فصل ۲: ساختار سیستم عامل

اهداف این فصل

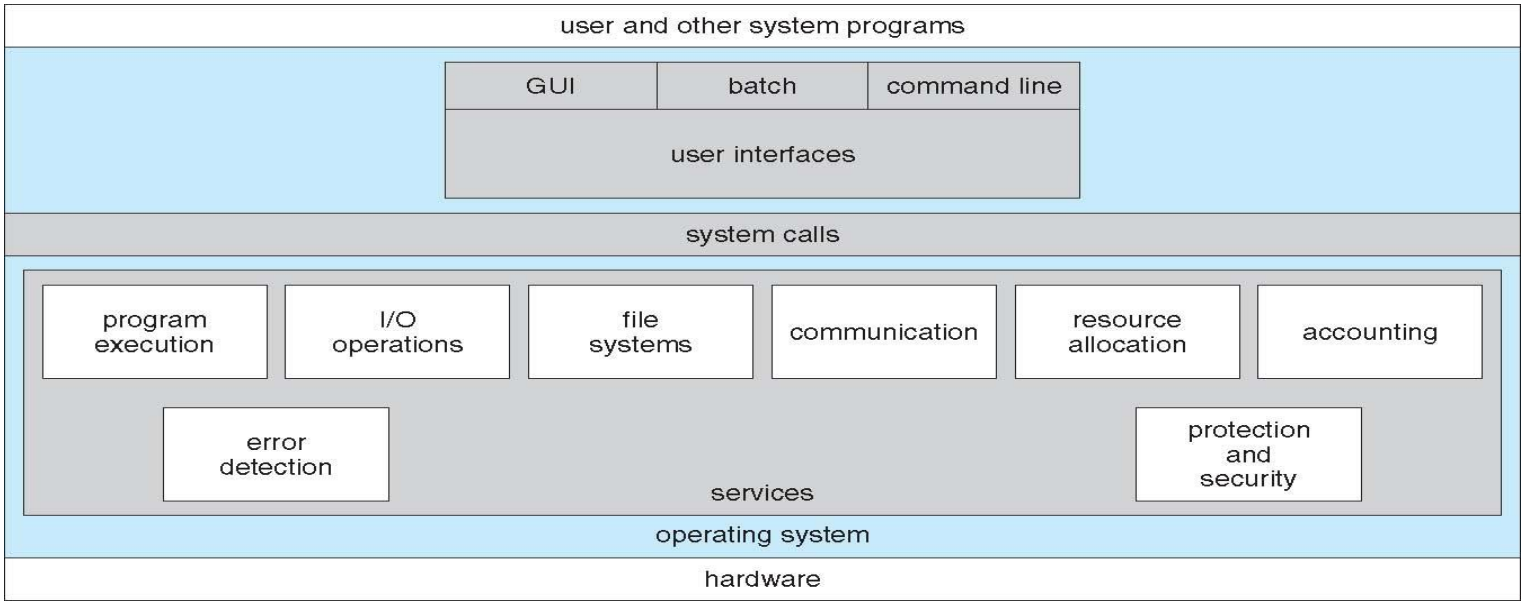
- تشریح خدمات ارائه شده توسط سیستم عامل به کاربران، پردازه ها و دیگر سامانه ها
- معماری های متداول سیستم عامل ها
- بررسی معماری چند سیستم عامل

سیستم های عامل

- از دیدهای مختلفی می توان سیستم های عامل را بررسی نمود.
  - سرویس ها و خدماتی که سیستم های عامل ارائه می نمایند
  - واسطه های کاربری که به کاربران و برنامه نویس ها ارائه می دهند
  - مولفه های سیستم عامل و ارتباط بین آنها

سرویس ها و خدمات سیستم های عامل

- سیستم عامل ها محیطی برای اجرای برنامه ها فراهم می کنند
- سیستم عامل ها سرویس هایی برای برنامه ها و کاربران آنها فراهم می کنند
- نوع سرویس های ارائه شده از یک سیستم عامل به سیستم عامل دیگر متفاوت است اما می توان آنها را در گروه های زیر دسته بندی نمود.
  - سرویس هایی که برای آسان کردن کار به کاربر و کمک به او فراهم می شود.
  - سرویس هایی که برای افزایش کارایی ارائه می شود.



سرویس های ارائه شده به کاربران

- واسطه کاربری
  - پردازش خط فرمان (Command-Line (CLI))
  - واسطه گرافیکی (Graphics User Interface (GUI))
  - پردازش دسته ای (Batch)
- اجرای برنامه ها
  - شامل بارگذاری، پایان و آزاد سازی برنامه ها
- عملیات ورودی و خروجی
- عملیات فایل و دایرکتوری
  - مانند خواندن، نوشتن، ایجا و حذف فایل
  - مانند ایجاد و حذف دایرکتوری
- ارتباط بین پردازه ها
  - حافظه اشتراکی و ارتباط با پیام
- تشخیص خطا

سرویس های ارائه شده برای بهبود کارایی

- تخصیص منابع مانند
  - تخصیص پردازنده
  - تخصیص و آزاد سازی حافظه
  - تخصیص ورودی و خروجی
- حسابداری
  - یک کاربر چقدر از هرکدا از منابع استفاده نموده است
- محافظت و امنیت

واسطه کاربری

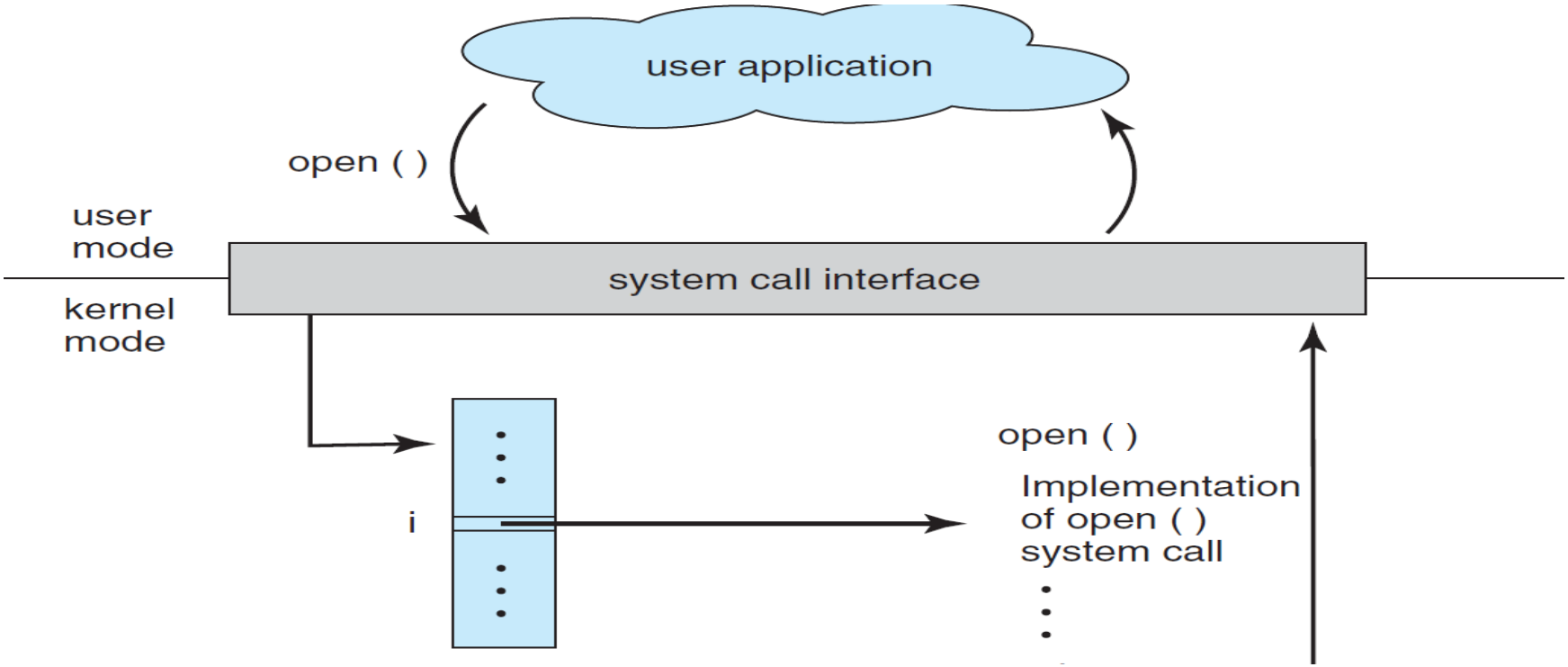
- پردازش خط فرمان ((CLI) Command-Line)
  - در برخی از از سیستم عامل ها بخشی از هسته است اما در بیشتر سیستم عامل ها از قبیل Linux و Windows این بخش برنامه جدایی بوده و زمانی که کاربر وارد سیستم می شود (Login) و یا یک برنامه را اجرا می کند بارگذاری می شود.
  - در برخی از سیستم عامل ها می توان CLI های متفاوتی داشت مانند Shell های مختلف در لینوکس و یونیکس
  - مهمترین عملکرد CLI دریافت و اجرای فرمان های کاربر
  - شیوه های پیاده سازی CLI
    - این بخش حاوی کدهای همه دستورها باشد
      - کاستی ها: مشکل گسترش، حجم کد و حافظه مورد نیاز جهت اجرا
    - این بخش تنها از دستورها برای پیدا کردن نام فایلی استفاده می کند که پیاده سازی دستور در آن است.
    - برتری ها : کوچک بودن این بخش و قابل گسترش بودن آن

واسطه کاربری (ادامه)

- واسطه گرافیکی
  - در اوایل دهه ۱۹۷۰ توسط شرکت زیراکس
  - در اوایل دهه ۱۹۸۰ توسط شرکت اپل

فراخوان های سیستمی (System Calls)

- فراخوان های سیستمی یک واسطه برای سرویس های ارائه شده توسط سیستم عامل را فراهم می کند.



