Arranque de sistemas embebidos. Lección 4

Prof.Ing. Jeferson González G.

CE-5303 Sistemas Embebidos

Área de Ingeniería en Computadores

Instituto Tecnológico de Costa Rica

1. Introducción

2. Arquitecturas de SW de SE
   - Firmware
   - Bootloader
   - Sistema Operativo
   - Middleware

3. Secuencia de arranque

Reutilización de componentes de Software

- No todos los componentes de un sistema empotrado necesitan ser diseñados desde cero.
- Componentes de Software que pueden ser reutilizados pueden ser:
  - Bibliotecas (*libs*)
  - Sistemas operativos
  - Firmware
  - Middleware
- Papel fundamental: **compilador**

Arquitecturas de Software

Computadora de Escritorio

Sistema Empotrado Complejo

Sistema Empotrado Simple

Firmware

- Software de más bajo nivel.
- Ejecutado por el procesador al arrancar el sistema para inicializar el hardware y preparar el entorno correctamente.
- Usualmente se encuentra grabado en una memoria ROM.
- Ejemplo: BIOS

Bootloader

- Es un programa que se ejecuta por el procesador y el cual lee el sistema operativo desde el disco o memoria no volátil y lo carga en RAM.
- El bootloader se encuentra en las computadoras de escritorio y en la mayor parte de sistemas empotrados

Ejemplos de bootloaders

	Licencia	Reside en	Arranca de	Arquitectura	Sistemas Operativos
GRUB	GPL	MBR	Disco Duro, CDROM, USB, red	i386, PowerPC, Sparc	Linux, Mac OS X, Windows
LILO	GPL	MBR	Disco Duro, CDROM, USB, red	i386	Linux, Windows
NTLRD	Propietaria	MBR	Disco Duro, USB	i386	Windows NT y XP
UBOOT *	GPL	Flash, MMC SD.	Flash, Disco Duro	PPC, ARM, AVR32, MIPS, x86, 68k, Nios, MicroBlaze, Blackfin	Linux

Sistemas operativos

Es software encargado de **administrar** los recursos de hardware de un sistema, para proveer un servicio al usuario

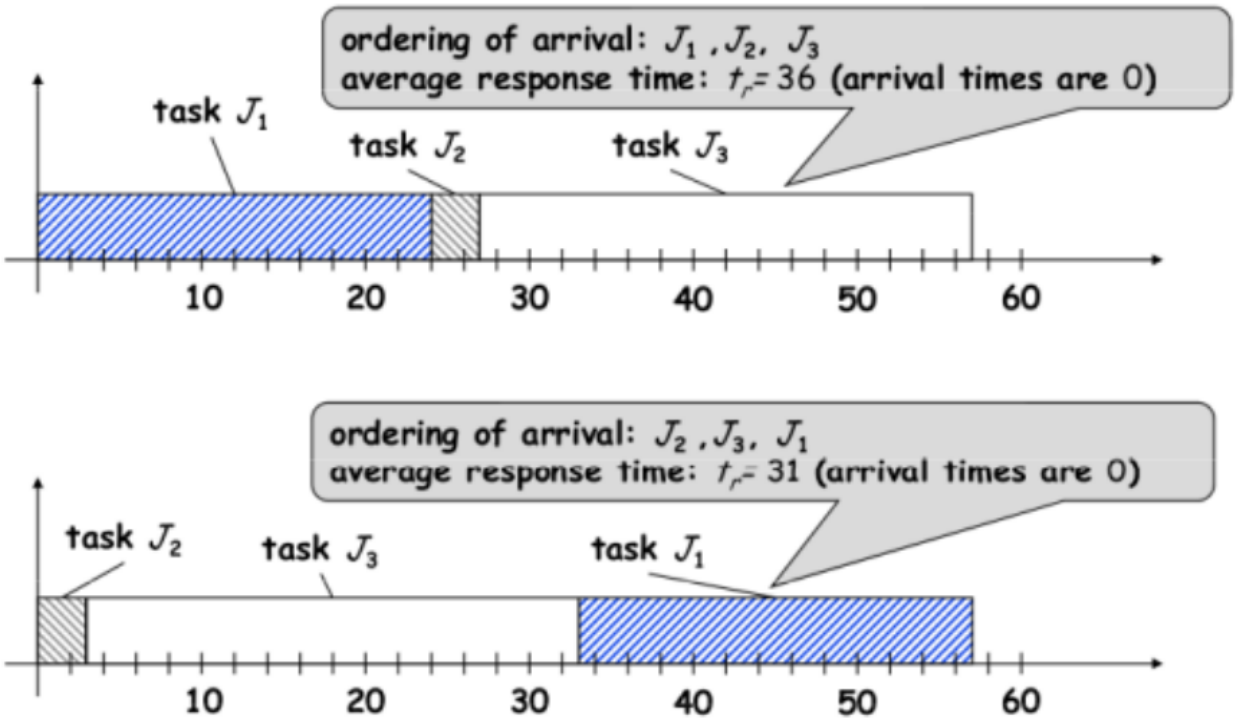
- Aspectos como la **calendarización**, intercambio de tareas y **el manejo de entradas y salidas** requieren el soporte de un sistema operativo apropiado.

