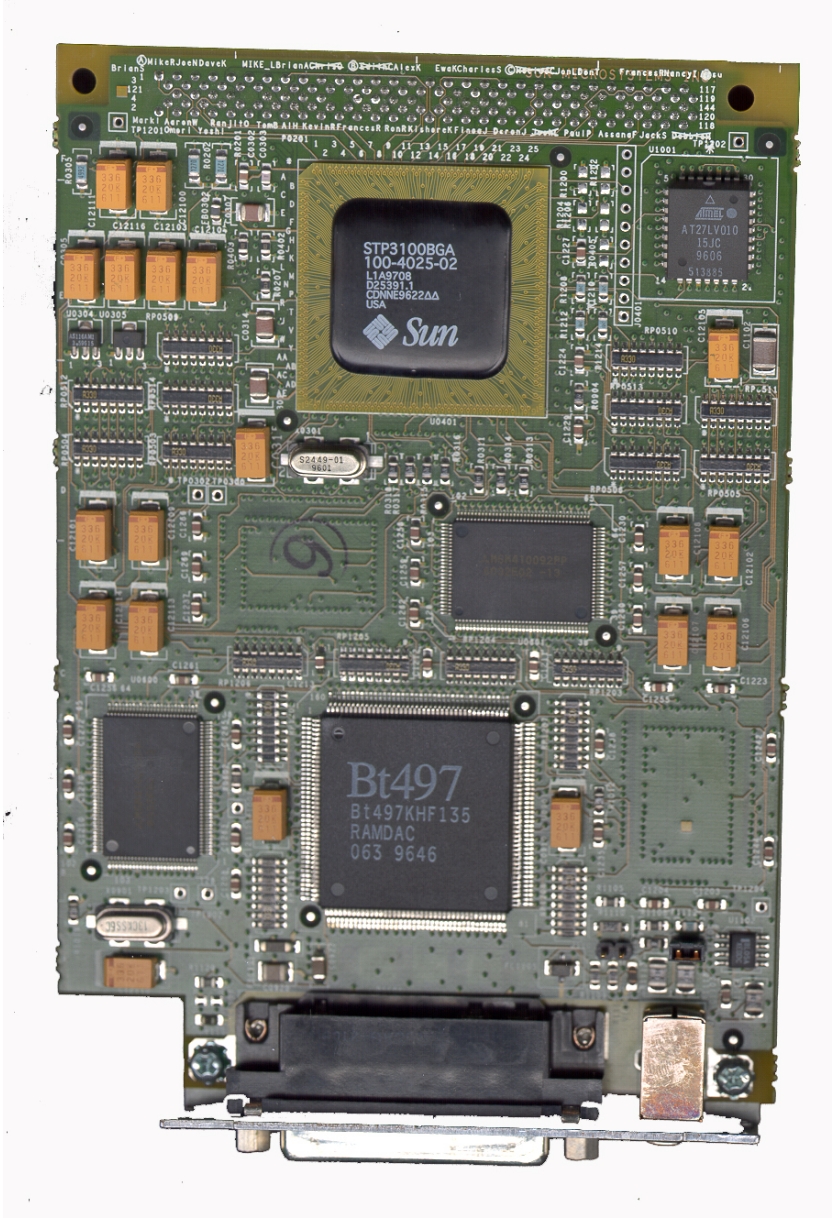


Abbildung 2.8: UPA Grafikkarte

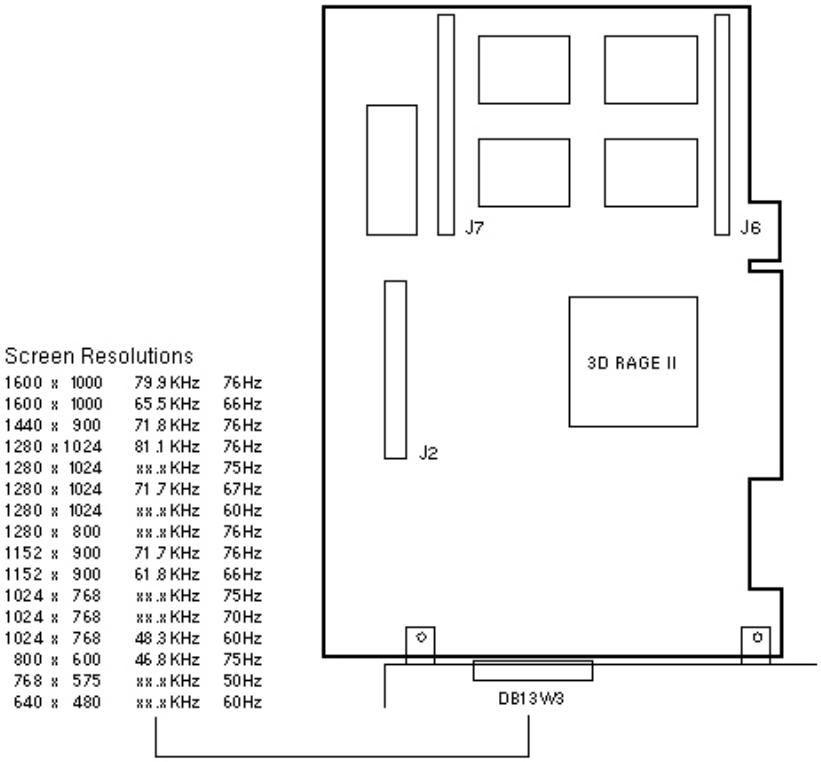


Abbildung 2.9: UPA Grafikkarte

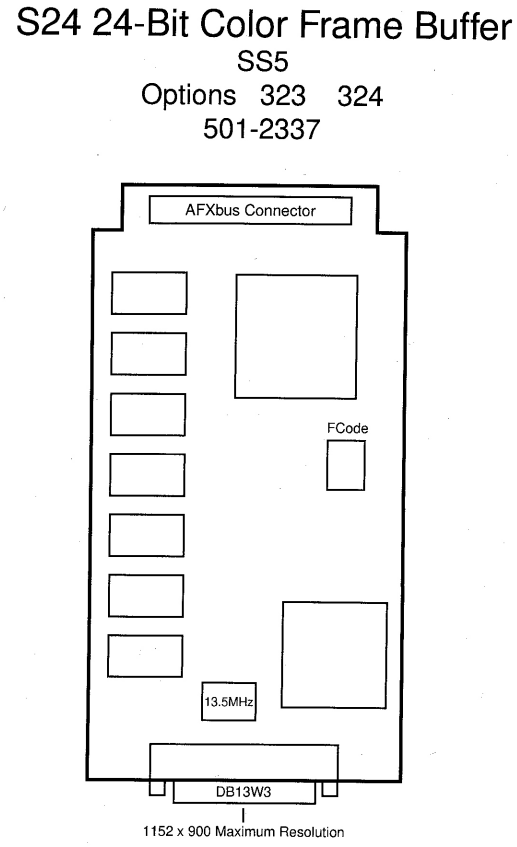


Abbildung 2.10: Grafikkarte AFX Bus

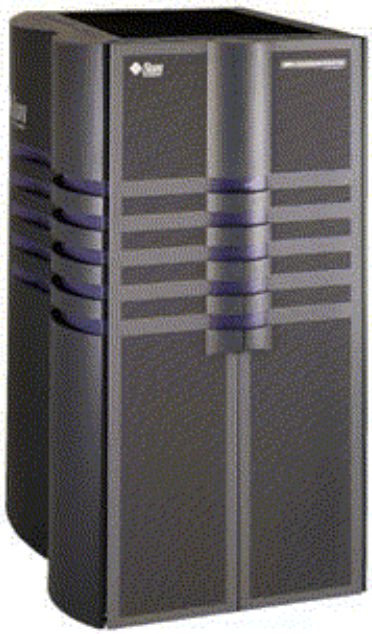


Abbildung 2.11: SUN Enterprise E10000 (E10K)

Welche Unterschiede bestehen zu PC Komponenten?

Wie schon erwähnt, passen die PC Komponenten nicht in ältere Sun Rechner, weil diese nur den SBus zur Erweiterung haben. Hier ein Vergleich der Schnittstellen:

Abbildung 2.12: SUN Enterprise E5000

2.2 Architektur

Welche verschiedenen Bussysteme zeichnen ein Sun System aus?

Die ersten Sun Rechner bis Ultra I haben folgende Busse:

- Der Prozessorbus. Er wird bei der Sun als MBus bezeichnet und arbeitet je nach Modell mit 40 oder 50 MHz
- Der SBus Er dient zur Erweiterung mit den oben genannten Karten und arbeitet mit 20 MHz.
- Der Speicherbus
- Der I/O Bus

Der Vorteil eines 64 Bit Systems liegt auf der Hand. Doppelt soviele Datenleitungen doppelt so viele Daten zur gleichen Zeit. Man kann also mit gutem Gewissen sagen, dass gleichgetaktete Systeme mit 64 Bit doppelt so schnell sind, wie die Systeme mit 32 Bit. Des weiteren ergibt sich noch ein Vorteil:

2^{32}=512 Mbyte max. adressierbarer Speicher
2^{64}=2Ebyte (ExaByte[3]) max. adressierbarer Speicher

Also es kann viel mehr Speicher adressiert werden als in einem 32 Bit System.

In welchen Verhältnis steht die Leistung von Sun Systemen zu ihrem Anschaffungspreis?