فراخوان سیستمی (ادامه)

- بیشتر سیستم عامل ها به جای فراخوان سیستمی، API های سطح برنامه نویسی را فراهم می کنند
  - Windows API
  - POSIX API
  - برای نمونه CreateProcess() در ویندوز NTCreateProcess() را فراخوانی می نماید.

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

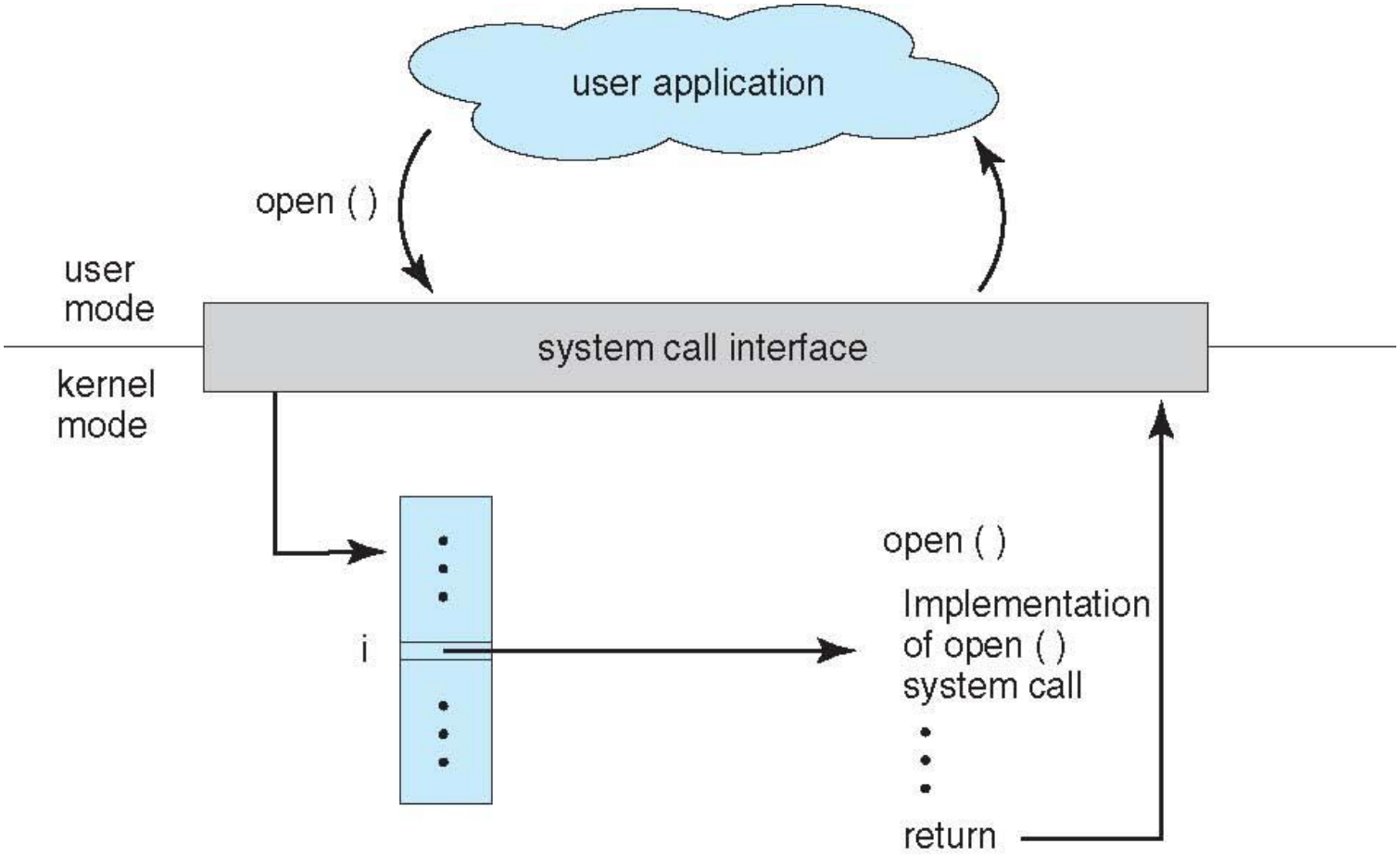
return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

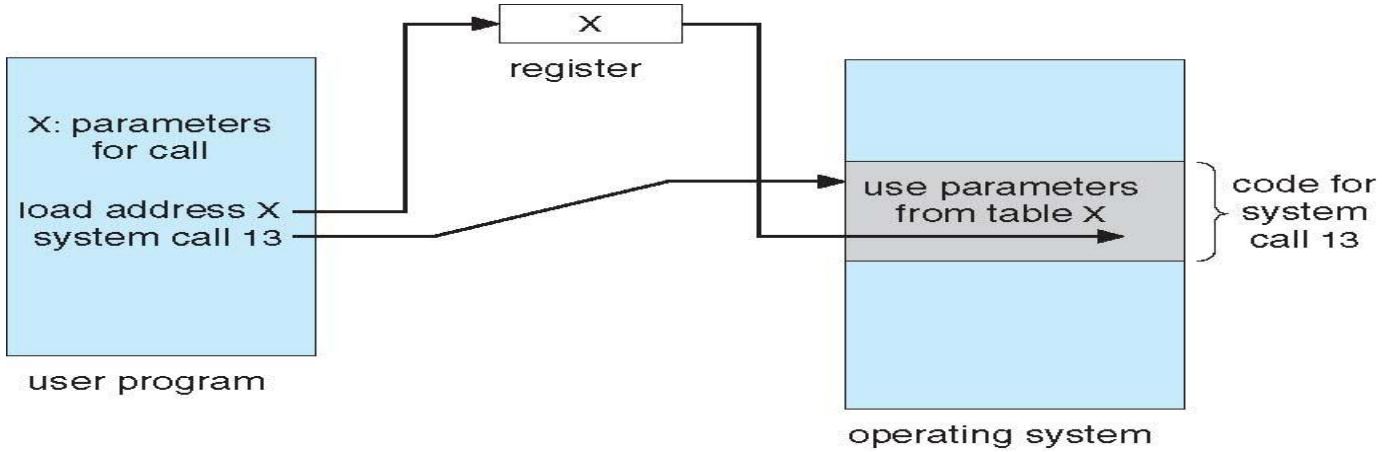
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

ارتباط بین API و فراخوان سیستمی



شیوه های مختلف ارسال پارامترها به فراخوان های سیستم

- ثبات
- ذخیره سازی در یک جدول و فرستادن نشانی جدول (مانند لینوکس)



