دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

تحت نظارت

دکتر علی فانیان

پاییز ۱۳۹۴

دستورکار جلسه اول

سیستم عامل و اجزای آن

وظایف اصلی سیستم عامل :

- ایجاد فرآیندها
- اختصاص و مدیریت منابع سیستم شامل پردازنده، حافظه، ورودی/خروجی ها و سایر منابع به فرآیندها
- اجرای فرآیندها

اجزای سیستم عامل :

- قسمت اصلی یک سیستم عامل یک هسته مرکزی یا kernel است که اجرای دستورات را برعهده دارد
- دستورات توسط یک خط فرمان یا shell به هسته ارسال شده و در آنجا اجرا می شوند

سیستم عامل UNIX

UNIX سیستم عاملی است که در ۱۹۶۹ در آزمایشگاه های Bell و توسطی گروهی از کارکنان شرکت مخابراتی AT&T ایجاد شد.

از جمله طراحان این سیستم Ken Thompson, Dennis Ritchie, Douglas Mcllory و Joe Ossana را می توان نام برد.

در همین زمان زبان برنامه نویسی C ایجاد شد، این زبان به عنوان ابزاری برای نگهداری ساختمان داده ها و ایجاد تغییرات در UNIX به کار گرفته شد. با گذشت زمان بر قابلیت های UNIX افزوده شد و شرکت های بزرگ نسخه های متفاوتی از این سیستم عامل را برای خود ایجاد کردند و به فروش رساندند. در واقع همه سیستم عامل های موجود ساختار خود را از UNIX گرفته اند گرچه ممکن است ساختاری متفاوت با UNIX نخستین داشته باشند.

در طراحی UNIX دو ویژگی زیر مورد تاکید بوده است :

۱. Multiuser (چند کاربره):

در هر لحظه چندین کاربر می توانند از سیستم استفاده کنند.

۲. Multitask (چند کاره)

هر لحظه چندین فرآیند را مدیریت و اجرا می کند.

مثال هایی از سیستم عامل های UNIX :

- Linux

این سیستم توسط Linus Torvalds و بر مبنای هسته Minix ساخته شد، گرچه معماری آن با Minix بسیار متفاوت است. امروزه نسخه های متنوعی از این سیستم عامل ارائه شده که هر یک کاربرد خاص خود را داراست.

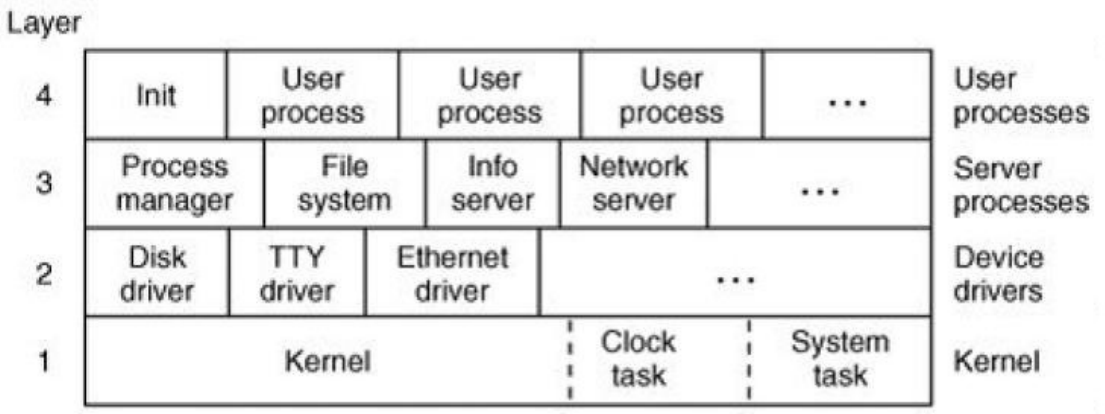
- FreeBSD

این سیستم عامل از Linux پیچیده تر و حرفه ای تر بوده و شباهت بیشتری به UNIX دارد. این سیستم عامل در زمینه شبکه و امنیت قدرت بالایی داشته و اغلب از آن به عنوان سرور با قدرت سرویس دهی بالا استفاده می شود.

- Minix

نسخه ای از سیستم عامل Unix که توسط Andrew S. Tannenbaum برای استفاده آموزشی ایجاد شده است. اولین نسخه آن در ۱۹۸۷ و برای ماشین های IBM PC و IBM PC/AT منتشر شد، همزمان با انتشار هر نسخه از آن کتاب درسی و مرجع آن، کتاب **Operating Systems-Design and Implementation** نیز منتشر می شود. ویرایش سوم این کتاب در سال ۲۰۰۵ و همزمان با سومین نسخه از این سیستم عامل منتشر شد.

لایه های سیستم عامل



- Kernel

در پایین ترین لایه سیستم عامل هسته (kernel) فعالیت می کند. وظیفه هسته عبارتست از :

- اختصاص منابع به فرآیندها
- زمانبندی فرآیندها و تغییر وضعیت اجرایی آنها (ready-running-blocked)
- پیام رسانی و ارتباط بین فرآیندها
- مدیریت وقفه ها
- کنترل ورودی/خروجی (I/O)

همچنین هسته با دو فرآیند اصلی دیگر در ارتباط است:

- Clock Task

ارتباط با ساعت سخت افزار و بررسی سیگنال های زمانی ارسال شده از طریق Clock Task صورت می گیرد

- System Task

Driver های سخت افزاری و Server ها در لایه های بالاتر دستوراتی را برای اجرا در هسته فراخوانی می کنند، این دستورات ترجمه شده و به هسته ارسال می شود، وظیفه بررسی و اجرای این دستورات در هسته بر عهده System Task می باشد.