Abbildung 2.13: SUN Ultra 10

Da die Sun Systeme wesentlich teuerer sind als PC's mit scheinbar der selben Leistung, stellt man sich die Frage, ob sich die Anschaffung eines solchen Systems lohnt.
Betrachten wir die Angelegenheit mal von einer anderen Seite. Wie Ausfallsicher sind PC's? Genau dort sollte man die Entscheidung für oder gegen ein Sun System treffen. Denn was nützt ein Computersystem, welches man nicht unterbrechungsfrei durchlaufen lassen kann?
Ein Test hat ergeben, die durchschnittliche Laufzeit eines PC Systems beträgt 3 Tage die eines Unix Systems beträgt 110 Tage.
Man sollte deshalb genau überlegen, welche Systemsicherheit gegeben sein soll und nur so kann man entscheiden, ob sich die Anschaffung eines teueren Sun Systems lohnt.

[3]eine Trillion Byte

Abbildung 2.14: SUN Ultra 1

Fazit:

Es ist also keine Frage des Anschaffungspreises sonder eine Frage der Zuverlässigkeit.

Welche verschiedenen Systeme bietet Sun an?

Je nach Anwendungsgebiet bietet Sun von der Workstation bis zum High-End Server alles - und zwar skalierbar.

AUFGABEN

1. **Welche Aufgaben hat das NVRam in einem Sun System?**
2. **Wie funktioniert der SCSi Bus?**
3. **Welche Aufgabe hat der Open Boot Prom?**
4. **Leistung unterschiedlicher Sun Systeme**
5. **Welche Vorteile besitzt die 64 Bit Architektur der neuen Sun Systeme?**
6. **Öffnen Sie ein Gerät und ordnen Sie den Komponenten die entsprechenden Bezeichnungen zu!**
7. **Versuchen Sie, die verwendeten Bus - Systeme zu identifizieren!**

2.3 Installation neuer Hardware

Was muß bezüglich elektrostatischer Entladung beachtet werden?

Da viele empfindliche Bauteile auf Computerplatinen zu finden sind, ist es ratsam die Platinen gegen elektrostatische Entladungen zu schützen. Je nach Kleidung kann sich ein Mensch schon bei geringer Bewegung auf mehrere tausend Volt aufladen. Entläd sich die Spannung über eine Computerplatine so kann diese schon kaputt sein. Deshalb gibt es Armbänder, die über einen Widerstand ständig die gefährlichen Spannungen entladen.

Welche Ausrüstung wird zur Installation neuer Hardware in einem Sun System benötigt?

Um die Sun zu öffnen benötigt man lediglich einen Kreuzschlitzschraubendreher. Ebenfalls kann man diesen einsetzen um diverse Erweiterungskarten zu befestigen bzw. zu lösen.

Geht möglicher Support - Anspruch verloren?

Nein, nur bei grob fahrlässiger Behandlung oder bei Hardwareschäden, d.h. wenn z.B. die Leiterplatte zerbrochen wird.

2.4 Monitore

Welche Monitore sind für Sun Systeme sinnvoll?

Je nach Anwendung muß entschieden werden, welche Größe der Bildschirm hat. Als Minimum ist ein 17" Monitor zu empfehlen. Besser jedoch sind die 19" bzw 20" und 21" Monitore.
Server benötigen nicht unbedingt einen Monitor.

Wie kann ich alte Monitore in einem Sun System einsetzen?

Alte Monitore haben meistens einen Synchronisierten RGB Eingang. Dazu gibt es einen RGB auf 13W3 Adapter.

Was passiert mit einem Sun System, wenn Monitor oder Grafikkarte ausfallen?

Ein Sun System kann immer über eine serielle Schnittstelle mit einer anderen Sun verbunden werden. Dort kann man die Vorgänge überwachen und steuern.

AUFGABEN

1. **Erstellen Sie sich einen elektrostatisch korrekt abgesicherten Arbeitsplatz!**

2. **Bauen Sie einzelne Komponenten ein und aus.**

3. **Simulieren Sie einen Defekt an der Grafikkarte, in dem Sie diese vom System entfernen!**
   Was passiert? Versuchen Sie per Konsolenverbindung Informationen über den Status des Systems zu erhalten!

4. **Welche Auswirkung hat es, wenn bei defekter oder fehlender Grafikkarte die Tastatur und die Maus angeschlossen bleiben?**

5. **Versuchen Sie, eine Belegung des "13W3" Anschlusses im Internet ausfindig zu machen!**

Kapitel 3

Software

3.1 Geschichtliches

Im Jahre 1969 entwickelte Ken Thompson an den Bell Laboratories die erste Unix-Version auf einer PDP-7, noch vollkommen in Assembler geschrieben. Zu dieser Zeit schrieb Dennis Ritchie die Programmiersprache C, die auf Thompsons' B basierte. Bis 1971 wurde Unix dann nahezu komplett neu in C programmiert und auf die PDP-11 portiert, da diese Sprache Unix eine hohe Portablitiltät garantierte. Somit gelten beide Entwickler als die Väter dieses Betriebsystems.

Abbildung 3.1: Dennis Ritchie

Die grundlegenden Ideen dazu holten sie sich aus dem, zusammen mit weiteren Entwicklern an den Bell Laboratories konzipierten Multics, sowie CTSS vom MIT und GENIE von Berkeley. Diese Ideen waren, im Gegensatz zu denen der zu dieser Zeit bekannten Systemen, die Verfügbarkeit im Quellcode, die Implementierung in einer Hochsprache und die Möglichkeit für jeden Benutzer mehrere Prozesse zur gleichen Zeit laufen zu lassen. Die erste offizielle Version, die auch an Interessierte weitergegeben wurde, war Version 6. Version 7 wurde möglichst portabel gehalten und lief auch auf Interdate 8/32 und der damals aufkommenden VAX-Architektur.

Unix History

Zu diesem Zeitpunkt entwickelten verschiedene Gruppen an den vorhandenen Quellen weiter:

- Bell Unix Time-Sharing System 8th bis 10th Edition (1989), danach Plan 9 (von Unix eher unabhängig)

- AT&T Unix Support Group (USG, System III), danach System V (1983) welches AT&T nach einem Gerichtsbeschluss auch kommerziell vermarkten durfte. SystemV Release 2 wurde dann 1984 von dem USDL (Unix System Development Laboratory) entwickelt und weiterhin von AT&T vertrieben. System V.3 (1987) sowie V.4 (1989) kamen von ATTIS (AT&T Information Systems) und wurden durch die AT&T-Tochter USL (Unix System Laboratories) übernommen, welche 1993 an Novel verkauft wurde. Der Markenname "Unix" wurde an das X/Open Konsortium übergeben, welche damit das alleinige Recht besass, Standards festzulegen, die von Unix-Systemen erfüllt werden müssen, um den Namen "Unix" tragen zu dürfen. 1995 wurde dieses Recht an SCO (Santa Cruz Operating Systems) verkauft, welche das PC-Unix Xenix zusammen mit Microsoft entwickelten und es nach deren Ausstieg als SCO-Unix für Intel-basierte Rechner verkauften.

- Der University of California in Berkeley entsprang Ende der 70er Jahre die Berkeley Software Distributions, welche als BSD-Unix bekannt sind. BSD war lange Zeit das einzig lauffähige Betriebssystem auf VAX-Rechnern und erfreute sich bis 1983 auch auf anderen Plattformen grösster Beliebtheit. Ab der Version 4.2BSD, die zeitgleich mit dem Vermarkten von System V zur freien Verfügbar stand, kam es zu Lizenzstreitigkeiten zwischen Berkeley und den System V Besitzern, da beide Systeme auf den Version 7 Quellen beruhten, aber nur letzteren das Recht zur Weitergabe und -vermarktung zugesprochen wurde. Die Entwicklung von 4BSD wurde durch die Defence Advanced Research Projects Agency (DARPA) gefördert, um Unix um Netzwerkunterstützung zu erweitern, die im ARPA-Net (Grundlage des Internets) verwendet wurde und auch heute noch Grundlage für jedes Unix-Netzwerk bildet. Da die BSD-Quellen nur an Besitzer von "Unix Source Licence"

weitergegeben werden durfte, entschied man sich bei Berkeley alle lizenzbehafteten Quellen von grundauf neu zu schreiben. Bei 4.3BSD gab es die lizenzfreien NET-Releases, bestehend aus der aktuellen 4.3BSD-Release ohne die jeweiligen lizenzbehafteten Teile. Erstmals 4.4BSD-Lite (1994) stellt eine nahezu komplette und vollkommen lizenzfreie Unix-Version dar, welche als Quellcode und Binary an jedermann weitergegeben werden durfte. Auf 4.4BSD-Lite basieren die momentan verfügbaren, freien Betriebssysteme FreeBSD, NetBSD und OpenBSD sowie das kommerzielle BSDI. Teile aus der neuen Lite2-Version gelangen noch heute in diese BSD-Varianten.

Technische Entwicklungen flossen quer durch die verschiedenen Zweige, wodurch jedes Unix-Derivat von Gedanken der anderen profitierte. Workstation-Hersteller und Software-Häuser verkauften ihre jeweilige Version von Unix, die auf den (lizensierten) Quellen der o.g. Zweige beruhten unter eigenem Namen (HP/UX, Unixware, AIX, A/UX, IRIX, SunOS, Solaris, Domain/OS, Interactive, BSDI, OSF/1, Ultrix, DEC Unix, ...). Bekannte, nicht-Unix-basierende Systeme (OS/2, Windows 95, Windows/NT), wurden als Unix-Killer gehandelt, aber keins davon hat es bisher geschafft Unix aus den Marktsegmenten zu verdrängen, für die es von Anfang an gedacht war und für die es heute noch keine wirklich brauchbaren Alternativen gibt.