- قرار دادن در پشته

انواع فراخوان های سیستمی

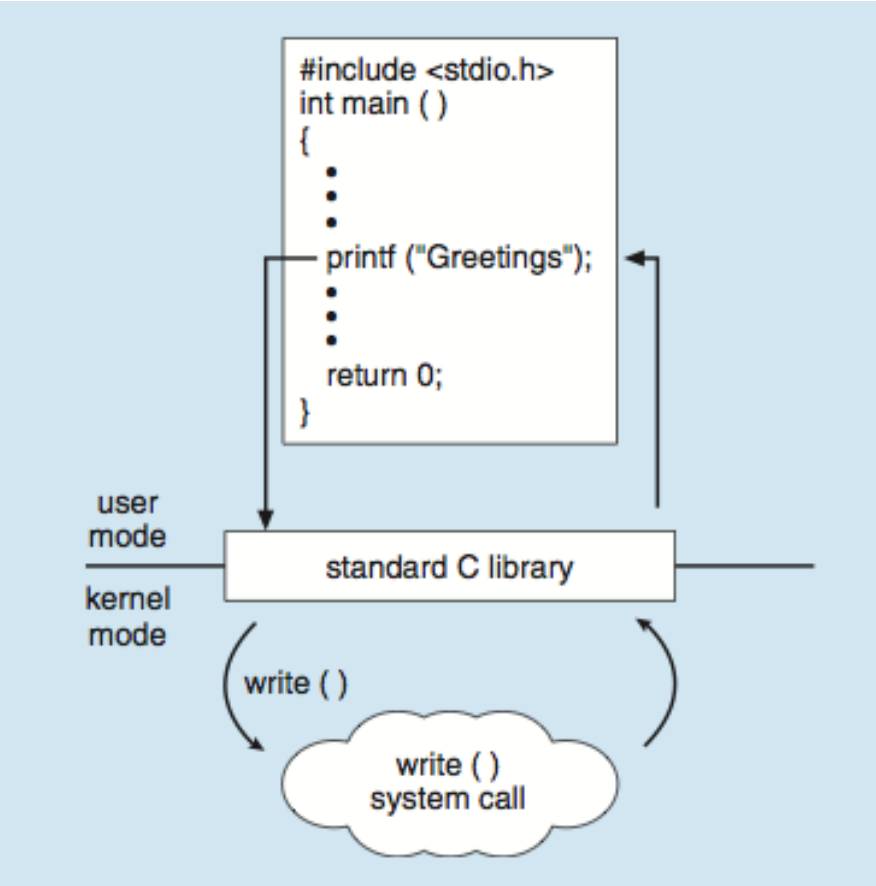
- کنترل پردازه ها
  - ایجاد، حذف، پایان دادن، بارگذاری و ... پردازه ها
- پردازش فایل ها
  - ایجاد، باز کردن، بستن و حذف فایل ها
- پردازش دستگاه ها
  - درخواست دستگاه، آزاد سازی دستگاه، خواندن از و نوشتن به دستگاه
- نگهداری داده ها
  - زمان، تاریخ و dump حافظه
- ارتباطات
  - ارسال و دریافت پیام بین پردازه ها
- محافظت

نمونه API ها

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

نمونه ای کتابخانه زبان C

- program invoking printf() library call, which calls write() system call



برنامه های سیستمی

- بیشتر سیستم عامل ها علاوه بر فراخوان های سیستمی ، برنامه های سیستمی را نیز در اختیار کاربران قرار می دهند. این برنامه ها را می توان به صورت زیر دسته بندی نمود.
- مدیریت فایل
  - ایجاد، حذف کپی، تغییر نام ... فایل
- ویرایش فایل
- پشتیبانی از زبانهای برنامه نویسی
- بارگذاری و اجرای برنامه ها
- ارتباطات
  - مانند مرورگر های وب
- برنامه های سیستمی دیگر

طراحی و پیاده سازی سیستم عامل ها

- اهداف طراحی
  - نوع سخت افزار
  - نوع سیستم
    - اشتراک زمانی، Batch، تک کاربره، چند کاربره، توزیع شده، بی درنگ ، کاربرد عمومی
  - نیازمندی ها
    - نیازمندی های کاربر شامل راحتی کار، سریع بودن، مطمئن بودن
    - نیازمندی های سیستم شامل پیاده سازی و نگهداری آسان، انعطاف پذیر و بدون خطا

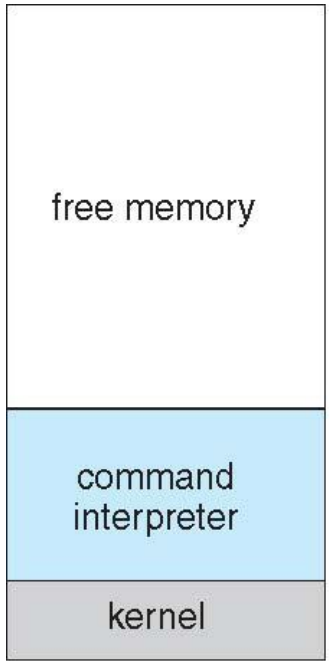
پیاده سازی سیستم عامل ها

- پیاده سازی با زبانهای سطح پایین
  - کارایی خوب
  - نگهداری پرهزینه و توسعه پذیری سخت
- پیاده سازی با زبانهای سطح بالا
  - نگهداری ساده
  - کاهش کارایی

پایان جلسه ۲

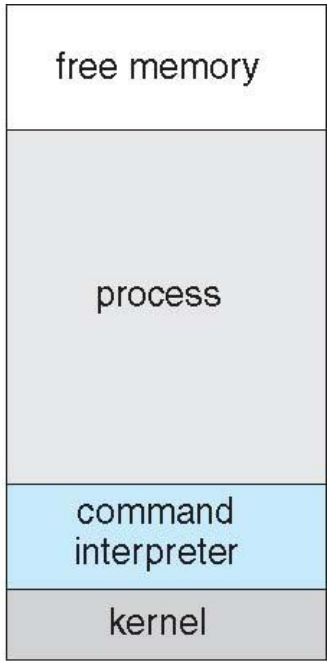
نمونه ای از سیستم عامل ها:MS-DOS

- تک وظیفه ای
- خط فرمان هنگام راه اندازی اجرا می شود.
- روش ساده برای اجرا و پردازه ای ایجاد نمی شود.
- دارای یک فضای آدرس حافظه
- بارگذاری برنامه در حافظه و پاک کردن همه حافظه به جز هسته
- بارگذاری دوباره خط فرمان هنگام خروج برنامه از حافظه.



(a)

At system startup

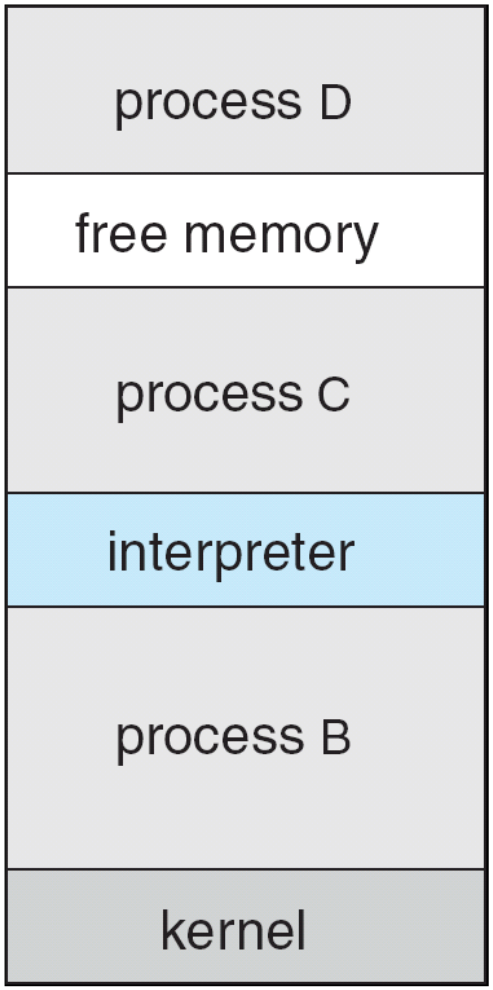


(b)

running a program

نمونه ای از سیستم عامل ها:FreeBSD

- نسخه ای از یونیکس
- چند وظیفه ای
- خط فرمان هنگام login بار گذاری می شود.
- خط فرمان از ()fork برای ایجاد پردازه و سپس از ()exec برای بارگذاری برنامه استفاده می کند.
- خط فرمان منتظر پایان پردازه می ماند یا می تواند به کار خود ادامه دهد.

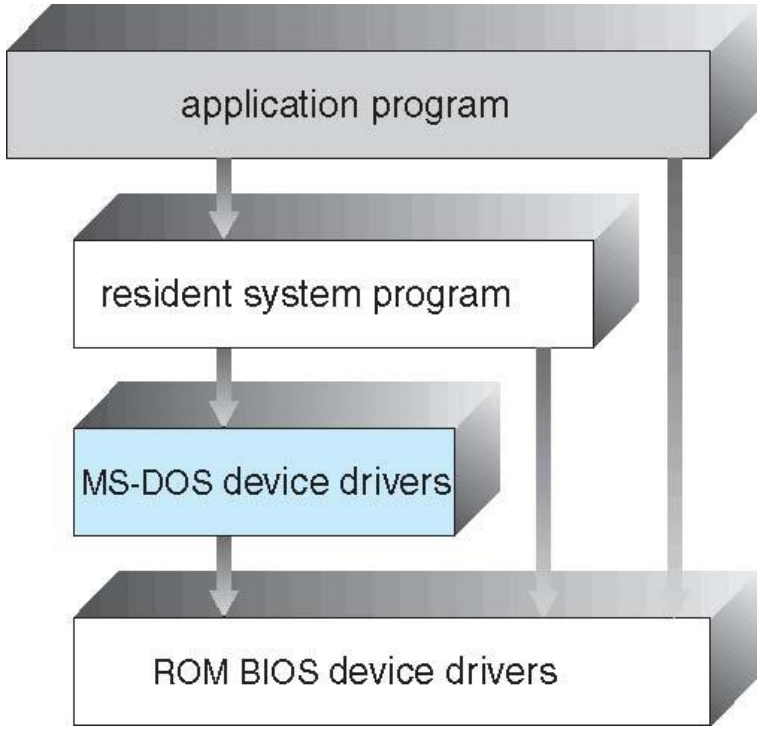


معماری سیستم عامل ها

- سیستم عامل های عمومی برنامه های بسیار بزرگی هستند و روش های مختلفی برای معماری آنها وجود دارد.
  - ساختار ساده (MS-DOS)
  - معماری پیچیده تر مانند یونیکس
  - معماری لایه ای
  - ریز هسته ها