- Driver

هر وسیله (Device) متصل به سیستم نیاز به واسطی برای ارتباط با هسته و مدیریت این ارتباط دارد. چنین واسطی driver نامیده می شود. در خواست هایی چون نوشتن بر روی دیسک، ارسال داده بر روی شبکه و ... ابتدا توسط برنامه به driver و سپس از driver به هسته ارسال می شود.

- Server

فرآیندهایی در محیط کاربر هستند که به طور تناوبی فعالیتی را انجام می دهند و یا منتظر یک رخداد (event) مانده تا پاسخی به آن بدهند. در سیستم های مبتنی بر UNIX از این فرآیندهای خاص با نام Daemon یاد می شود و در سیستم عامل Windows تحت عنوان Server.

- User Process

این دسته از فرآیندها کمترین سطح دسترسی را به منابع سیستم دارا هستند، هر درخواست آنها بایستی به دو لایه قبل و از آنجا به هسته منتقل شده، اجرا شود.

فضای اجرا و وضعیت کارکرد پردازنده (CPU Execution Mode)

هر یک از لایه های ذکر شده در قسمت قبل در قالب مجموعه ای از فرآیندها در سیستم عامل اجرا می شوند.

فضای کارکرد سیستم عامل به دو دسته تقسیم می شود: فضای هسته (kernel mode) و فضای کاربر (user space)

۱. فضای هسته :

اجرای یک تابع در فضای هسته به معنی دسترسی کامل به همه منابع سیستم عامل است (شامل CPU,RAM,I/O و ...) به همین جهت اگر اشکالی در کد اجرایی در هسته پیش آید باعث ازکار افتادگی (crash) کل سیستم خواهد شد، کدهای اجرایی در هسته با دقت بالایی نوشته می شوند.

۲. فضای کاربر:

برنامه هایی که امکان خراب شدن و از کار افتادگی آنها وجود دارد و در کل از اهمیت کمتری نسبت به هسته برخوردارند در فضای کاربر اجرا می شوند

- مزیت تفکیک وضعیت اجرایی افزایش تحمل سیستم در برابر خطاهای ایجاد شده است، با طراحی کد کارآمد و بدون خطا برای هسته و اجرای سایر موارد در وضعیت کاربر می توان به این سطح بالای تحمل در برابر خطا دست یافت.
- دو فضای موجود صرفا تعاریف نظری نیستند بلکه تغییر بین فضای کاربر و فضای هسته در کد سطح پایین (Assembly) تعیین می شود و پس از آن CPU بین این دو فضا عملکردی جابجا می شود.
- این تفکیک به منظور جلوگیری از تخریب قسمتهای حساس حافظه توسط برنامه های معیوب صورت گرفته است و تنها کارکردهای اساسی از درون هسته به کل سیستم دسترسی دارند

انواع هسته :

تفکیک فرآیندها و توابعی که در فضای کاربر یا در فضای هسته اجرا می شوند سبب ایجاد تعاریف متعددی از هسته شده که در زیر مهمترین آنها بررسی می شوند :

۱. هسته ی یکنواخت (Monolithic Kernel)

در هسته ی یکنواخت همه لایه های سیستم عامل در فضای هسته قرار می گیرند .

مزیت :

سریعتر بودن سیستم به علت یکدست بودن کد

عدم وجود حفره های امنیت بزرگ و مشکلات در عملکرد

عیب :

بازنگری و ایجاد تغییر در کد دشوار است زیرا برطرف کردن مشکلات مجموعه گسترده ای از تغییر در هر مرحله را به دنبال دارد

کامپایل کردن هسته و تست آن زمان زیادی نیاز دارد

سیستم عامل های **Linux**, **FreeBSD**, **NetBSD**, **Solaris** همگی دارای هسته ای یکنواخت هستند.

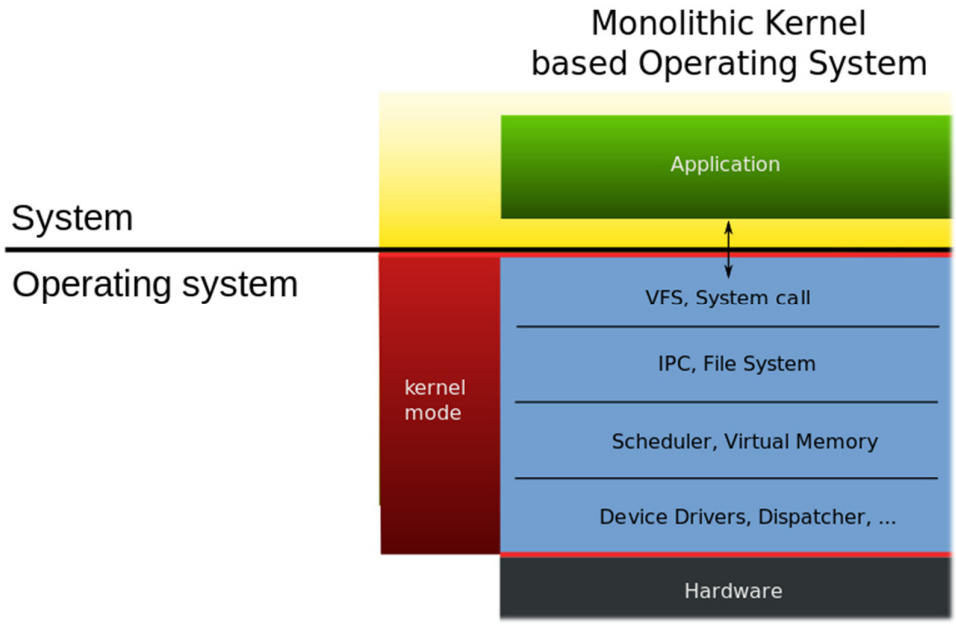
نکته : ایجاد تغییرات در هسته های یکنواخت به دو روش انجام می گیرد :

۱. تغییر کد هسته و کامپایل آن

به دلیل پیچیدگی تا حد امکان از این کار اجتناب می شود.