3.2 Multitasking/Multiuser

Eine grundlegende Eigenschaft von Unix ist dessen Multitaskingfähigkeit. Unix lebt von der Möglichkeit, mehrere Prozesse quasi-gleichzeitig (im Zeitscheibenverfahren) ablaufen zu lassen. Dies geschieht nach dem Prinzip des *präemptiven Multitaskings* bei dem einzig das Betriebssystem für die Vergabe von Rechenzeit an einzelne Programme zuständig ist und nicht wie beim *cooperativen Multitasking* die Programme selber entscheiden ob und wieviel an Prozessorkapazität sie beanspruchen, bzw. an andere Programme abgeben (z.B. Windows 3.x). Praktisch jede Aufgabe, jeder Dienst und was sonst noch alles zu einem Betriebsystem gehört, wird jeweils durch einen eigenen Prozess gelöst. Jeder Prozess bekommt vom Unix-Kernel[1] einen Adressbereich[2] zugewiesen, auf den nur er Zugriff hat. Benötigt der Prozess mehr Speicherplatz, so muss er ihn beim Kernel "beantragen". Zur Vergabe und Verwaltung der einzelnen Prozess-Adressbereiche ist einzig der Kernel zuständig. Dies hat den Vorteil, dass kein Prozess (z.B. ein Anwenderprogramm), ohne den Weg über den kontrollierenden Kernel zu nehmen, Zugriff auf den Speicherbereich eines anderen Prozesses hat. Gelöst wird dies durch die Verwendung verschiedener Speichersegmente: Das Program selber, das *Executable*, wird im Codesegment abgelegt, welches für den Prozess selber nur lesbar, nicht aber schreibbar ist. Dadurch wird vermieden, dass sich Programme während ihrer Laufzeit modifizieren können, welches in Bezug auf Systemsicherheit sehr wichtig ist. Im Datasegment des Prozesses werden alle anfallenden Daten des Programms verwaltet und nur der Prozess selber hat darauf Lese- und Schreibrechte. Beantragt der Prozess allerdings beim Kernel einen sog. *Shared-Memory* Bereich, so kann aus diesem auch von anderen Prozessen gelesen werden. Im Stacksegment eines Prozesses werden Daten wie Aufrufparameter, Rückgabewerte, Register und ähnliches zwischengelagert. Auch dieser Speicherbereich ist nur Les- und Schreibbar für den Prozess

[1] Der Kernel ist jenes Programm, welches beim Booten von Unix geladen wird. Seine Aufgabe ist es, alle Prozesse, den Speicher, angeschlossene Geräte und Ein/Ausgabe-Operationen, gleich welcher Art, zu verwalten.

[2] Der gesamte, zur Verfügung stehende Adressraum berechnet sich normalerweise aus physikalischem RAM + Swap-Bereich auf der Festplatte (oder im Netz).

selber, andere besitzen keine Rechte darauf. Programme, die durch Speicheroperationen andere Programme oder gar das komplette Betriebsystem (den Kernel) zum Absturz bringen, können durch dieses Speicherschutz-Konzept nicht vorkommen. Dies gewährt natürlich ein hohes Maß an Ausfallsicherheit und begründet die teilweise immens hohen Laufzeiten (Uptimes) von Unix-Systemen; 500 Tage oder mehr sind keine Seltenheit.
Dadurch, dass immer mehrere Prozesse "laufen", also einen Bereich im Speicher belegen und vom Kernel verwaltet werden, müssen gleichzeitige Zugriffe auf ein und dasselbe Gerät (Bildschrim, Schnittstellen, Netzwerkkarte, Festplatte, Tastatur, Maus, usw.) vermieden werden. Auch das ist Aufgabe des Kernels. Indem kein anderer Prozess ausser ihm Zugriff auf die Hardware-Resourcen hat und jeder Hardware-Zugriff eines Prozesses, z.B. das Schreiben auf die Festplatte oder das Lesen einer CD, letzlich vom Kernel getätigt wird, kann dieser auch konkurrierende Zugriffe vermeiden. Zu diesem Zweck ist jedes Gerät, selbst die Grafikkarte eines Rechners oder der "Speicher", als Datei im Verzeichnis /dev (zum Thema Filesystem siehe 3.3) abgebildet. Benötigt ein Prozess nun Zugriff auf ein Stück der Hardware, so liest oder schreibt er einfach in die stellvertretende Datei (z.B. /dev/tty1 für eine serielle Schnittstelle oder /dev/dsk für ein Disk Device [3]). Die eigentliche Ein/Ausgabe-Operation wird dann vom Kernel durchgeführt. Dies bedingt natürlich, das der Kernel über die Funktionsweise der Gerätschaften informiert ist. Das was unter DOS oder Windows als "Treiber" bekannt ist, findet man unter Unix direkt im Kernel, da ja nur hier alle hardwarenahen Operationen durchgeführt werden dürfen. Ein Nachteil dieser Funktionsweise ist, dass bei jedem neuangeschafften Gerät (z.B. neue Ethernetkarte) oder neugewünschten Funktion die der Kernel behandeln soll, dieser neu übersetzt werden muss. Da dies mitunter ziemlich zeitaufwendig sein kann, ist man dazu übergegangen einige Treiber-Funktionen in sogenannte *loadable kernel modules* auszulagern, welche dann bei Bedarf zur Laufzeit des Systems zum Kernel "hinzugelinkt" werden können. Um bei änderungen von Hardware-Informationen, wie IRQ oder Base-Adress, einer PC-Steckkarte, nicht jedesmal einen neuen Kernel compilieren zu müssen, versuchen einige PC-Unixe diese zu "proben" (auf

[3] Die Dateinamen für Device-Files unterscheiden sich je nach Unix-Derivat.

Verdacht alle Adressen durchgehen und schauen ob sich eine Karte meldet), z.B. Linux, oder einen Hardware-Konfigurations-Modus zwischen Laden und Ausführen des Kernels zu starten, wie z.B. FreeBSD, in dem man die gewünschten Einstellungen dann von Hand durchführen kann.

Mit der o.g. Fähigkeit des Multitaskings geht die Multiuserfähigkeit einher. Jeder Benutzer eines Unix-Systems muss diesem bekannt sein. Der Account (Zugangsberechtigung) wird durch den Loginnamen repräsentiert und besitzt ein Passwort. Durch die Möglichkeit von Unix, mehrere Prozesse gleichzeitig laufen zu lassen und die Unterscheidung zwischen verschiedenen Benutzern ist es möglich, dass mehrer Benutzer auch gleichzeitig mit dem System arbeiten und ihrerseits mehrere Prozesse laufen lassen können. Die Ein- und Ausgabe zu den Benutzer-Prozessen (z.B. dessen Shell, s. 3.9) kann z.B. auf der Console[4] (angeschlossener Monitor und Tastatur), seriellen Terminals (ASCII, vt100) oder Netzwerkverbindungen (*telnet, rlogin*) geschehen, aber auch aus einer Datei gelesen, bzw. in ein geschrieben werden, oder ganz unterdrückt werden. Das Betriebssystem muss gewährleisten, dass die Daten des Benutzers (seine Dateien, seine laufenden Prozesse) vor unberechtigten Zugriffen anderer Benutzer geschützt sind, bzw. nur soweit veröffentlicht werden, wie es der Benutzer möchte.

Zusammenfassend kann man daher drei wichtige Unterschiede eines Mehrbenutzerbetriebsystems gegenüber einem Einzelplatzsystems benennen:

- Die Anwenderprogramme können sich gegenseitig nicht stören
- Es müssen konkurrierende Hardwarezugriffe verhindert beziehungsweise verwaltet werden.
- Es müssen die privaten Daten der Benutzer geschützt werden.

[4]Solaris verwaltet mehrere virtuelle Consolen.

AUFGABEN

1. **Definieren Sie den Bergriff präemtives Multitasking!**
2. **Warum können sich Anwenderprogramme nicht gegenseitig stören?**

3.3 Dateisystem

Das Filesystem von Unix ist anders aufgebaut als man es von Windows her kennt. Es besitzt keine Laufwerke wie *C:*, *D:* usw. Es hat eine Baumstruktur. Wenn man Windows genauer betrachtet, kann man dort auch eine Baumstruktur erkennen, jedoch gibt es da keine "Wurzel" von der alles ausgeht. Die Wurzel bei der die Struktur im Unix System beginnt ist das Root-Directory. Von hier aus kann man per *cd* überall hin gelangen.

Überblick

Im Folgenden werden die wichtigsten Directories und ihre Funktion erklärt:

/	Der Slash "/" steht für das sogenannte Root-Directory. Das Root-Directory ist das Oberverzeichnis aller Verzeichnisse und damit auch der Einstiegspunkt in die Verzeichnishierarchie. Von diesem Verzeichnis aus sind alle Verzeichnisse des Dateibaumes durch Abstieg in der Dateihierarchie erreichbar.
/bin	In diesem Verzeichnis befinden sich Standardprogramme, die fest zum Bestand des Unix-Betriebssystems gehören.
/sbin	Dieses Verzeichnis enthält Programme, die zum Booten und Wiederherstellen des Betriebssystems gebraucht werden.
/dev	Das Verzeichnis */dev* und seine Unterverzeichnisse enthalten die Special Files für die Geräte. In dem Unterverzeichnis */dev/term* befinden sich die Dateien für Terminals, in */dev/dsk* die Dateien für Festplatten (genauer Partitionen der Festplatte/platten) und in */dev/mt* die Dateien für die Bandlaufwerke.