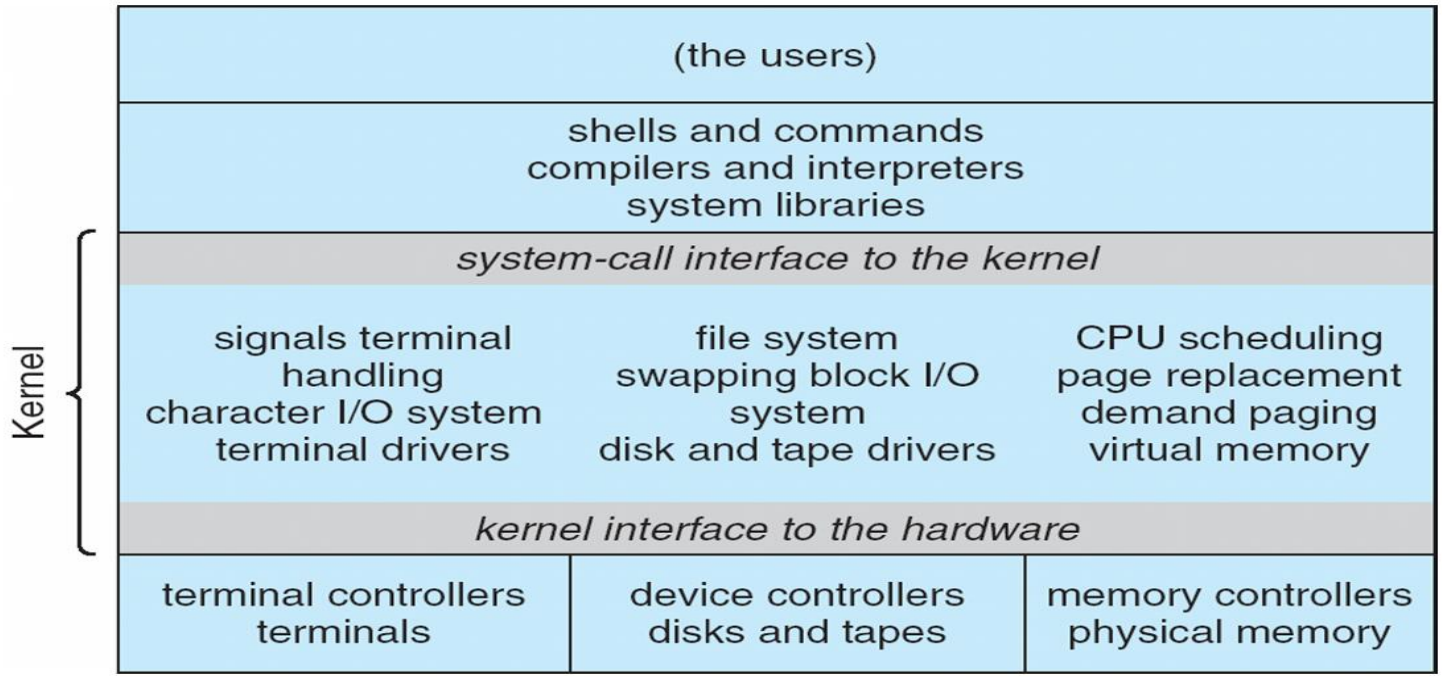
معماری ساده (MS-DOS)

- طراحی برای بیشترین عملکرد در کمترین فضا
  - بخش بندی پیمانه ای ندارد
  - هر چند ساختار دارد اما عملکرد لایه هایش خوش تعریف نیستند.
  - محافظت ندارد (محدودیت سخت افزار)



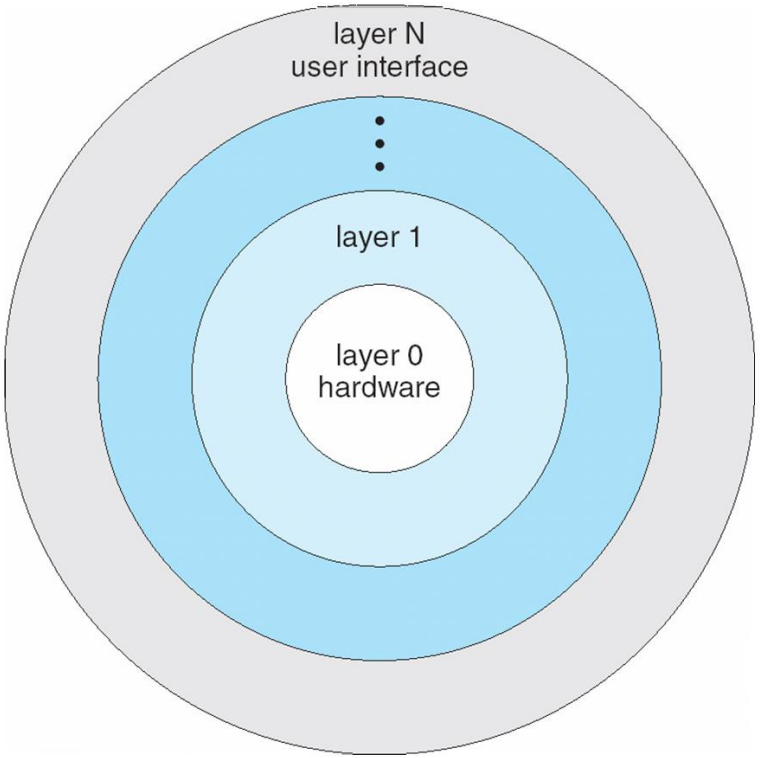
معماری ساده (Unix)

- محدودیت های سخت افزاری سبب محدودیت ساختار نسخه های اولیه بوده است.
- یونیکس از دو بخش تشکیل شده است.
  - برنامه های سیستمی
  - هسته
    - هر آنچه که زیر system call interface و روی سخت افزار است.
    - شامل مدیریت حافظه، برنامه ریزی پردازنده، سیستم فایل و ..



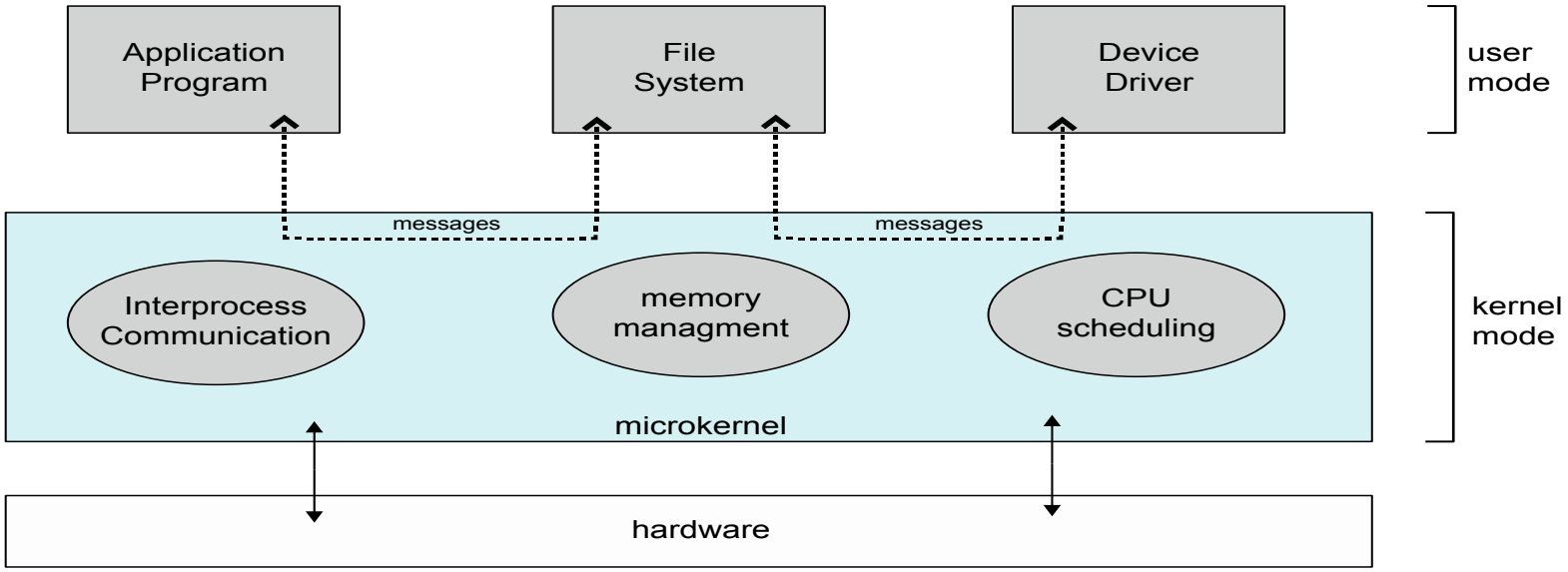
معماری لایه ای

- سیستم عامل به لایه های مختلف بخش بندی می شود. هر لایه از لایه پایین تر سرویس گرفته و به لایه بالاتر سرویس می دهد.
- برتری ها
  - پیاده سازی ساده
- کاستی ها
  - خوش تعریف نبودن لایه ها
  - سربار زیاد لایه ها و افت کارایی