۲. اضافه کردن ماژول (module) به هسته ی در حال اجرا

ماژول : کدی است به زبان ماشین که می تواند به هسته ی در حال اجرا اضافه شود. این کار باعث کندی موقت سیستم می شود ولی نسبت به تغییر کد هسته بسیار با صرفه تر و سریعتر است.



۲. هسته ی ریز (Micro Kernel)

هسته ی ریز فقط شامل عملکرد های اصلی و پایین ترین لایه سیستم عامل است

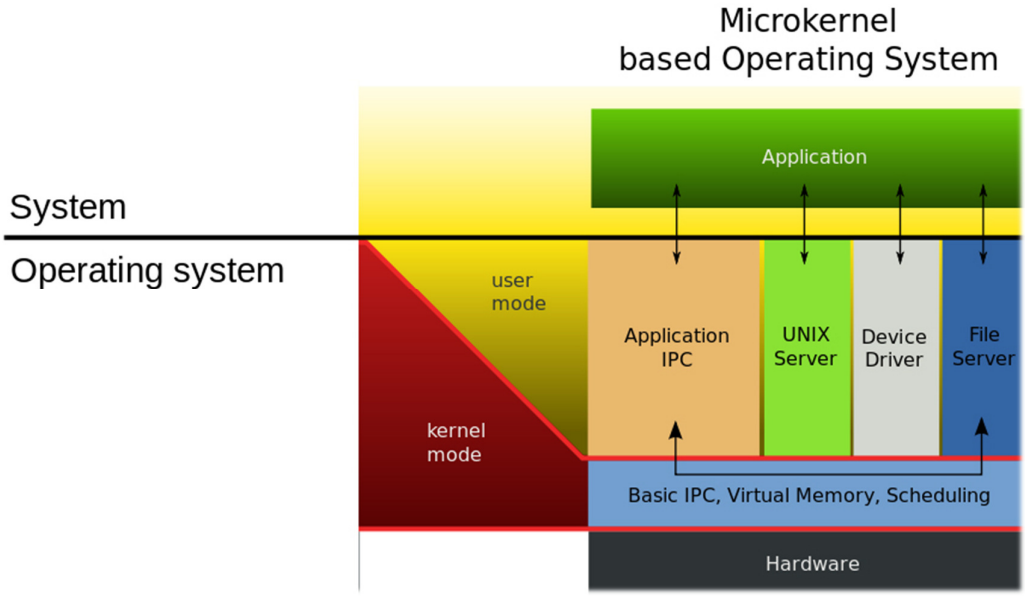
مزیت :

- جدابودن سرویس ها و درایور از هسته میزان تحمل خرابی را افزایش می دهد و در صورت از کار افتادن یکی سرویس ها، تمامی هسته از کار نخواهد افتاد.
- کمتر بودن کد هسته به معنای ایجاد تغییرات سریع تر در آن است

عیب :

- هسته ریز نسبت به دو نوع دیگر عملکرد کندتری دارد.
- مدیریت فرآیندها در آن دشوارتر است.

در حال حاضر هسته ی به کار رفته در **Minix** و همچنین هسته ی **Mach** در این دسته قرار می گیرند. از این هسته با عنوان Server/Client نیز نام برده می شود با این توجیه که چون هر فرآیندی برای اجرا به هسته مراجعه می کند، هسته در نقش سرویس دهنده (Server) و سایر فرآیندها در نقش مشتری (Client) ظاهر می شوند.