/etc	In diesem Verzeichnis befinden sich die Konfig-Dateien zur Systemadministration und -konfiguration (z.B. die Paßwortdatei oder Dateien für die Netzwerkkonfiguration).
/opt	Unterhalb von */opt* können Zusatzanwendungen installiert werden. Auf Solaris werden hier die Betriebssystemerweiterungen installiert.
/home	Die Unterverzeichnisse von /home sind zur Aufnahme der Daten der einzelnen Benutzer gedacht. Dabei hat jeder Benutzer ein eigenes Verzeichnis, sein sogenanntes Home Directory.
/spool	Dieses Verzeichnis und seine Unterverzeichnisse nehmen temporäre Dateien für den Druckerspooler*(/var/spool/lp*), den uucp-Spooler (*/var/spool/uucp)* und für den Mailspooler *(/var/spool/mqueue*) auf. Ein Spooler ist ein Warteschlangenverwalter. Solche Warteschlangen werden dann benutzt, wenn ein bestimmter Dienst nicht sofort verfügbar ist.
/tmp	In diesem Verzeichnis werden von Benutzern und Programmen temporäre Dateien abgelegt. Der Inhalt von /tmp wird beim Booten automatisch gelöscht.
/var	Dieser Unterbaum enthält die Programme und Dateien, die sich in den unterschiedlichsten Unix-Betriebssystemen unterscheiden. Das Unterverzeichnis */var/adm* enthält z.B. Protokolldateien für die Rechneraktivitäten und */var/mail* die Dateien für die elektronische Post.

/usr	Der unter */usr* befindliche Dateibaum enthält für die Benutzer relevante Dateien. Häufig befinden sich */usr* und seine Unterverzeichnisse auf einer eigenen Partition oder Festplatte. */usr/bin* und */usr/sbin* ernthalten die Standardkommandos und Benutzerprogramme, */usr/lib* die Libraries zur Programmentwicklung bzw. Shared Libraries und */usr/share/man* enthält in seinen Unterverzeichnissen die man-Pages.

Eine Datei ist eine Sammlung von Daten in strukturierter oder unstrukturierter Form, die unter logischen Namen zusammengefaßt werden. Unix kennt vier Dateiarten:

- Plain File
- Dateikataloge (Directories)
- Gerätedateien
- Pipes

Unix verwendet einen sehr allgemeinen Dateibegriff, der Benutzern zunächst ungewohnt erscheint.
Unter dem Begriff Plain File werden Datendateien, wie sie auch von anderen Systemen bekannt sind, aber auch Programme zusammengefaßt. Da Unix keine Erweiterungen wie .exe oder .bat zur Unterscheidung der Programme von den Datendateien fordert, ist die Einteilung in dieselbe Klasse von Dateien nicht weiter verwunderlich. Für die Unterscheidung dieser Plain Files gibt es das Kommando file. Das Kommando zeigt an, um was für ein File es sich handelt.

Zugriffsrechte

Jede Datei unter Unix verfügt über einen Satz von Zugriffsrechten. Diese Rechte legen fest, wer eine Datei lesen darf, wer sie verändern kann und wem es erlaubt ist, eine ausführbare Datei, also ein Programm oder ein Skript, zu starten. Die Zugriffsrechte stehen im direkten Zusammenhang mit dem

Multiuserkonzept, da sie für den Schutz der Daten der einzelnen Benutzer sorgen. Allerdings gibt es auf jedem Unix Betriebssystem mindestens einen Benutzer, für den die Zugriffsrechte keine Rolle spielen: Dies ist der Systemverwalter mit der Benutzerkennung root, der die Zugriffsrechte für jede Datei ändern darf.

Beispiel Zugriffsrechte:

```
-rw-r--r-- 1 einstein admin        665 Mar 11 22:58 einkaufszettel

-rw-r--r-- 1 einstein admin        665 Mar 11 22:58 milchpreis

drwxr-xr-x 2 einstein admin       1024 Mar 11 22:58 einkauf1999

drwxr-xr-x 2 einstein admin       1024 Mar 11 22:58 einkauf2000

lrwxr-xr-x 1 einstein admin       2048 Mar 11 22:58 heute
```

Kennzeichnung des Dateityps:

Der Bindestrich (eigentlich ein Minuszeichen) kennzeichnet ein Plain File. Ob es sich dabei um ein Programm handeln könnte, läßt sich nur an den Zugriffsrechten ablesen (x= Ausführungsrecht $\cong$ Programm).
Das "d" steht für Directory, kennzeichnet also ein Verzeichnis.
Das "l" zeigt einen Symbolic Link (einen Verweis auf eine andere Datei) an.

Die Zugriffsrechte:

r steht für das Vorhandensein des Leserechtes (read)
w steht für das gesetzte Schreibrecht (write)
x steht für das Ausführungsrecht (execute) und
"-" steht für das Fehlen eines Zugriffsrechts.

AUFGABEN

1. **Lesen Sie die Manpages zu *mkdir*, *rm*, *chmod* und *ls*! (siehe auch 3.6)**

2. **Erstellen Sie direkt im Root-Verzeichnis ein Verzeichnis */mirror*!**
   Versuchen Sie, mit möglichst wenig Befehlen, die Verzeichnisstruktur unter / nach */mirror* abzubilden!

3. **Ändern Sie die Zugriffsrechte von */mirror* und testen Sie die Auswirkung!**

3.4 An- und Abmelden

An einem Unix System muß man sich mit einem Namen und einem Paßwort anmelden. Meist werden die Workstations, an denen man arbeiten will, von einem Server verwaltet.
Es gibt unterschiedliche Arten, sich an einer Sun anzumelden. Zum einen kann das graphisch geschehen oder auch per Terminal.

CDE oder OpenWindows

CDE oder OpenWindows? Diese Frage stellt sich beim Arbeiten mit Solaris. Jedoch ist die Benutzung dieser beiden Oberflächen Geschmackssache. Bei Solaris ist es auch möglich fast alles auf der Kommandoebene zu machen, jedoch braucht man für bestimmte Anwendungen doch ein sogenanntes X-Window System. Nur eines der vielen Anwendungen ist der Netscape. Jedem Benutzer ist es selbst überlassen, welches X-Window System er benutzt. Der eine bevorzugt OpenWindows und ein anderer CDE. Implementieren kann man auf Solaris auch KDE. Jedoch sind nur standardmäßig die beiden anderen dabei.

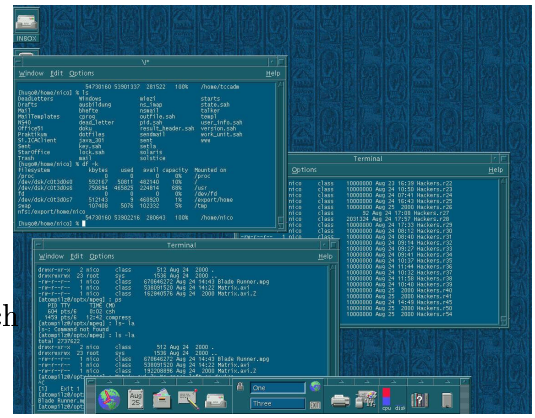


Abbildung 3.2: Das CDE (Common Desktop Environment)

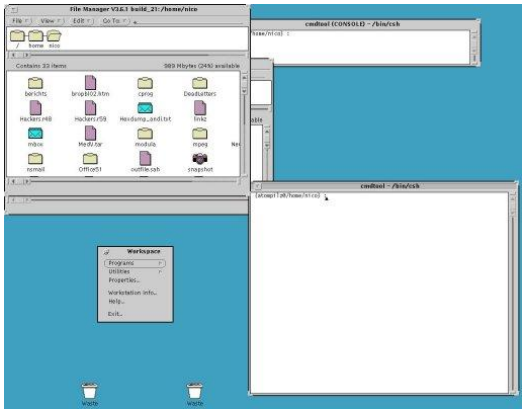


Abbildung 3.3: Der OpenWindows Desktop

3.5 Usernamen und Passwörter

Unter Unix ist es unerlässlich, sich den Kopf über sichere Passwörter zu zerbrechen. Mittlerweile ist es ganz einfach möglich, mit Hilfe eines logistisch gut strukturierten und leistungsstarken "Brute - Force[5]" Angriffes, viele Passwörter in kurzer Zeit auszuprobieren.

Diese Brute Force Angriffe werden meistens noch mit einer Dictionary Attack[6] kombiniert. Somit bleiben einem Benutzer nicht mehr so viele Möglichkeiten.

Um ein halbwegs sicheres Passwort zu bekommen, sollte man auch Sonderzeichen und Zahlen verwenden.

[5]Name für das systematische "Durchprobieren" aller Möglichkeiten eines zur Lösung eines Problems.

[6]Das Wörterbuch einer oder mehrerer Sprachen wird systematisch durchprobiert. Wort - Zahlen - Kombinationen werden sehr schnell erkannt.

AUFGABEN

1. **Loggen Sie sich einmal mit der Benutzeroberfläche *CDE* (Common Desktop Environment) und dann mit der Oberfläche *OpenWindows ein*!**
   Versuchen Sie unter beiden Oberflächen ein Terminal zu starten!

2. **Überlegen Sie sich ein Passwort, was nicht durch *Brute-Force* oder *Dictionary Attacks* ausspioniert werden könnte!**

3.6 Kommandos

Egal ob man im CDE oder im OpenWindows arbeitet, man benutzt beispielsweise zur Administration immer ein Terminal. Im Folgenden werden einige der wichtigsten Kommandos beschrieben, die man braucht, um sich in der Solarisumgebung zurechtzufinden.