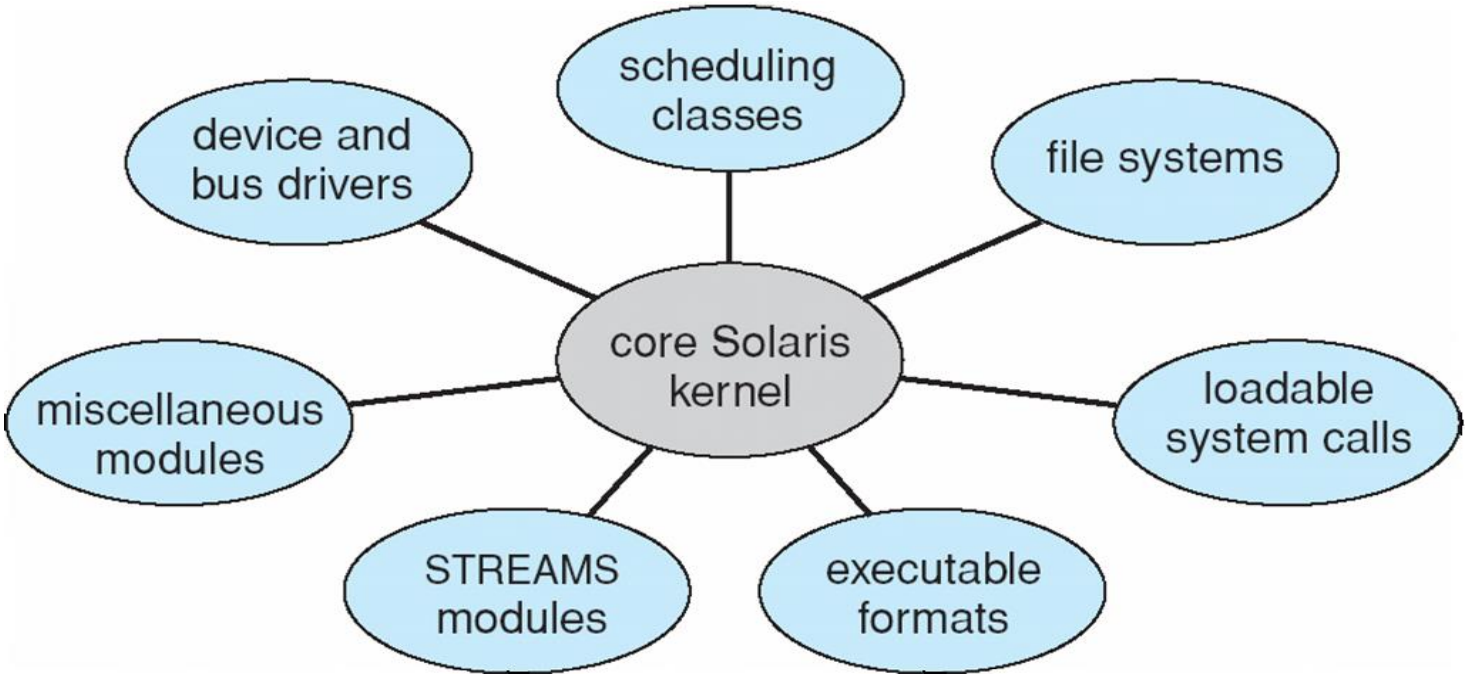
ریز هسته ها

- انتقال بیشتر بخش ها از هسته به فضای کاربر و پیاده سازی به صورت ماژول جدا
- ارتباط بین ماژول ها از طریق پیام
- برتری ها
  - گسترش ساده
  - انتقال ساده به سخت افزار های جدید
  - قابلیت اطمینان بیشتر بدلیل کوچک بودن هسته
  - امنیت بیشتر
- کاستی : سربار زیاد پیام ها



معماری پیمانه ای

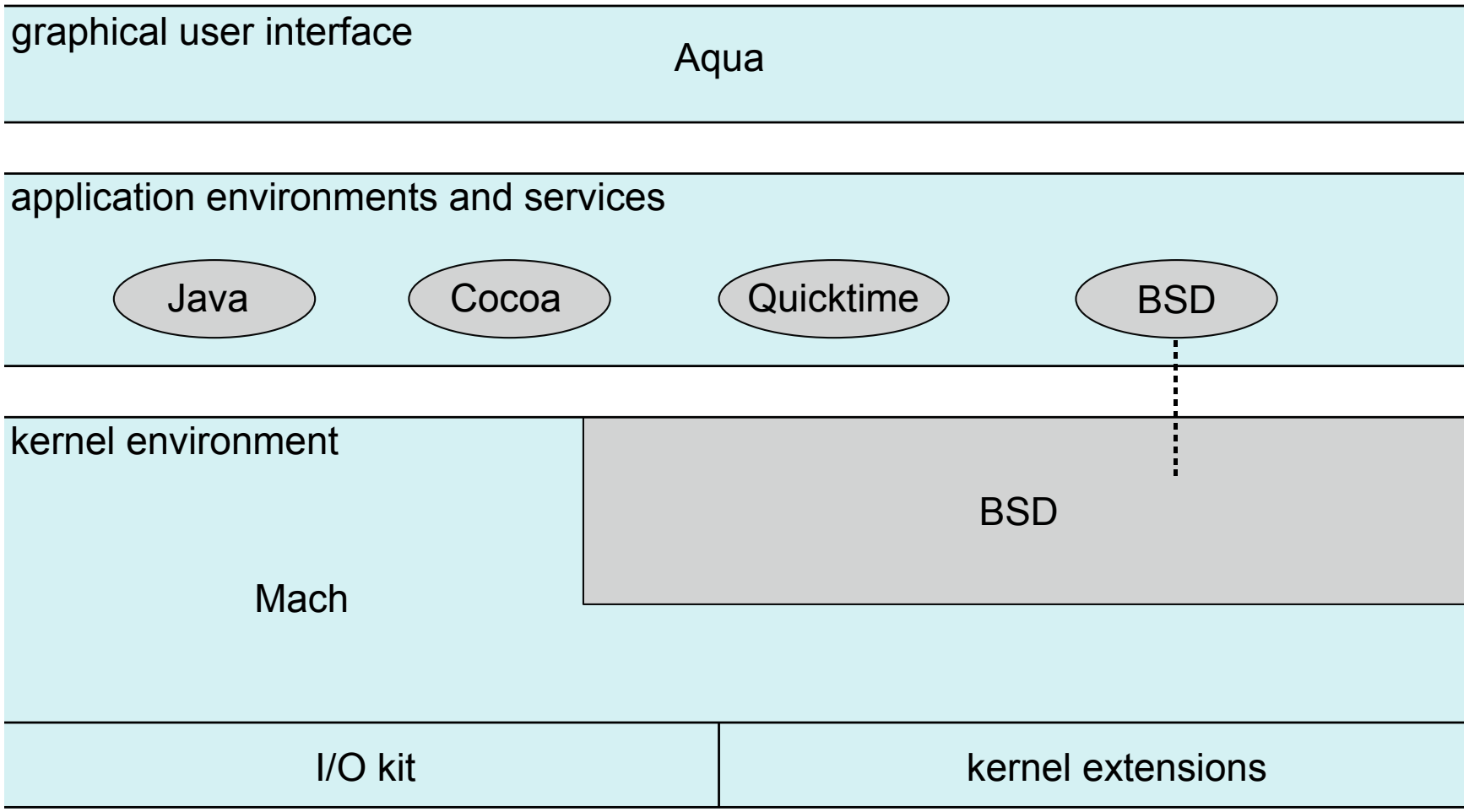
- سیستم عامل های مدرن از هسته های قابل بارگذاری استفاده می کنند
  - استفاده از روش طراحی شی گرا
  - جدا بودن هر مولفه
  - ارتباط با همدیگر از طریق واسطه
  - بارگذاری هنگام نیاز
- شبیه معماری لایه ای اما انعطاف پذیر تر (مانند Linux, Solaris)