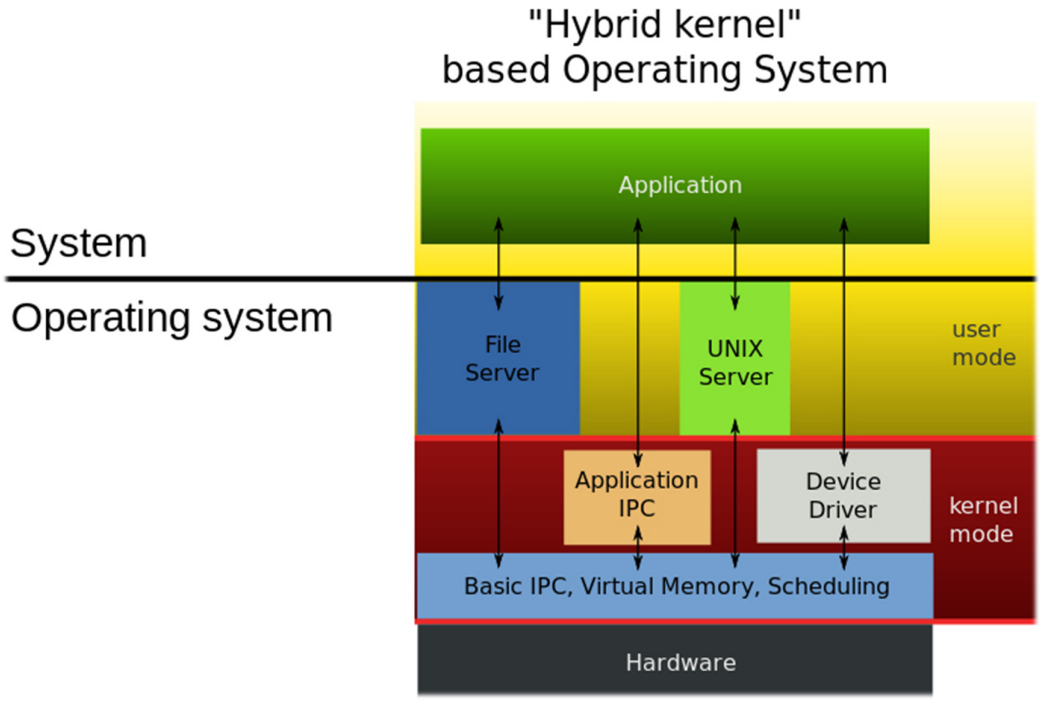


۳. هسته ی ترکیبی (Hybrid Kernel)

این هسته ها ترکیبی از هسته ی یکنواخت و هسته ریز هستند. در پیاده سازی آنها سرویس های سیستم عامل در فضای هسته پیاده سازی شده اند و توجیه این کار نیز سریع تر شدن سرویس ها و در عین حال افزایش اطمینان پذیری سیستم است. سایر موارد در فضای کاربر پیاده سازی شده اند.

بهترین مثال برای هسته های ترکیبی، هسته Windows NT (New Technology) است که نخستین بار در سیستم عامل Windows 3.1 به کار گرفته شد و از آن پس تمامی نسخه های Microsoft Windows از آن استفاده کرده اند.

همچنین هسته XNU(X is Not UNIX) نیز ساختاری ترکیبی دارد، این هسته در سیستم عامل Mac OS X به کار رفته است.



POSIX (Portable Operating System Interface)

به این معنا که همه یا بیشتر ماشین هایی که دارای یک سیستم عامل مبتنی بر UNIX می باشند بتوانند کدی را بدون ایجاد تغییرات اساسی اجرا نمایند.

اغلب سیستم عامل های کنونی با این استاندارد سازگاری دارند، در زیر تعریفی دقیق تر از سازگاری با POSIX ارائه شده است:

۱. سیستم عامل های کاملا سازگار با POSIX (Fully POSIX-Compliant)

سیستم عامل های این دسته ۱۰۰٪ با POSIX مطابقت دارند، از جمله Mac OS X، Solaris و UNIXWare

۲. سیستم عامل های غالبا سازگار با POSIX (Mostly POSIX-Compliant)

این دسته گرچه به طور رسمی به عنوان سازگار با POSIX اعلام نشده اند ولی تا حدود زیادی از این استاندارد پشتیبانی می کنند. از جمله مشهورترین این سیستم عامل ها موارد زیر را می توان نام برد :

Linux, FreeBSD, NetBSD, Minix

آخرین نسخه این استاندارد در ۲۰۰۸ منتشر شده و از طریق آدرس زیر قابل دسترسی می باشد :

http://pubs.opengroup.org/onlinepubs/9699919799/

جدول زیر خلاصه ای از سرویس های اصلی استاندارد POSIX ارائه کرده است :

ساخت و کنترل فرآیندها	Process Creation and Control
سیگنال ها و ساختارهای مدیریت آنها	Signals
خطاهای مربوط به مقادیر اعشاری	Floating Point Exceptions
خطاهای حافظه	Segmentation / Memory Violations
عملیات مربوط به فایل ها و دایرکتوری ها	File and Directory Operations
ساخت و مدیریت خط لوله بین فرآیندها	Pipes
کتابخانه استاندارد C	C Library (Standard C)
کنترل ورودی/خروجی سیستم	I/O Port Interface and Control
زمان سنج ها و ساعت سیستم	Clocks and Timers
مدیریت ارتباط بین فرآیندی : سمافور ها	Semaphores
مدیریت ارتباط بین فرآیندی : تبادل پیام	Message Passing
حافظه مشترک برای مدیریت ارتباط بین فرآیندی	Shared Memory
قفل های حافظه	Memory Locking Interface
مفسر خط فرمان برای تبدیل دستورات کاربر به فراخوانی های سیستم	Command Interpreter
برنامه های کاربردی ارائه شده	Utility Programs

UNIX File System

در ساختار UNIX فایل ها در سه رده طبقه بندی می شوند:

- File: هر داده ای در UNIX به عنوان یک File ذخیره می شود.
- Directory: مجموعه ای از فایل ها در یک directory نگهداشته می شوند.
- File System: چگونگی قرار گرفتن File ها و Directory ها فایل سیستم نامیده می شود.

استاندارد سلسله مراتب فایل سیستم (Filesystem Hierarchy Standard)

این استاندارد توسط بنیاد لینوکس **(Linux Foundation)** نگهداری می شود و در آن سلسله مراتبی برای نحوه ی قرارگیری دایرکتوری ها و محتوای آنها ارائه شده است.

برای دسترسی به ساختار سلسله مراتبی در سیستم عامل می بایست نتیجه دستور `man hier` را مشاهده کرد.

سلسله مراتب آورده شده در زیر ممکن است در نسخه های مختلف سیستم عامل های مبتنی بر UNIX تفاوت های جزئی داشته باشد :

- /

در UNIX همه فایل ها بایستی جزئی از فایل سیستم موجود آن باشند و همه آدرس ها نسبت به یک مبدا تعیین می شوند.