Übersicht über die wichtigsten Unix Kommandos

man [Optionen] [Titel]	Zeigt Informationen aus den auf der Maschine gespeicherten Referenzhandbüchern. Ein Titel ist in der Regel die Bezeichnung für einen Befehl.
cd [dir]	Wechselt in das dahinter angegebene Verzeichnis. cd ist ein eingebauter Shell-Befehl . Bsp: cd.. ⇓ wechselt in das darunterliegende Verzeichnis. cd / ⇓ wechselt in das Root-Directory

ls [Option] [Namen]	Listet den Verzeichnisinhalt auf. Werden keine Namen angegeben, werden die Dateien im aktuellen Verzeichnis gelistet. Namen dürfen Metazeichen enthalten. Die Optionen erlauben die Ausgabe einer Vielzahl von Informationen in unterschiedlichen Anzeigeformaten. Die Option -l liefert die lange Ausgabeform mit einem Eintrag pro Zeile. -a listet alle Dateien, einschließlich der normalerweise versteckten Dateien. Die Option -c listet Dateien entsprechend der Erzeugungs-/ Modifikationszeit auf.
mkdir [Name]	Legt das Verzeichnis mit dem darauffolgenden Namen an.
rm [Optionen] [Dateien]	Löscht eine Datei. Mit Optionen kann man auch ganze Verzeichnisse löschen. Die Option -f Löscht schreibgeschützte Dateien ohne Bestätigung. -i Fragt mit y oder mit n die Bestätigung ab. Die Option -r Löscht dieses Verzeichnis mit dem gesamten Inhalt und allen Unterverzeichnissen, wenn die Datei ein Verzeichnis ist. **Warnung! Diese Option kann gefährlich werden!!!!!**

rmdir [Optionen] [Verzeichnisse]	Löscht das bezeichnete Verzeichnis, jedoch nur den Namen und nicht den Inhalt.
pwd	Gibt den vollständigen Pfadnamen des aktuellen Verzeichnisses aus.
ps [Optionen]	Auflistung aller aktiven Prozesse. Mit Optionen gibt es eine Vielzahl von Formaten. -e Listet alle Prozesse auf. -f Erzeugt eine vollständige Liste. -l Erzeugt eine umfangreiche (lange) Liste
cp [Optionen] [Dateien] [Verzeichnis] Quelle Ziel	Kopiert die Dateien in das hinten angegebene Verzeichnis. Ist eine Zieldatei vorhanden, wird diese überschrieben. -i Fragt nach einer Bestätigung (y oder n) bevor eine vorhandene Datei überschrieben wird. -r Kopiert rekursiv ein Verzeichnis, dessen Dateien und Unterverzeichnisse in ein Zielverzeichnis und dupliziert dabei die Baumstruktur
hostname	Gibt den Rechnernamen aus, auf dem man sich gerade befindet.
id	Gibt die eigene ID und die Gruppe mit ID aus.
who	Listet alle User auf, welche sich auf dem Rechner befinden.

chmod	ist der Befehl für das Ändern von Rechten nach Oktal-System. Bsp: chmod 755 egon
chown	mit diesem Befehl kann man den Eigentümer des Verzeichnisses oder des Files ändern.
which [Name]	which überprüft die eingetragenen Pfade, ob dort das Programm oder Skript zu finden ist.
uname	Gibt an, auf welchen Rechner man sich befindet.

3.7 Der Editor *vi*

Übersicht über einige wichtige Kommandos

^f	Eine Terminalseite vor springen
^b	Eine Terminalseite zurück spingen
0	Anfang der Zeile springen
$	Zum Ende der Zeile springen
w	Ein Wort vor
b	Ein Wort zurück
1G	Zum Anfang der Datei springen
G	Zum Ende der Datei springen

Kommandos zum Text löschen, andern kopieren löschen

i	Text einfügen vor dem Cursor		I	Text einfügen vor dem Zeilenanfang
a	Text anhängen hinter dem Cursor		A	Text anhängen hinter dem Zeilenende
o	Zeile einfügen unter der aktuellen Seite		O	Zeile einfügen über der aktuellen Zeile
r	Ersetze ein Zeichen		R	Überschreiben ab Cursorposition
cw	Ändern des Wortes		C	Ändern bis Zeilenende
Y	Yank (kopieren) einer Zeile		nY	Kopieren von n Zeilen
P	Einfügen des Yank-Buffers vor einer Zeile		p	Einfügen des Buffers hinter der aktuellen Zeile
x	Zeichen löschen hinter dem Cursor		X	Zeichen löschen vor dem Cursor

dw	Wort löschen		dd	Löschen der ganzen Zeile
:5,$d	Zeile 5 bis Dateiende löschen			
:g/string/d	Alle Zeichen löschen, die String enthalen			

/string	Suchen nach string
/^...$	Suche nach Zeilenanfang/ Zeilenende
n	Weitersuchen vorwärts
N	Weitersuchen rückwärts
s/Katze/Maus/g	String ersetzen in ganzer Zeile

File Kommandos

:rw	Datei schreiben		:w!	Datei überschreiben
:q	Datei verlassen		:p!	Datei verlassen trotz Änderungen
:wq	Datei schreiben und verlassen		ZZ	Datei schreiben und verlassen
:e file	Editiere neue Datei		:e!	Reeditiere aktuelle Datei und vergiß Änderungen
:r file	Einfügen von Datei file unterhalb des Cursors		:r !ls	Einfügen des Outputs von Kommando "ls"
:f file	Umbenennen der aktuellen Datei		:f	Aktuellen Dateinamen anzeigen

Verschiedenes

u	Letztes Kommando rückgängig machen

%	Springe zur passenden Klammer
:set nu	Zeilennummerierung einschalten
:set nonu	Ausschalten der Zeilennummerierung
:set tabstop=4	Tabulatorenbreite auf 4 Zeichen setzen

3.8 Benutzerverwaltung

Da jeder Benutzer, im Folgenden auch User genannt, dem System bekannt sein muss, werden in einer Datei mit dem Namen *passwd* im Verzeichnis */etc* alle benötigten Benutzer-Daten abgelegt. Am Beispiel eines Eintrages in dieser Datei möchte ich diese Daten erläutern. */etc/passwd* ist Zeilenweise aufgebaut, d.h. für jeden Benutzer gibt es genau eine Zeile. Die einzelnen Datenfelder sind durch Doppelpunkte getrennt:

```
einstein:Hg98bga1Kq7cgE:2004:2000:Dr. Basti:/home/gast/einstein:/bin/tcsh
```

- Das erste Feld gibt den Usernamen an, in diesem Fall "einstein", welcher i.d.R. nicht länger als acht Zeichen sein darf.

- Das zweite Feld enthält eine verschlüsselte[7] Form des Passworts für diesen Benutzer, welches er mit dem Befehl *passwd* jeder Zeit ändern kann. In letzter Zeit trifft man aber auch immer mehr *Shadow-Password* Systeme an, bei denen der Passwort-String in einer eigenen, nur für den Systemverwalter lesbaren Datei (z.B. */etc/master.passwd*, /etc/shadow) gespeichert wird und in der */etc/passwd* nur ein Platzhalter im zweiten Feld steht. Dadurch wird es Benutzern des Systems unmöglich gemacht, den verschlüsselten Passwort-Eintrag zu lesen, um mit diesem dann z.B. einen Crack-Versuch[8] zu unternehmen.

- Im dritten Feld wird die UserID verwaltet. Diese ist fest mit dem Benutzernamen verknüpft und dient der eindeutigen Zuordnung[9] von Dateien und Prozessen zum Benutzer auf diesem System oder im Netzwerk.

- Das vierte Feld stellt die GroupID dar. Dies ist die Gruppe, der der User nach dem Einloggen angehört.

[7] Meist nach dem DES-Algorithmus, eine Art "Falltür"-Algorithmus. Alternativen sind z.B. MD5 oder Blowfish.

[8] Mit Hilfe von Crack-Programmen, Wörterbuch-Dateien und genügend Rechenleistung lassen sich einfach gewählte Passwörter entschlüsseln.

[9] Zur Verwaltung von Zugehörigkeiten einer Datei oder eines Prozesses wird einzig die UID bzw. GID verwendet. Ein Auflösung zum Namen geschieht über die *passwd*-Datei. z.B. wird bei NFS nur die UID/GID übertragen.

- Im fünften, dem sogenannten GECOS-Feld, können prinzipiell beliebige Daten zum Benutzer gespeichert werden, allerdings gilt es als Standard dort den vollen Vor- und Zunamen und bei Interesse noch Adresse und Telefonnummer abzulegen. Diese Informationen werden durch Komma getrennt und z.B. von Mailprogrammen ausgewertet. Der Benutzer kann diese Daten mit dem Befehl *chfn* modifizieren.

- Das sechste Feld gibt das Home-Verzeichnis des Benutzers an. In diesem Verzeichnis befindet sich der User direkt nach dem Einloggen (Bekanntmachen mit dem System) und besitzt normalerweise Schreibrechte dafür. Im Home-Verzeichnis eines Benutzers werden z.B. die meisten Konfigurations-Dateien für die von ihm verwendeten Programme abgelegt und nur hier hat er i.d.R. uneingeschränkte Schreibrechte und kann sich seine eigene, private Verzeichnisstruktur zur Ablage eigener Dateien schaffen.

- Das letzte Feld bezeichnet das Programm, welches direkt nach dem Einloggen des Users gestartet wird, im Normalfall ist dies eine Shell, dann Loginshell genannt. Ist dieses Feld leer, so geht das System von dem Programm */bin/sh* aus. Geändert werden kann dieses Feld durch den Befehl *chsh*, dafür muss das Programm aber vorher in der Datei */etc/shells* vom Systemverwalter als mögliche Loginshell berechtigt werden.