سامانه های ترکیبی

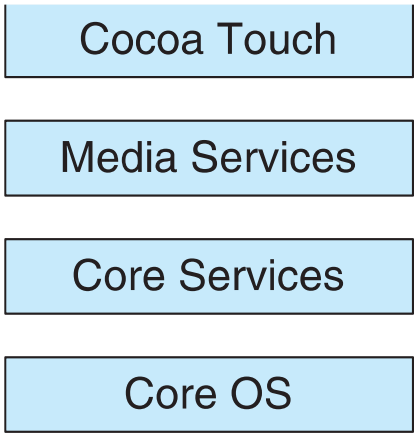
- بیشتر سیستم عامل های مدرن از معماری ترکیبی استفاده می کنند.
- ترکیب چند روش برای بهبود کارایی، امنیت و نیاز های کاربران
- برای نمونه هسته سیستم های عامل Linux and Solaris بزرگ هستند اما امکان بارگذاری پیمانه ها را دارد.
- سیستم عامل Windows از یک هسته بزرگ بهره می برد اما از معماری ریز هسته برای برخی از زیر سامانه ها استفاده می کند.
- سیستم عامل Mac OS از دو معماری لایه ای و ریز هسته بهره می برد.

MAC OS معماری



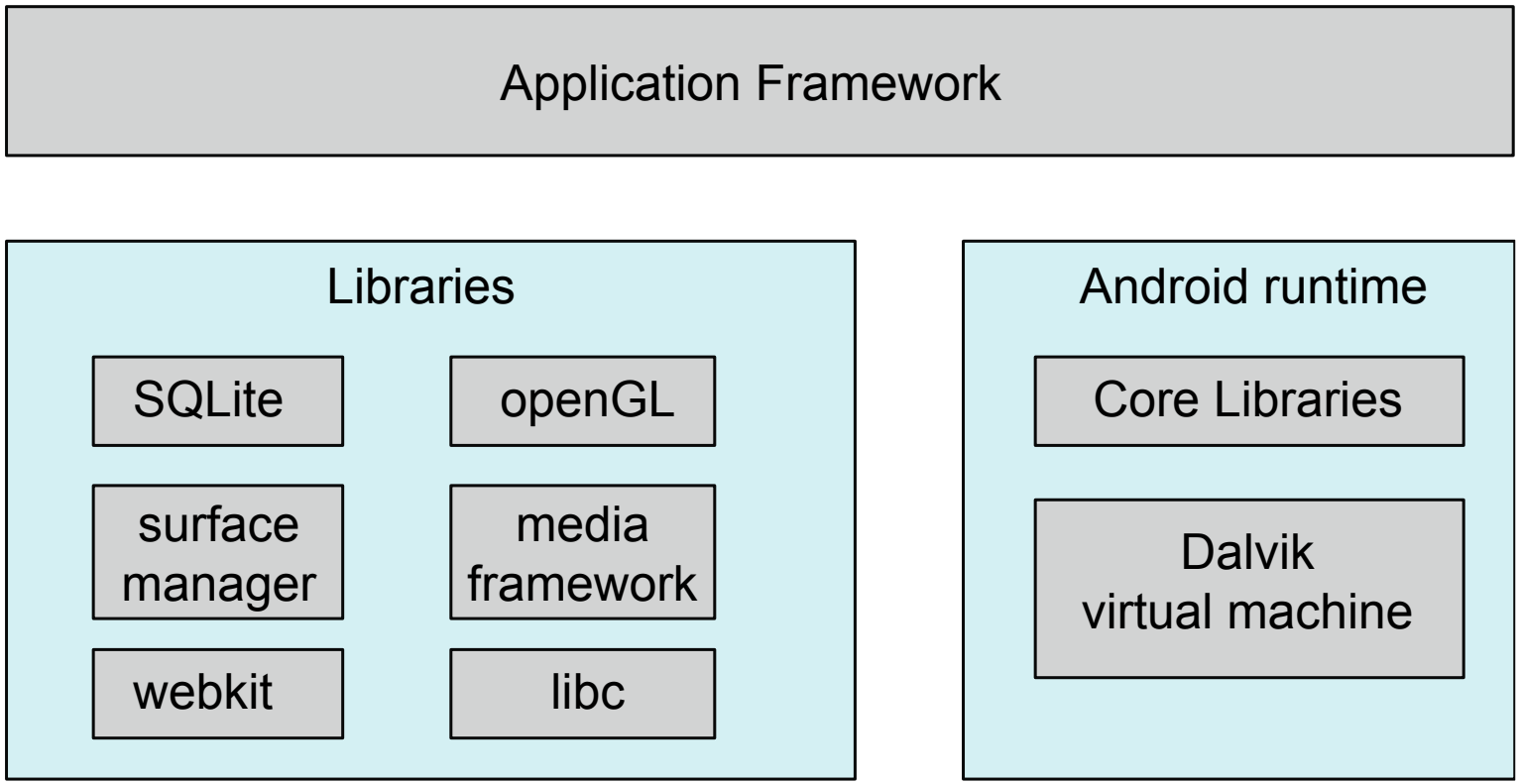
معماری iOS

- طراحی شده برای ***iPhone***, ***iPad***
  - بر مبنای Mac OS طراحی شده ام عملکرد هایی به آن افزوده شده است.
  - قابلیت اجرا روی سخت افزار های مختلف را دارد.