مبدا آدرس ها در UNIX دایرکتوری root می باشد که با نماد / نمایش داده می شود، این به این معناست که همه مسیر های سیستم از این مبدا آدرس دهی می شوند و بنابراین می توان با دسترسی به دایرکتوری root به همه فایل های درون سیستم دسترسی داشت

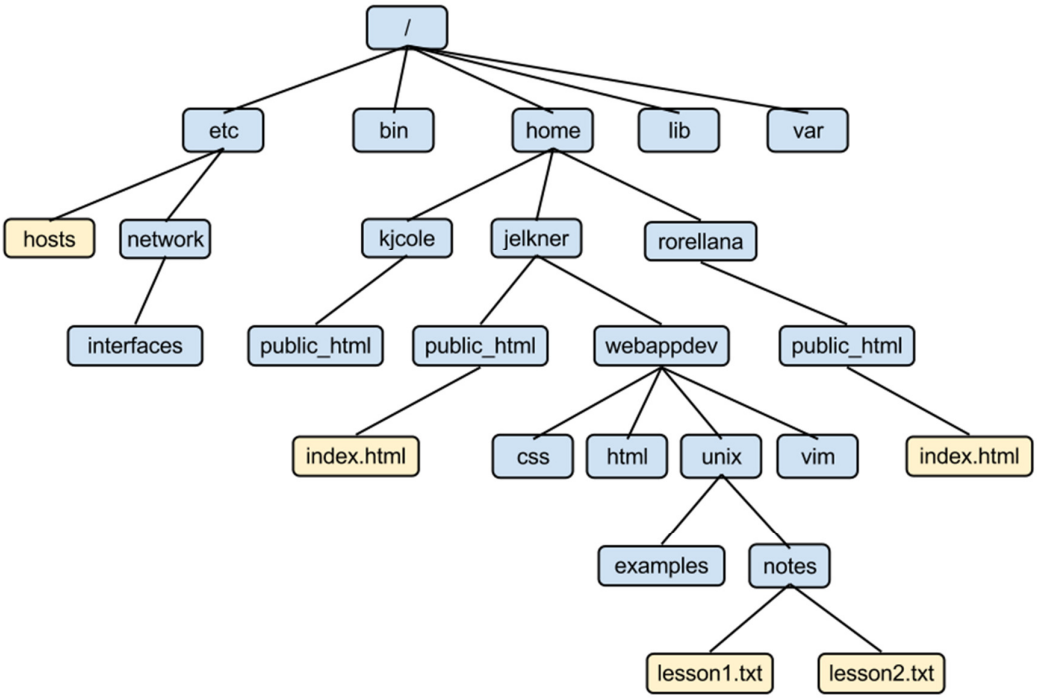
البته دسترسی به این دایرکتوری نیازمند داشتن سطح دسترسی در سطح کاربر root می باشد.

نکته :

دایرکتوری / به عنوان دایرکتوری root شناخته می شود، ولی در عین حال دایرکتوری دیگری به نام /root نیز وجود دارد که با / تفاوت دارد و خود جزئی از / است. دایرکتوری /root به عنوان دایرکتوری خانه برای کاربر root به کار می رود.

- دایرکتوری خانه :~

برای هر کاربر یک دایرکتوری خانه (home directory) ساخته می شود. این دایرکتوری در آدرس /home/username قرار دارد که در آن username نام کاربر مورد نظر است. همچنین هر کاربر پس از وارد شدن می تواند تنها با رفتن به آدرس ~ به دایرکتوری خانه خود دست پیدا کند.



- انتصاب (mount)

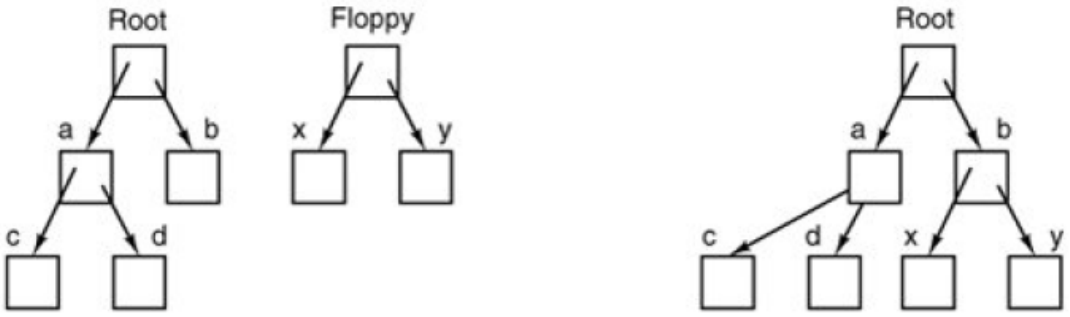
اضافه کردن موقت یک آدرس خارجی به دایرکتوری root را mount می نامند.

هر دایرکتوری خارج از فایل سیستم (همانند یک دایرکتوری که از یک حافظه قایل حمل خوانده می شود برای مثال از حافظه فلش یا هارد دیسک) بایستی نسبت به / آدرس دهی شود.

- نقطه ی انتصاب (mount point)

جایی از فایل سیستم که فایل سیستم خارجی به آن متصل می شود را نقطه ی اتصال می نامند. تا زمانی که انتصاب فایل سیستم خارجی جریان دارد، محتوای نقطه ی انتصاب غیر قابل دسترس است.

مثال : در تصویر سمت چپ Floppy با دو دایرکتوری x و y وجود دارد، در صورت اتصال به سیستم بایستی به عنوان بخشی از فایل سیستم اصلی منتصب شود. در تصویر سمت راست دایرکتوری b به عنوان نقطه انتصاب قرار گرفته و دایرکتوری های x و y از مسیر /root/b/ قابل دسترس اند، همانطور که گفته شده تا زمانی که انتصاب برقرار است محتوای b غیر قابل دسترس خواهد بود.