In neueren UNIX Arten werden noch drei weitere Felder (ein meist noch unbenutztes namens "class", eins zur Angabe wie lang das Passwort gültig sein wird und wann es geändert werden muss und eins zur Angabe wann der Account ungültig wird) verwendet. Diese befinden sich zwischen dem GID- und dem GECOS-Feld und sind vom Benutzer nicht einsehbar (in master.passwd).

Der erste Eintrag in der *passwd*-Datei bezeichnet den sogenannten Superuser, Accountname "root". Seine UID und GID ist 0 und er besitzt uneingeschränkte Lese- und Schreibrechte auf jede lokale Datei und das Recht Prozesse anderer Benutzer zu manipulieren. Sein Passwort ist sozusagen der "Schlüssel"

des Unix-Rechners, mit ihm kann alles am System modifiziert werden.

Unix kann mehrere Benutzer zu Benutzergruppen zusammenfassen und Zugriffsrechte für Mitglieder einer solchen Gruppe vergeben. Die Gruppen werden in der Datei */etc/group* verwaltet. Der Aufbau ist ähnlich der der *passwd*, z.B.:

```
admin:*:30:floriang,danielo
```

Das erste Feld bezeichnet den Gruppennamen, fest verknüpft wieder mit dem dritten Feld, dem der GroupID. Dazwischen kann ein (verschlüsseltes) Passwort stehen, wird aber i.d.R. nicht verwendet und deshalb ein "*" als "kein mögliches Passwort" verwendet. Im letzten Feld werden, durch Komma getrennt, alle Mitglieder der Gruppe aufgezählt. Mit dem Befehl *id* kann man überprüfen in welchen Gruppen man Mitglied ist:

```
einstein@ameise1:~> id
uid=2004(einstein) gid=2000(admin) groups=2000(admin),0(wheel),30(skiclub)
```

In diesem Fall ist die UID 2004, aufgelöst als "einstein", die Login-Group ist 2000, aufgelöst als "admin". Desweiteren gehört der User noch der Gruppe 0 ("wheel"[10]) und der SkiClub-Gruppe (GID=30) an.

In Netzwerken, in denen sich jeder Benutzer auf jedem Rechner einloggen können soll, empfiehlt es sich die *passwd*-Datei netzweit zu verteilen. Dies kann z.B. durch regelmässiges kopieren (z.B. mit dem *rdist* Kommando) der Datei auf alle Rechner geschehen oder durch den Einsatz eines Network-Information-Systems (NIS). Bei diesem wird die *passwd*- und ebenso die *group*-Datei (oder auch weitere) von einem zentralen NIS-Server verwaltet. Auf ihm läuft ein Prozess (*ypserv*), welcher auf Anfragen von Prozessen auf den NIS-Clients (*ypbind*) hin, die gewünschten Dateien zur Verfügung stellt. Auf den Clients wird am Ende der *passwd*- bzw. *group*-Datei ein Eintrag mit "+" beginnend angefügt, welcher darauf hinweist, dass hier die Datei vom NIS-Server "angedacht" wird. Dem NIS-Client Prozess *ypbind* muss

[10]Nur Mitgliedern dieser Gruppe ist es gestattet mit dem Befehl *su(1)* Root-Rechte zu erhalten.

dann nur noch mitgeteilt werden, welcher Rechner im Netz für ihn als Server zuständig ist und welcher NIS-Domain[11] er angehört.

[11]Zur Verwaltung verschiedener Netzbereiche in denen verschiedene Dateien gleichen Typs verteilt werden, trennt NIS zwischen den NIS-Domains.

3.9 Shells

Die Shell ist der Prozess welcher die Benutzereingaben entgegennimmt und die vom Benutzer gewünschten Befehle ausführt, ähnlich der "`COMMAND.COM`" unter DOS. Sie wird üblicherweise direkt nach dem Einloggen[12] gestartet. Die typischen Shells beinhalten in der Regel eigene Programmiersprachen, die Schleifenkonstrukte und Funktionen beherrschen. Man unterscheidet im groben drei Klassen von Shells, die Bourne-Shells, die C-Shells und alle übrigen. Die Gruppe der Bourne-Shells umfasst die Bourne-Shell (sh), die Korn-Shell (ksh), die GNU Bourne-Again Shell (bash) und die Z-Shell (zsh). Diese Gruppe versteht (mehr oder weniger) die gleiche Syntax an Befehlen, deshalb ist es besonders leicht zwischen ihnen hin und her zu wechseln. Die zweite bekannte Gruppe der Shells umfasst die TC-Shell (tcsh) und die C-Shell (csh). Sie unterscheiden sich von den Bourne-shells durch eine andere, mehr an die Programmiersprache C angelehnte, Syntax, welche sie inkompatibel zu den Bourne-Shells macht. Zuletzt gibt es auch noch jede Menge Shells, die nicht in die o.g. Gruppen passen, z.b. die Plan9-Shell (rc), die Key-Shell (keysh) und viele andere, die aber alle meistens für irgend einen speziellen Zweck und nicht als allgemeine Shell zu empfehlen sind. Um die Wahl der richtigen Shell zu erleichtern hier ein kleiner Überblick:

- `sh`
  Die sh ist sozusagen die Ur-Shell, man wird sie auf jedem Unix-System als /bin/sh finden. Sie ist als User-Shell nicht zu empfehlen, da sie kaum Editiermöglichkeiten der Eingabezeile bietet. Allerdings wird sie aufgrund ihrer hohen Verfügbarkeit gerne für Shell-Skripten verwendet.

- `csh`
  Die csh ist die "erste" benutzerfreundlichere Shell gewesen, und deshalb auch auf vielen Systemen unter /bin/csh zu finden, allerdings hat sie viele BUG's[13] und ist wie alle C-Shells nicht kompatibel zur sh. Darum kann ich sie nicht empfehlen. [14]

[12] siehe auch 3.4
[13] Bug's sind Fehler in einem Programm
[14] http://late5.e-technik.uni-erlangen.de/Software-Doku/csh-harmful.html

- `tcsh` Die tcsh ist die Weiterentwicklung der csh, und deshalb zu dieser kompatibel. Sie ist schon in der Grundeinstellung eine der benutzerfreundlichsten Shells, und wird deshalb auch von den mit Abstand meisten Benutzern bevorzugt. Allerdings ist die mangelnde Bourne-Shell Kompatibilität ein Manko wenn man ab und zu auch Shell-Skripten schreiben will, denn wer lernt schon gerne doppelt so viel ?

- `ksh` Wieder eine Shell aus der Bourne-Shell-Reihe, die auch recht benutzerfreundlich ist und einem die Wahl zwischen einem Emacs- und einem vi-kompatiblen Editiermodus bietet. Sie ist eine der wenigen Shells, die mehrzeilige Kommandos editieren und in der History verarbeiten kann.

- `bash` Die Neuimplementation der Bourne-Shell von GNU. Sie ist auch in der Grund-Einstellung recht benutzerfreundlich, bietet aber nicht so viele Konfigurationsmöglichkeiten. Sie ist mit der tcsh eine der beliebtesten Shells.

- `zsh` Die zsh bietet alles was die vorhergehenden Shells auch bieten (insbesondere ein vernünftiges multi-line editing (ähnlich der ksh) und eine programmierbare Completition (ähnlich der tcsh)) allerdings ist es ein höherer Aufwand die Features zu lernen und zu konfigurieren, so dass sie den eigenen Wünschen entsprechen.

Nun etwas über die Benutzung einer typischen Bourne-Shell:

kommando

Falls "kommando" ein Alias oder eine Shell-Funktion ist, wird dieses intern ausgeführt. Ansonsten wird kommando im Pfad (`$PATH`)[15] gesucht und ausgeführt, (Ausführbare Programme werden unter UNIX mit einem x-Bit gekennzeichnet und nicht mit einer Endung (.com, .exe)). Die Trennung zwischen einem Befehl und dem nächsten erfolgt entweder durch den Beginn einer neuen Zeile oder durch ein" ;".
Beispiel: `ls -l ; who`

[15]Im Gegensatz zu DOS wird . nicht implitzit durchsucht

Normalerweise wird ein Befehl von der Tastatur eingelesen und die Daten werden auf dem Bildschirm ausgegeben. Dies kann man natürlich auch ändern:

kommando < datei1 >datei2 2>datei3

Liest die Eingabe für den Prozess "kommando" aus datei1 und schreibt das Ergebnis in datei2. Etwaige Fehlermeldungen landen in datei3.[16]
Beispiel:
`grep Milch <einkaufszettel; uptime > 8.April; make 2>errors`

kommando<<Marke

Liest die Eingabe für "kommando" aus der selben Quelle aus der die Befehle gelesen werden, bis zur Endemarkierung **Marke**:

```
mail sec@42.org <<ende
Hi,
dies ist meine testmail
ende
```

Man kann natürlich auch die Ausgabe eines Prozesses direkt als Eingabe für den nächsten Prozess nehmen. Dies ist eine sogenante "Pipeline":

kommando1 | kommando2 | kommando3

Verwendet die Ausgabe von kommando1 direkt als Eingabe für kommando2 u.s.w.
Beispiel: `ls -l | sort -n +4 | more`

‘kommando‘

Die sogenannten "Backticks" ersetzen den Platz in der Befehlszeile durch die Ausgabe des Befehls.
Beispiel: `finger ‘whoami‘`

[16] In Wirklichkeit wird lediglich Filedescriptor Nr. 2 in die Datei umgelenkt, per Konvention sollte ein Programm Fehlermeldungen aber dort ausgeben, was dann zum gewünschten Effekt führt.