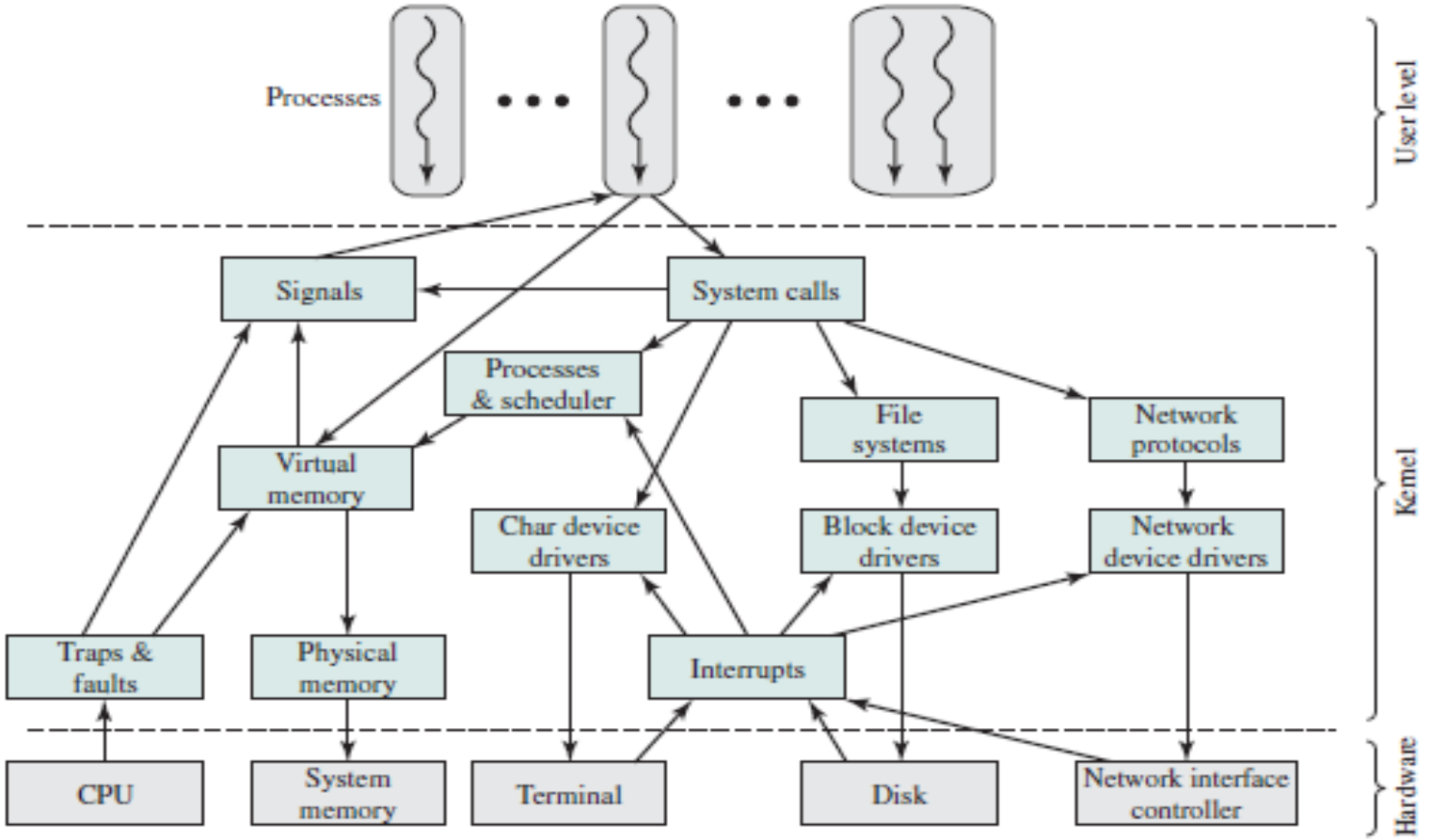


معماری Android

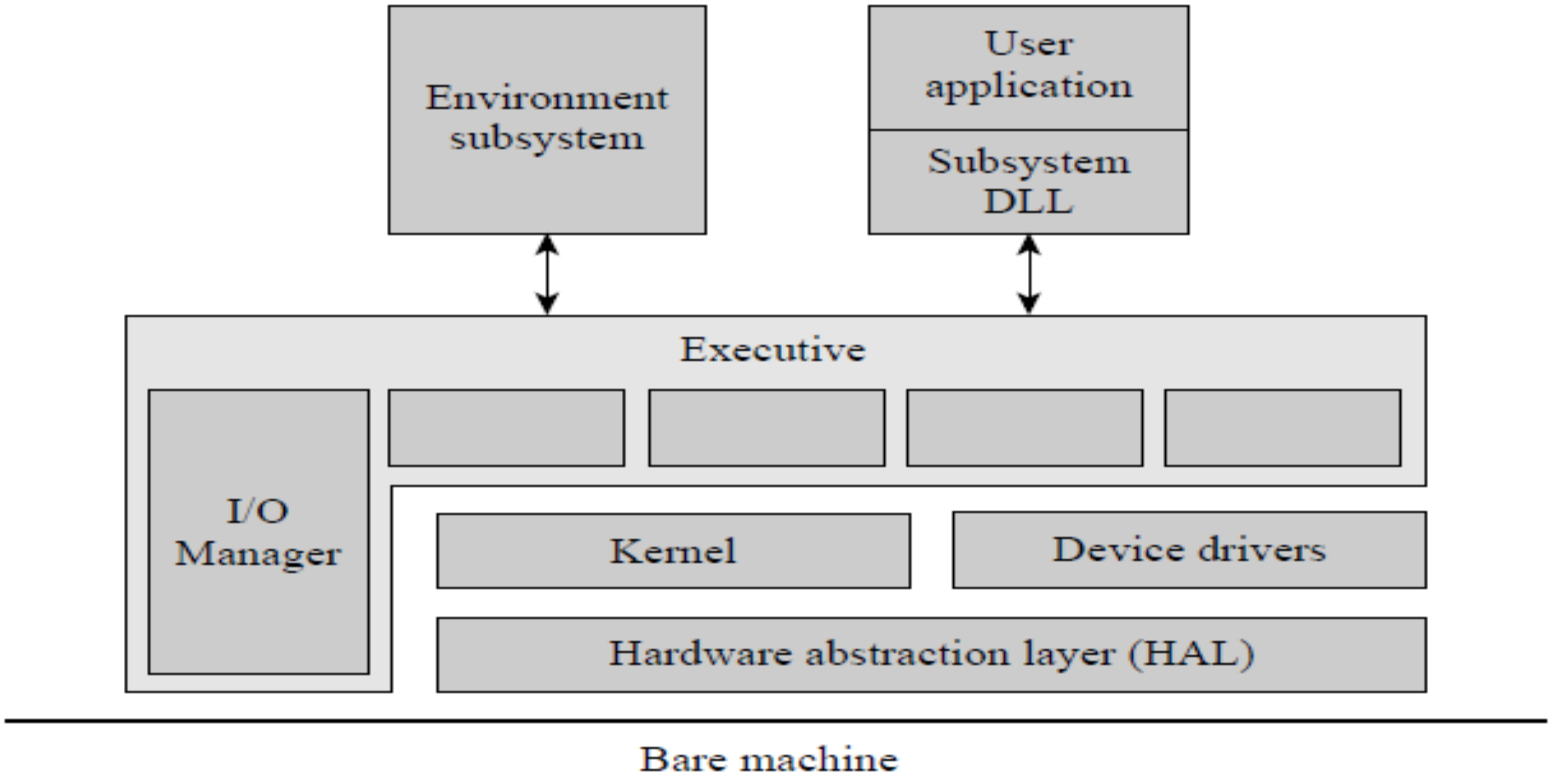
- براساس سیستم عامل لینوکس
- افزودن مدیریت توان
- افزودن ماشین مجازی Dalvik جهت اجرای برنامه های جاوا