- جدول زیر خلاصه ای از فایل سیستم و محتوای هر دایرکتوری ارائه می دهد :

/	شامل دایرکتوری های سیستم عامل در بالاترین سطح
/bin /usr/bin	محل قرار گرفتن فایل های اجرایی (binary)، این فایل ها برای همه کاربران قابل دسترسی است
/dev	محل قرار گرفتن فایل های واسطه ی سخت افزاری
/etc	اغلب فایل های پیکربندی سیستم (configuration)، اطلاعات مربوط به کاربران و گروه ها، اطلاعات مربوط به سیستم (host) در این دایرکتوری قرار می گیرند
/lib	محل قرار گرفتن فایل های کتابخانه ای سیستم (library)
/boot	محل قرار گرفتن فایل های مورد نیاز برای راه اندازی سیستم (boot)
/home	محل قرار گرفتن دایرکتوری خانه برای هر کاربر
/mnt	محل قرار گرفتن فایل سیستم های موقتی که به سیستم mount شده اند
/proc	محل قرار گرفتن اطلاعات مربوط به فرایندها (process)
/tmp	محل قرار گرفتن فایل های موقتی که در حین راه اندازی سیستم به کار می روند این دایرکتوری اغلب خالی ست به این دلیل که سیستم پس از راه اندازی محتوای آن را پاک خواهد کرد
/usr	این دایرکتوری کاربردها مختلفی دارد از جمله قرارگرفتن برخی فایل های اجرایی درآن، فایل های کتابخانه ای ، فایل های به اشتراک گذاری شده و ...
/var	محل قرار گرفتن فایل های متغیر (variable)، از جمله فایل های ثبت وقایع (log) ، فایل های مربوط به چاپگر

محل قرار گرفتن فایل های اجرایی (binary)، تفاوت این دایرکتوری با bin/ در این است که sbin/ فقط برای کاربر root قابل دسترسی است و فایل های اجرایی مربوط به مدیریت سیستم (ابزارهای پیکربندی دیسک، شبکه و ...) می باشند در این محل قرار می گیرند	/sbin /usr/bin
محل قرار گرفتن فایل های هسته	/kernel

دستورات خط فرمان

چند صد دستور پایه در خط فرمان UNIX وجود دارد که در اینجا مهمترین دستورات بررسی می شوند :

- man

نمایش راهنما برای یک دستور

نکته : ممکن است دستوری با یک نام کاربردهای متفاوت وجود داشته باشند، در جدول زیر انواع این کاربردها بررسی شده اند:

Section	Description
1	General commands
2	System calls
3	Library functions, covering in particular the C standard library
4	Special files (usually devices, those found in /dev) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellaneous
8	System administration commands and daemons

- apropos

این دستور عبارتی را می گیرد و در بین همه راهنماهای موجود آن عبارت را جستجو می کند.

- دستورات File System

ساخت یک فایل	touch
ساخت فایل با نام 1.txt	touch 1.txt
ساخت یک یا چند دایرکتوری	mkdir
ساخت دایرکتوری با نام dir1	mkdir dir1
ساخت مجموعه ای از دایرکتوری ها به ترتیب وارد شده	mkdir dir1 --> cd dir1 mkdir dir2 --> cd dir2 mkdir dir3
	mdkir -p ~/dir1/dir2/dir3
حذف یک یا چند دایرکتوری خالی	rmdir
	rmdir dir1 dir2 dir3
حذف یک یا چند دایرکتوری دارای فایل	rmdir -r
	rmdir -r dir1 dir2 dir3
کپی یک یا چند فایل و یا کپی یک یا چند دایرکتوری خالی	cp
	cp 1.txt 2.txt
کپی یک یا چند دایرکتوری غیر خالی (دارای فایل)	cp -r
	cp dir1 /path/dir2
جابجا کردن یک یا چند فایل و یا جابجا کردن یک یا چند دایرکتوری خالی (دارای فایل)	mv mv 1.txt /path/2.txt mv 1.txt 2.txt mv dir1 /path/dir2
جابجا کردن یک یا چند دایرکتوری غیر خالی (دارای فایل)	mv -r
تغییر نام فایل یا دایرکتوری	ren
تغییر نام فایل 1.txt به 2.txt	ren 1.txt 2.txt

نمایش مسیر کنونی	pwd
تغییر مسیر کنونی به /path	cd /path
برابر با دایرکتوری فعلی	.
برابر با دایرکتوری بالاتر از از دایرکتوری فعلی	..
لیست فایل های درون یک دایرکتوری، در حالت پیش فرض فایل های درون دایرکتوری کنونی	ls /path
لیست همه فایل های درون دایرکتوری شامل فایل های مخفی (فایل های مخفی با . شروع می شوند)	ls -a /path
لیست فایل های درون دایرکتوری به همراه جزییات آنها	ls -l /path

- دستورات System

خاموش کردن سیستم	halt و shutdown
ریست کردن سیستم	reboot
نمایش نام کاربر در پوسته ی جاری	whoami
تغییر رمز عبور کاربر، در حالت پیش فرض برای کاربر فعلی	passwd
نمایش اطلاعات کاربران، در حالت پیش فرض برای کاربر فعلی	id

- دستورات Process Management

نمایش فرآیندهای در حال اجر	ps
نمایش فرآیندهایی که در ترمینال اجرا می شوند	ps -a
نمایش فرآیندهای	ps -l
نمایش همه فرآیندها، شامل فرآیندهایی که در حال اجرا هستند ولی در ترمینال نمایش داده نمی شوند.	ps -x
ارسال سیگنال برای یک فرآیند، همچنین به پایان رساندن یک فرآیند	kill
ارسال سیگنال SIGNUM به فرآینده با شماره ی pid	kill -SIGNUM pid