Filename globbing

Die Shell verwendet gewisse Zeichen um Filenamen zu "erzeugen"[17]
`*` steht für beliebig viele (auch garkeine) Zeichen, **nicht** jedoch für einen Punkt am Anfang eines Filenamens.
`?` steht für genau ein beliebiges Zeichen, wiederum jedoch **nicht** für einen Punkt, wenn es am Anfang eines Filenamens steht.
`[abc]` steht für genau eins der Zeichen zwischen den eckigen Klammern.
`[a-q]` für genau ein Zeichen aus dem Bereich.
`[!...]` für genau ein Zeichen das **nicht** aus ... ist
Beispiel: `ls einkauf*` oder `lp [1-3].April`

Variablen

Die Shell verwaltet sogenannte "Variablen" die beliebige Werte (Zahlen, Strings) annehmen können. Man unterscheidet 2 Arten von Variablen, normale und sog. Environment-Variablen. Erstere sind nur der Shell selbst bekannt, wobei letzere auch an von der Shell gestartete Prozesse weitergegeben werden. Eine normale Variable wird zu einer Environment-Variablen in dem sie mit dem `export`-Befehl als solche markiert wird.
Beispiel: `DISPLAY=pcinfo7:0;export DISPLAY;xterm`

Die Shell selber expandiert Variablen zu ihren werten, wenn man ihnen ein $ voranstellt: `eins=zwei; echo eins: $eins`
Man kann allerdings eine Variable gezielt an genau ein Programm übergeben, indem man die Definition durch ein Leerzeichen getrennt vor den Programmaufruf stellt
Beispiel: `NNTPSERVER=news.lrz-muenchen.de tin`

Die Shell stellt uns einige vordefinierte Variablen zu Verfügung:

[17] Im Gegensatz zu DOS, wo jede Anwendung selbst dafür verantwortlich ist, diese Zeichen auszuwerten.

`$SHELL`	Pfad zur Loginshell
`$HOME`	Home-Directory
`$USER`	Username
`$?`	Exit-Code des zuletzt ausgeführten Prozesses
`$$`	Prozess-Id der Shell
`$!`	Prozess-Id des zuletzt im Hintergrund ausgeführten Prozesses

Quoting

Es gibt verschiedene Möglichkeiten die Expansion von Variablen und das Filename-globbing zu verhindern, dies nennt man Quoting. Wenn ein Text von "single quotes" `'` begrenzt wird, findet darin keinerlei Umwandlung durch die Shell statt, selbst Zeilenumbrüche können auf diese Weise eingegeben werden: `echo 'eink*' '$eins'`
Verwendet man zur Begrenzung "double quotes" `"`, so wird das Filenameglobbing unterbunden, alles andere (wie Variablenexpansion, Auswerten von Backticks ...) wird weiterhin durchgeführt: `echo "*" "`id -u`"`
Der Backslash `\` verhindert lediglich die Sonderbehandlung des nachfolgenden Zeichens: `echo \'eink*\'`

Job control

Natürlich kann man auch von einer Shell aus mehrere Prozsse starten, und sogar verwalten. Um einen Befehl im "Hintergrund" auszuführen schliesst man ihn einfach mit einem `&` statt mit einem `;` ab: `xterm &`
Um einen bereits normal im Vordergrund laufenden Prozess in den Hintergrund zu befördern muss man ihn zuerst "suspenden" (anhalten) um wieder zur Shell zurückzukehren. Dies erreicht man in der Regel mit `^Z`[18] in der Shell in der Prozess gestartet wurde. Man kann allerdings auch aus einer anderen Shell heraus dem Prozess ein STOP-Signal schicken: `kill -STOP <pid>`
Die Shell selber bietet einem dann die Möglichkeit mit `jobs` die laufenden Prozesse einzusehen, und sie dann mit `fg` oder `bg` in der Vorder- bzw. Hintergrund zu verschieben. Zu diesem Zweck nummeriert die

[18] Dieses Kontroll-Zeichen kann man mit dem Befehl `stty(1)` beliebig umstellen

Shell die Prozesse durch, diese Zahlen können an stelle der PID's fuer fg,bg und kill verwendet werden, wenn man ihnen ein `%` voran stellt. die Sobald ein im Hintergrund laufender Prozess etwas von der Tastatur lesen[19] will wird er jedoch angehalten (um Konflikte zu vermeiden) was die Shell mit einem `<pid> Stopped (input)` anmerkt und wartet bis man ihn wieder in den Vordergrund holt.

Funktionen

Man kann beliebige Shell-stückchen als Funktion deklarieren, indem man das Stück in `{}` fasst und ein `funktionsname()` voranstellt. Man kann innerhalb der Funktion auf die Aufrufparameter mittels `$1 $2 ...` oder mit `$@` und `$*` zugreifen: (die beiden unterscheiden sich nur durch ihre Expansion innerhalb von double quotes: `"$@"` → `"$1" "$2"` und `"$*"` → `"$1 $2"`)

```
hello(){
  echo Hello $*, this is a nice new World.
}

hello you there
```

Kontrollstrukturen

Nun fehlen uns noch die bedingten Befehle:

`befehl1 && befehl2` : befehl2 wird nur ausgeführt, wenn befehl1 wahr zurückgeliefert[20] hat.

`befehl1 || befehl2` : befehl2 wird nur ausgeführt, wenn befehl1 false zurückgeliefert hat.

```
if bedbefehl ; then
  wbefehl
else
  fbefehl
fi
```

[19] Manchmal auch schon bei der Ausgabe auf den Bildschirm, dieses Verhalten l"asst sich jedoch wiederum mit `stty tostop` einstellen

[20] wahr entspricht einem Rückgabewert (`$?`) von 0, false allen anderen.

Wenn der bedbefehl wahr zurückliefert, wird wbefehl ausgeführt, ansonsten fbefehl. Der else-Fall kann natürlich auch komplett weggelassen werden. Für die Verwendung von if ist die Kenntniss des "test(1)" - Befehls wichtig:
`test ausdruck` liefert true zurück wenn "ausdruck" wahr ist, und false sonst. Die wichtigsten Elemente für "ausdruck" sind:

`-d file`	wahr wenn file ein Directory ist
`-w file`	wahr wenn file schreibbar ist
`-e file`	wahr wenn file existiert
`n1 -lt n2`	wahr wenn n1<=n2
`n1 -eq n2`	wahr wenn n1=n2 (Zahlen)
`s1 = s2`	wahr wenn sq=s1 (Strings)
`! ausdruck`	wahr wenn ausdruck falsch ist
`a1 -a a2`	wahr wenn a1 und a2 wahr sind
`a1 -o a2`	wahr wenn a1 oder a2 wahr ist

test(1) kann auch als `[` aufgerufen werden, wenn man der Argumentliste eine abschliessende `]` hinzufügt.

```
case a in
match) befehl ;;
...
esac
```

wenn der Inhalt von Variable a auf einen der "match[21]" passt, dann wird der zugehörige Befehl ausgeführt.

Schleifen

Um die Shell zu einer richtigen Programmiersprache zu, machen fehlt natürlich noch etwas. Richtig – Schleifen:

```
for variable in liste ; do
  befehl
done
```

[21] verwendet die selben Sonderzeichen wie beim Filename globbing

"liste" sind durch Spaces getrennte Werte. Die Schleife wird für jeden Wert aus liste einmal durchlaufen

```
while befehl; do
  befehl2
done
```

solange die Ausführung von "befehl" nicht fehlschlägt, wird befehl2 ausgeführt.

Beispiel:

```
crsr(){
  BKSP=‘tput kb‘
  BKSP="$BKSP$BKSP"
  while true ; do
    echo -n " |$BKSP" ; sleep 1
    echo -n " /$BKSP" ; sleep 1
    echo -n " -$BKSP" ; sleep 1
    echo -n " \\$BKSP"  ; sleep 1
  done
}

checkfiles() {
  for file in * ; do
    if [ -d $file ] ; then
      cd $file && checkfiles && cd ..
    fi
    case file in
      *.bak) echo $file sollte gelöscht werden;;
    esac
  done
}

crsr &
prozess=$!
checkfiles
kill $prozess
echo Fertig.
```

3.10 Systemsicherheit

Unter Systemsicherheit versteht man alle Möglichkeiten bzw. Vorgänge, die nötig sind, um ein System vor unterlaubtem Zugriff zu schützen. Auch ist unter Systemsicherheit zu verstehen, dass alle Versuche von möglichen Angriffen mitprotokolliert werden und das System nicht zum Absturz bringen können. Logischerweise gibt es viele Schritte, die nötig sind, um ein System zu sichern. Hier sollen aber nur die Wichtigsten behandelt werden.

In diesem Kursabschnitt werden folgende Möglichkeiten für die Systemsicherheit behandelt:

- Misslungene Loginversuche mit Hilfe einer Datei in */var/adm/loginlog* speichern

Misslungene Loginversuche protokollieren

Egal wie sich ein Benutzer an ein System anmeldet, also lokal oder remote, wird vom System immer die Datei */etc/passwd* und die Datei */etc/shadow* überprüft. Ist der Benutzer authentifiziert, dann wird der Zugang zum System gewährt. Ansonsten wird der Zugang verweigert, und der Benutzer hat keinen Zugang.

In diesem Fall macht es Sinn, die misslungenen Versuche mit zu protokollieren, damit ein Systemadministrator erkennen kann, dass es versucht wurde, unerlaubten Zugang zu erhalten.