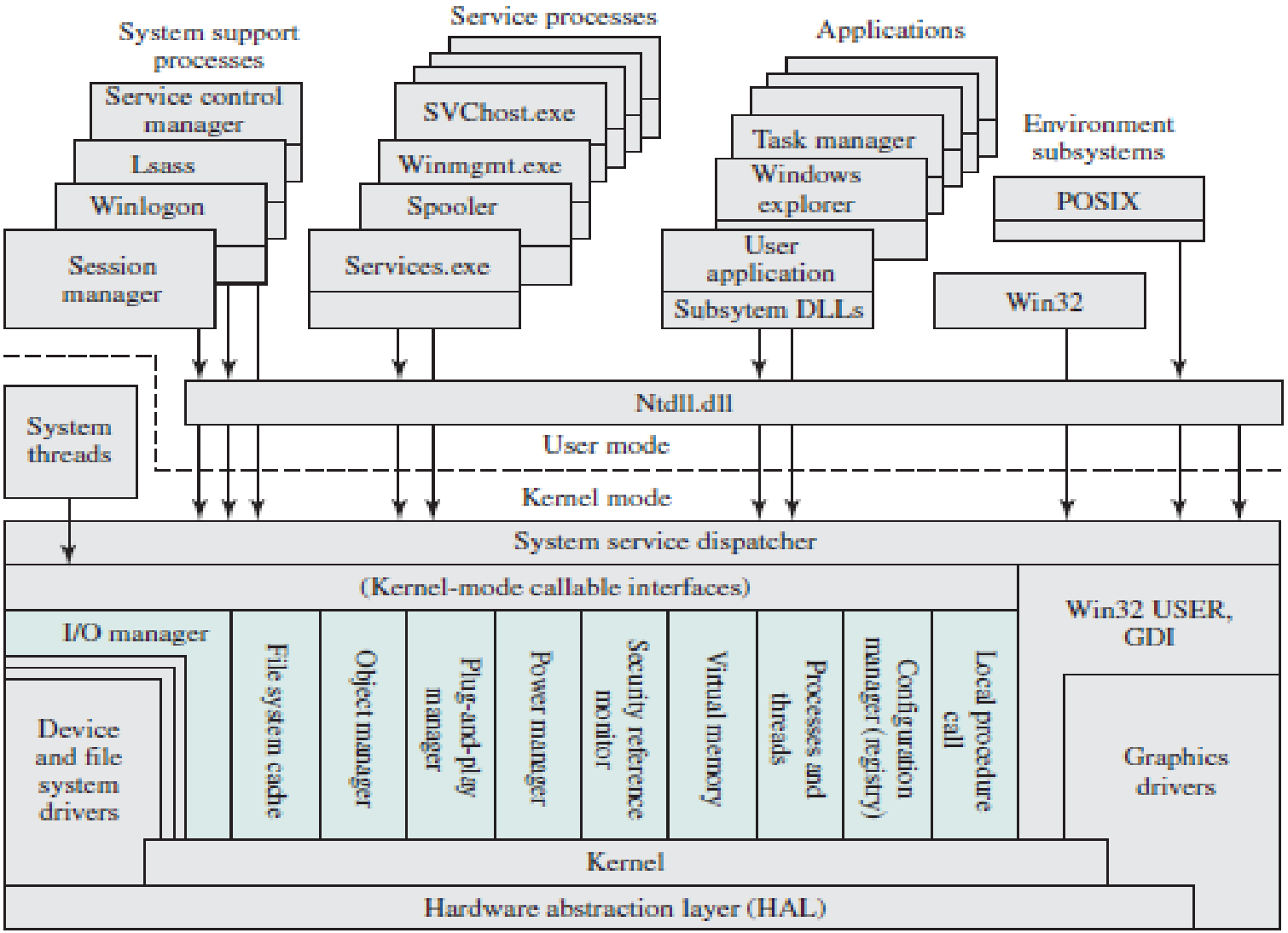
مولفه های لینوکس



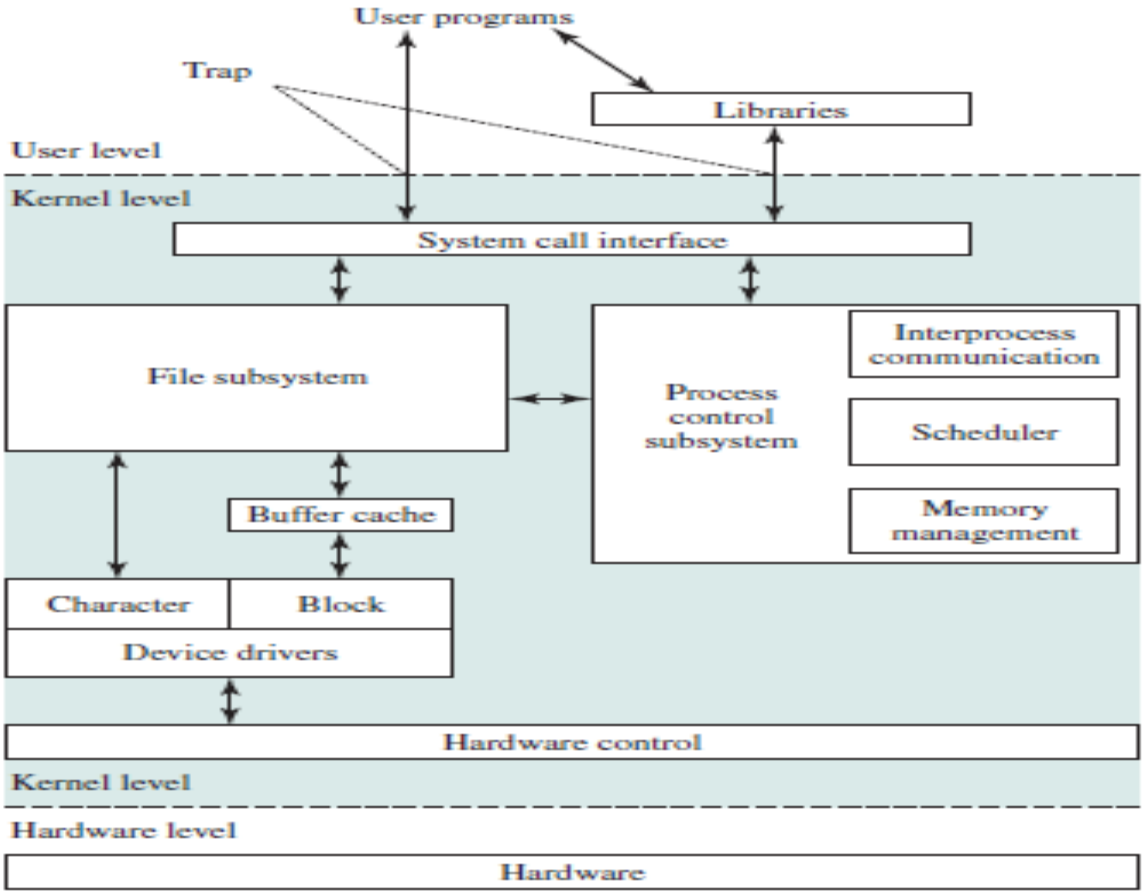
معماری سطح بالای Windows



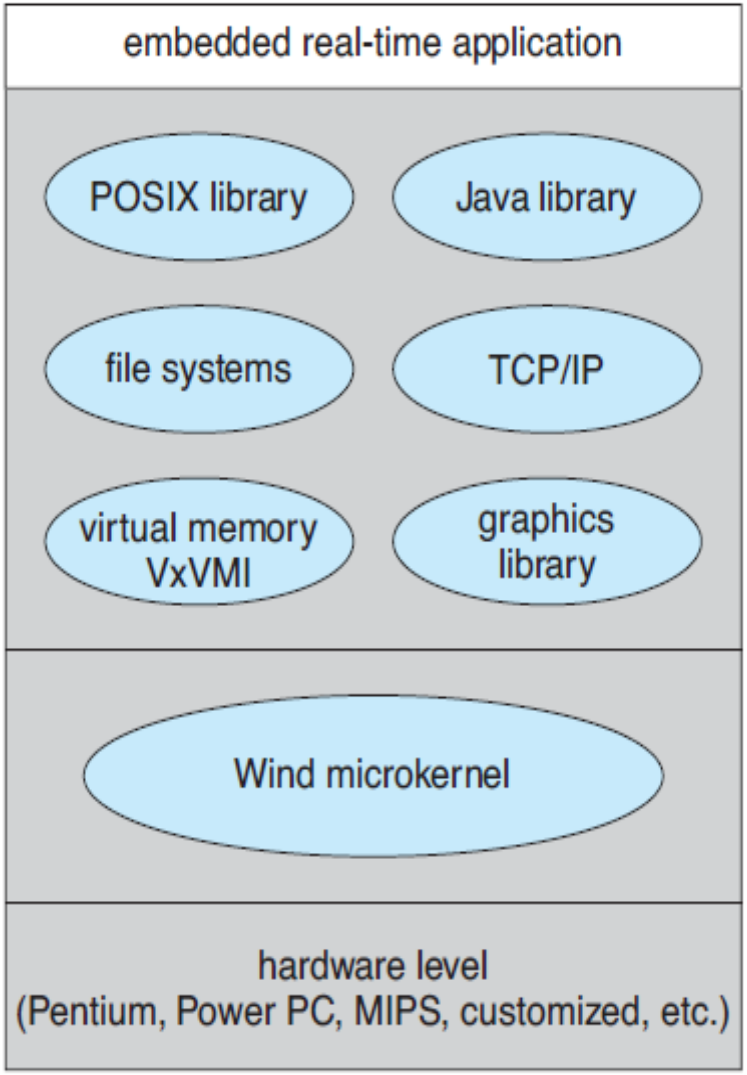
معماری Windows



معماری یونیکس

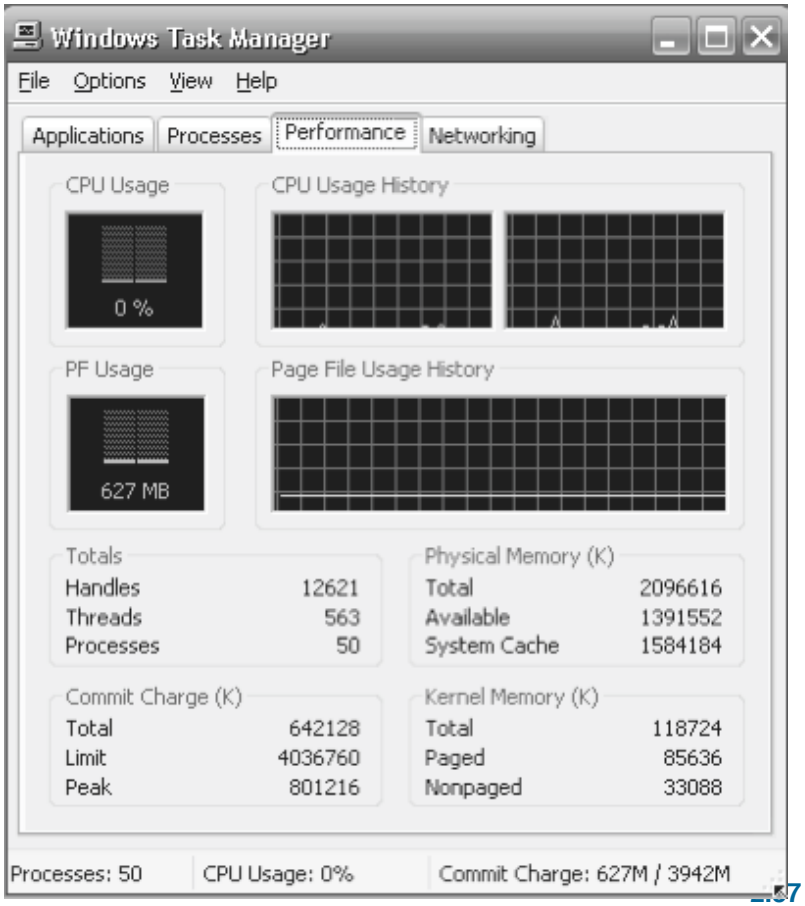


معماری VxWorks



اشکال زدایی سیستم عامل

- سیستم عامل فایل log مجزا تولید می نماید.
- در هنگام خطای یک برنامه می تواند اطلاعات حافظه پردازه و هسته را در اختیار کار بر از طریق dump file قرار دهد.
- بهبود کارایی هدف دیگری است که این اطلاعات کمک می کنند.



راه اندازی سیستم عامل

- در زمان روشن شدن سخت افزار، از آدرس خاصی اجرا آغاز می گردد.
- از ROM برای بارگذار اولیه استفاده می شود.
- بارگذار اولیه بارگذار سیستم عامل را بار گذاری نموده و آن را اجرا می نماید.
- بار گذار سیستم عامل ، سیستم عامل را بارگذاری می نماید.
- می توان از بار گذارهایی جهت انتخاب بار گذاری سیستم عامل استفاده نمود.

پایان فصل ۲