ارسال سیگنال ۹ برای کشتن فرآیند با شماره ی pid	`kill -9 pid`

- دستورات Shell

خروج از پوسته جاری و بازگشت به پوسته ی قبلی در صورتی که پوسته کنونی تنها پوسته در حال اجرا باشد، کاربر خارج می شود (مشابه logout)	`exit`
پاک کردن همه نوشته های ترمینال جاری	`clear`
یک دنباله از دستورات که توسط علامت | از یکدیگر جدا شده اند را یک خط لوله (pipeline) می نامند. عملکرد خط لوله به این شکل است که خروجی دستور سمت چپ | به عنوان ورودی دستور سمت راست | تلقی می شود، اجرای این دستورات از سمت چپ ترین دستور شروع شده و مرحله به مرحله پیش می رود.	`pipeline`
این دستور ورودی خود را در یک صفحه ی مجزا نمایش می دهد	`more`
این دستور ورودی خود را در یک صفحه ی مجزا نمایش می دهد، اغلب زمانی که نتایج اجرای یک دستور بیشتر از یک صفحه باشد از less استفاده می شود.	`less`
نمایش 1.txt در صفحه ای متمایز و چاپ کردن شماره خطوط	`less -N 1.txt`
	`ls -a | less`
نمایش محتوای یک فایل با شروع از ابتدای آن	`head`
نمایش ۱۰ خط اول 1.txt	`head -10 1.txt`
نمایش محتوای یک فایل با شروع از انتهای آن	`tail`
نمایش ۱۰ خط آخر فایل 1.txt	`tail -10 1.txt`
محل قرار گرفتن فایل اجرایی را نمایش می دهد	`whereis`

پیدا کردن محل دستور mkdir	`whereis mkdir`
	`/bin/mkdir`
	`/usr/bin/mkdir`

مشابه whereis	which

- cat

یکی از دستوراتی که برای الحاق چند فایل به یکدیگر یا مشاهده محتوای فایل ها به کار می رود	cat
نمایش محتوای 1.txt بر روی خروجی استاندارد (stdout)	cat 1.txt
2.txt به انتهای 1.txt الحاق شده و نتیجه در 3.txt ذخیره می شود	cat 1.txt 2.txt > 3.txt
2.txt به انتهای 1.txt الحاق شده و نتیجه آن به انتهای 3.txt اضافه می شود.	cat 1.txt 2.txt >> 3.txt
آنچه در خروجی استاندارد نوشته شود در 1.txt ذخیره می شود، این روند تا دریافت EOF ادامه پیدا می کند (در UNIX فشردن Ctrl + D باعث ارسال EOF می شود.)	cat > 1.txt

grep

جستجو درون یک فایل بر حسب یک الگوی تعیین شده (pattern)	grep

- find

جستجوی سلسله مراتبی درون فایل سیستم	find
در مسیر path فایل ها و دایرکتوری های که در نام خود pattern را داشته باشند جستجو می کند.	find path -name pattern
فایلها یا دایرکتوری های که با نام include شروع می شوند و در یکی از زیرشاخه های / قرار گرفته اند.	find / -name include

- Disk

نمایش فضای اختصاصی و فضای خالی متعلق به دایرکتوری های منتصب شده (mount) در فایل سیستم	df
نمایش فضای اختصاصی به هر فایل درون یک دایرکتوری	du
مدیریت جدول پارتیشن های دیسک (Partition Table Editor)	part

انواع فایل در UNIX

- سطح دسترسی فایل ها

در UNIX، هر موجودیتی تحت عنوان یک فایل شناخته می شود.

از طرف دیگر از هر سیستم تعدادی کاربر استفاده می کنند که هر یک از آنها متعلق به یک یا چند گروه تعریف شده در سیستم هستند.

هر فایل در سیستم متعلق به یک کاربر و یک گروه است، مالک و گروه هر فایل در هنگام ایجاد آن تعیین می شوند و به طور پیش فرض مالک هر فایل ایجاد کننده آن و گروه هر فایل همان گروهی ست که مالک فایل در لحظه ایجاد فایل به آن تعلق دارد.

می توان پس از ایجاد فایل مالک و گروه آن را عوض کرد.

برای هر فایل در UNIX برای سه گروه سطح دسترسی تعریف شده است :

مالک فایل (owner)

گروه فایل (group)

سایر افراد (other)

برای هر یک از سه حالت فوق، سه سطح دسترسی در نظر گرفته شده است :

خواندن (read)

نوشتن (write)

اجرا کردن (execute)

نکته : برای دایرکتوری گزینه خواندن به معنای مشاهده لیست فایل های داخل آن است ولی برای دسترسی به درون دایرکتوری باید گزینه اجرا نیز فعال شده باشد.

با اجرای دستور "ls -l" می توان سطح دسترسی های هر فایل یا دایرکتوری را مشاهده کرد که در یک رشته با ۱۰ کاراکتر قرار دارد :

- rwx rwx rwx

کاراکتر اول نوع فایل را مشخص می کند:

Regular	-
Directory	d
Socket	s
named pipe	p
symbolic link	l
block device	b
char device	c

از آن پس هر دسته ۳ تایی کاراکتر ها به ترتیب سطح دسترسی برای مالک، گروه و سایر افراد را مشخص می کنند.