Misslungene Loginversuche können in der Datei */var/adm/loginlog* aufgezeichnet werden. Diese Datei is standardmäßig nicht vorhanden. Um dieses Feature benutzen zu können, muss diese Datei angelegt werden. Sie muss Lese- und Schreibrechte nur für root haben und muss der Gruppe sys angehören.

Jeder fehlerhafte Loginversuch wird nun protokolliert. Sollten weniger als fünf fehlerhafte Logins vorkommen, wird kein Eintrag im *loginlog* aufgezeichnet.

3.11 Dateirechte

Um festzulegen wer eine Datei lesen, schreiben oder ausführen darf, besitzen auch Dateien User- und Gruppenzugehörigkeiten. Wird eine angelegt, so geschieht dies unter der UserID und (normalerweise) Login-Group des Benutzers der dies tut. Darüber hinaus werden zu jeder Datei die Dateirechte mit im Filesystem abgelegt, sowie eine Art Typ-Bezeichnung. Die kompletten Zugriffsinformationen zu einer Datei erhält man mit dem Befehl *ls -l*[22] (*ls(1)* ist vergleichbar mit dem `DIR`-Befehl von DOS). Eine typische Ausgabe kann z.B. so aussehen:

```
einstein@kamasutra:~> ls -l einkaufszettel
-rw-r--r--  1 einstein admin         665 Mar 11 22:58 einkaufszettel
```

Der erste Bereich (-rw-r--r--) gibt die Zugriffsrechte an, das zweite Feld die Anzahl der Hardlinks welche diese Datei hat, gefolgt von der User- und Gruppenzugehörigkeit, der Grösse der Datei in Bytes, dem Datum und der Uhrzeit des letzten Schreibzugriffes und dem Dateinamen. Das Feld der Zugriffsrechte setzt sich wie folgt zusammen:

- Das erste Zeichen (hier "-") gibt den Typ dieses Files an:
  - Eine normale Datei, also ein Programm oder Daten in jedlicher Form, werden eben mit jenem "-" dargestellt.
  - Verzeichnisse sind im Sinne des Unix-Dateisystems auch nur spezielle Dateien und werden in diesem Feld mit "d" gekennzeichnet.
  - Dateien im /dev-Verzeichnis, also die Hardware-Repräsentanten, die sogenannten Devices-Files werden mit einem "b" für blockorientierte oder einem "c" für zeichenorientierte Geräte wiedergegeben.
  - Ein symbolischer Link wird hier mit einem "l" gekennzeichnet.
- Die darauffolgenden 9 Zeichen geben die eigentlichen Zugriffsrechte wieder und zwar in drei 3er Gruppen geteilt:

[22] -l für "long". Bei einigen Systemen benötigt es noch ein "-g" zum Anzeigen der GroupID.

  - Die ersten 3 Zeichen bezeichnen die Zugriffsliste für den Inhaber (User, UID) der Datei
  - Die zweiten 3 Zeichen die Rechte für Mitglieder in der Gruppe die dieser Datei zugeordnet wurde.
  - Die letzten 3 Zeichen bezeichnen die Rechte für alle anderen Benutzer (Others), die nicht den ersten beiden zuzuordnen sind.

Innerhalb eines 3er-Blocks gilt ein

- "r" als das Recht zum Lesen der Datei oder im Falle eines Verzeichnisses, um die Dateien in einem Verzeichnis anzeigen zu können.
  - "w" als das Recht in diese Datei oder das Verzeichnis zu schreiben oder zu löschen.
  - "x" als Ausführungsrecht. Dieses wird z.B. von allen Programmen zum ausführen benötigt, aber auch für Verzeichnisse um diese mit *cd* zu betreten.

Bei der Datei im oberen Beispiel handelt es sich also um ein "normales" File mit der Grösse 665 Bytes. Besitzer dieser Datei ist "einstein" und ihre GID ist "admin". Der Besitzer darf diese Datei beschreiben, Mitglieder der Admin-Gruppe, aber auch alle anderen Benutzer dieses Systems, daraus lesen.
Im User- und Group-Feld für das Ausführungsrecht ("x") kann als Sonderfall auch noch ein "s" oder "S", das sogenannte Set-User/Group-ID-Flag gesetzt sein. Ist ein "s" im User-Feld einer normalen Datei gesetzt, so bedeutet dies, dass im Fall einer Ausführung der Datei als Programm, dieses unter der UID des Datei-Besitzers abläuft. Folgender Fall ist denkbar:

```
einstein@kamasutra:~> ls -l /usr/bin/passwd
-r-sr-xr-x  2 root  bin  20480 Nov 16  1995 /usr/bin/passwd
```

Dies bedeutet, dass jeder Benutzer des Systems dieses Programm lesen[23] und starten kann, es aber sobald es läuft die effektive UID (EUID) "root", also

[23] Leserechte benötigt man u.a. zum Kopieren.

"0" annimt[24] und nicht wie normalerweise unter den Rechten des Aufrufers läuft.
Ist das s-Flag im Group-Feld eines Verzeichnisses gesetzt, so werden Dateien unterhalb dieses Verzeichnisses unter der GroupID dieses Verzeichnisses abgelegt, falls man Mitglied dieser Gruppe ist. Ein populäres Beispiel wäre:

```
einstein@bella:/usr/local/www/home> ls -lg
total 2
drwxrwsr-x  3 admin admin       512 Oct 12 20:28 class
drwxrwsr-x 17 admin admin       512 Mar 10 15:35 firmengruppe
```

Falls man Mitglied in der Gruppe "admin" ist und Dateien in einem der beiden aufgeführten Verzeichnisse erzeugt (darf man ja dank dem "w" für die Gruppe), so werden diese unter der GID "admin" abgelegt, was das Bearbeiten von Dateisystem-Bereichen (hier z.B. die WWW-Dateien) durch mehrere Benutzer u.U. vereinfachen kann.
Ein "S" zählt ähnlich einem "s", nur dass keine Ausführungsrechte für User, bzw. Group bestehen.

Um die User- und Gruppenzugehörigkeit einer Datei nachträglich zu bestimmen, kann man diese mit dem Befehl *chown* ändern.
"*chown floriang:SkiClub test.gif*"[25] ändert (entsprechende Rechte vorrausgesetzt) die Zugehörigkeit der Datei "test.gif" zum User "floriang" und der Gruppe "SkiClub". Um nur die Gruppenzugehörigkeit zu ändern, genügt der Befehl *chgrp <Gruppe> <Datei>*. Die Zugriffsrechte lassen sich nachträglich mit dem Befehl *chmod <Mode> <Datei>* modifizieren. Als Mode kann man die Zugriffsrechte im Zahlencode angeben. Diese sind in einer vierstelligen Zahl codiert:

- Eine "2" in der ersten Ziffer setzt das SUID-Flag, eine "4" das SGID-Flag.
- Die weiteren drei Ziffern sind analog der ls-Ausgabe für User-, Group- und Others zu setzen, wobei

[24] Nur so ist das Ändern von */etc/passwd*, die nur für Root schreibbar ist, möglich.
[25] Oft kann man statt des ":" auch ein "." verwenden.

- eine “1” für ausführbar
- eine “2” für schreibbar
- eine “4” für lesbar

steht.

Möchte man z.B. eine Datei für sich selbst und Mitglieder der gleichen Gruppe ausführ- und lesbar machen, für sich selbst schreibbar lassen und allen anderen nur Leserechte auf diese Datei geben, so setzt man *chmod 0754 <Datei>* was in der ls-Ausgabe “-rwxr-xr--” endet.
Eine weitere Möglichkeit Zugriffsrechte mit dem Befehl *chmod* zu setzen, besteht darin die bestehenden Rechte einzeln abzuändern. Dies geschieht

- <Wer> kann aus einem oder einer Kombination von “u” für User, “g” für Group, “o” für Others und “a” für alle bestehen.
- “+” oder “-” sagen aus, ob das Recht hinzukommen oder weggenommen werden soll.
- <Was> kann wiederrum aus einem oder einer Kombination aus “r”, “w”, “x” und “s” bestehen.

So entzieht *chmod go-rwx <Datei>* z.B. allen Nicht-Eigentümern der Datei alle Rechte.

Um die Zugriffsrechte einer neuangelegten Datei zu beeinflussen, gibt es eine Art “default[26]” für Zugriffsrechte, die mit der *umask* Funktion der Shell gesetzt werden. Mit dieser Funktion wird eine Maske gesetzt, welche Rechte ausblendet. Ausgehend von “0777” des o.g. Zahlenmodells (also Schreib- und Leserechte für jedermann; Bei Erzeugung von Executables, z.B. durch einen Compiler, auch noch Ausführungsrechte), werden mit *umask <Maske>* Rechte ausgeblendet. z.B. setzt *umask 22*[27] die Default-Rechte auf “-rw-r--r--” oder *umask 0007* auf “-rw-rw----”.

[26] Grundeinstellung
[27] Führende Nullen werden ignoriert.

3.12 Mounten

Unter mounten versteht man das Einhängen einer physikalischen Resource, z.B. einer Plattenpartition unter einem Directory im Filesystem.

Der Mountpoint

Das Verzeichnis, unter dem die Resource eingehängt ist, wird als Mountpoint bezeichnet.

Der Mountvorgang (manuelle Mounts)

Für temporäres Einbinden wird der manuelle Mount benutzt. Hierzu steht das Kommando *mount* zur Verfügung.

Beispiel für das Mounten:

```
mount -F ufs /dev/dsk/c0t3d0s7 /export/home
```

Beispiel für das Unmounten:

```
umount /export/home
```

Index

Abbildungsverzeichnis