برای هر یک از این سطح دسترسی یک مقدار octal در نظر گرفته شده است :

execute = 1 write = 2 read = 4

در هر حالت اگر دسترسی وجود داشته باشد، عدد آن را لحاظ می کنیم و اگر دسترسی وجود نداشته باشد، مقدار معادل آن را ۰ در نظر می گیریم، برای محاسبه عدد نهایی سطح دسترسی این ۳ مقدار با یکدیگر جمع می شوند

سطح دسترسی خواندن و نوشتن و اجرا	4+2+1 = 7
سطح دسترسی خواندن و نوشتن	4+2+0 = 6
سطح دسترسی خواندن و اجرا	4+0+1 = 5

سطح دسترسی نوشتن و اجرا	0+2+1 = 3
فقط سطح دسترسی خواندن	4+0+0 = 4
فقط سطح دسترسی نوشتن	0+2+0 = 2
فقط سطح دسترسی اجرا	0+0+1 = 1

دو روش برای تعریف سطح دسترسی وجود دارد :

۱. استفاده از معادل octal سطح دسترسی

	chmod octal_permission file_name
خواندن، نوشتن و اجرا برای مالک خواندن و اجرا برای گروه خواندن و اجرا برای سایرین	chmod 755 1.txt
تغییر سطح دسترسی برای همه فایل های داخل دایرکتوری	chmod -R permission directory

۲. استفاده از معادل الفبایی سطح دسترسی

	chmod who ± mode file_name

who :

u	user
g	owner
o	group
a	all

mode :

r	read
w	write
x	execute

chown user_name file_name	تغییر مالکیت فایل
chgrp group_name file_name	تغییر گروه فایل

- Wildcards

محیط پوسته به ما این امکان را می دهد که از الگوهایی که در یک رشته وجود دارند استفاده کنیم، به این ترتیب می توانیم :

۱. عملیاتی را برای دسته ای از فایل ها که نامشان دارای یک الگوی مشترک است را انجام دهیم.

۲. یک الگوی خاص را درون یک فایل نوشته شده جستجو کنیم.

برای این کار باید از دو ابزار Standard Wildcard و همچنین عبارات منظم (regular expression) استفاده کرد.

Standard Wildcards :

مجموعه ای از کاراکتر ها که در محیط پوسته معنای ویژه ای دارند :

به عنوان یک کاراکتر در رشته تلقی می شود.	?
به عنوان صفر کاراکتر یا بیشتر تلقی می شود.	*
برای تعیین یک بازه به کار می رود.	[ ]
برای تعیین لیستی از مقادیر به کار می رود.	{ }
عملکردی مشابه [ ] دارد با این تفاوت که دستور را برای الگوی داده شده اجرا نمی کند.	[!]

مثال :

فرض کنید به دنبال اجرای دستور ls برای تمام دایرکتوری هایی باشید که طول ۴ نامشان ۴ حرف است و با abc شروع می شود، می توانید از دستور مقابل استفاد کنید :

```
ls abc?
```

مثال :

فرض کنید به دنبال اجرای دستور ls برای تمام دایرکتوری هایی باشید که طول نامشان بین ۳ تا ۶ حرف است، همگی با ab شروع می شوند و با e تمام می شوند، در اینصورت می توانید از دستور مقابل استفاده کنید :

```
ls    ab*e
```

```
ls    ab???e
```

مثال :

فرض کنید تعدادی فایل به نام های زیر وجود دارند :

```
 app11 app10 app9 app8 app7 app6 app5 app4 app3 app2 app1
                                                    app12
```

حال اگر به دنبال اجرای دستور ls برای تمامی فایل هایی هستید که نامشان با app شروع می شود و عددشان بین ۱ تا ۵ است می توانید از دستور مقابل استفاده کنید :

```
ls app[1-5]
app1  app2  app3  app4  app5
```

نکته : دقت کنید که فاصله ای بین عبارت مورد جستجو و [ ] وجود نداشته باشد، همچنین بین هر , و الگویی که در [ ] آورده شده نباید هیچ فاصله ای وجود داشته باشد.

مثال :

فرض کنید در مثال قبل به دنبال اجرای دستور ls برای فایل های app1,app3,app5,app7 هستید، در اینصورت می توانید دستور مقابل را اجرا کنید :

```
ls app[1,3,5,7]
```

نکته : بازه ی تعریف شده باید دقیقا به وسیله ۱ کاراکتر و یا ۱ رقم توصیف شود.

مثال :

در مثال قبل اگر به دنبال اجرای دستور ls برای فایل های app باشید که اندیس آنها بین ۱۰ تا 19 باشد، باید دستور مقابل را اجرا کنید :

```
ls app[1][0-9]
```

مثال :

فرض کنید فایل های زیر وجود دارند :

```
applicaion.py    application.c    application.sh
```

شما به دنبال اجرای دستور ls برای application.py و application.sh هستید، در اینصورت اجرای دستور مقابل با خطا مواجه می شود :

```
ls application[.sh,.py]
```

علت خطا اینست که بازه ی تعیین شده باید دقیقا توسط یک کاراکتر تعیین شود و نه بیشتر

مثال :

در مثال قبل اگر شما به دنبال اجرای دستور ls برای application.py و application.sh هستید، می توانید دستور مقابل را برای آن اجرا کنید :

```
ls application{.py,.sh }
```

توجه داشته باشید که در این حالت نیز نباید فاصله ای بین اسم مورد جستجو و { } وجود داشته باشد.

همچنین داخل بین هر , و الگویی که به دنبال آن هستیم و در داخل { } آورده شده اند نباید هیچ فاصله ای وجود داشته باشد.