Características de SO para Sistemas Embebidos

- **Configurabilidad**
- Los dispositivos pueden ser manejados por tareas/archivos
- Interrupciones no son exclusivas para el SO
- Capacidad en el manejo del tiempo real

Calendarización

Forma en que un SO asigna tareas al procesador para ser ejecutadas.



Modo de un sistemas operativo

Modo: Formas de uso del SO

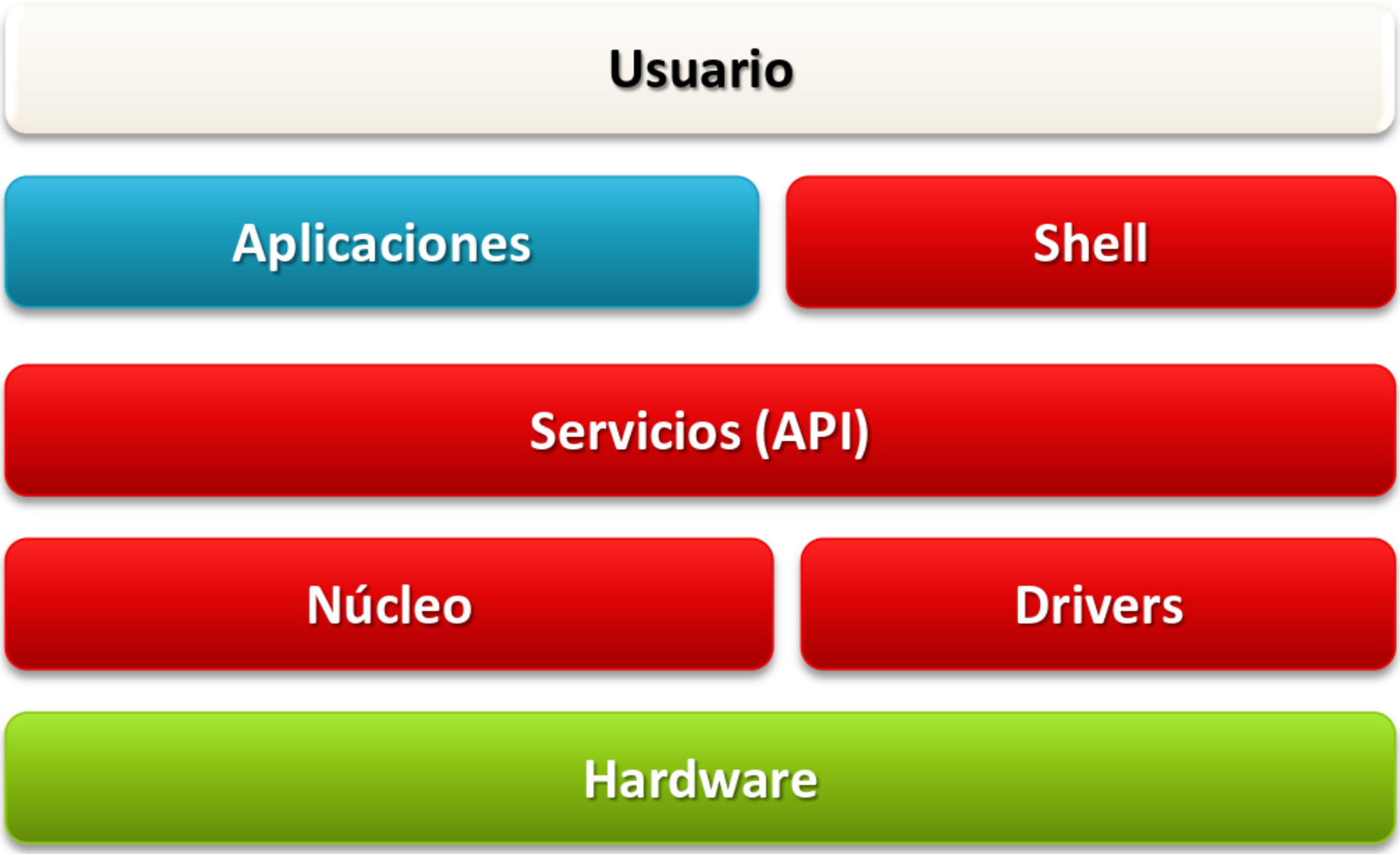
- **Modo Supervisor** o **Privilegiado**: Es utilizado dentro del sistema operativo. El SO utiliza **todas** las instrucciones del procesador, incluyendo aquellas que podría originar un fallo en el sistema.
- **Modo no privilegiado**: incluye instrucciones que un proceso simple pueda utilizar (nivel usuario)

Llamadas al sistema - System Calls

- Son rutinas que permiten que el SO esté disponible para un programa.
- La mayoría de los núcleos proporcionan bibliotecas en C que permiten la invocación de llamadas al sistema.

Permite utilizar comandos de terminal en un código fuente (.c)

Niveles de un sistema operativo



Núcleo - Kernel

Software responsable de **facilitar** a aplicaciones a**cceso al hardware** del computador. Es así como es el encargado de gestionar recursos del hardware, a través de servicios de llamada al sistema.

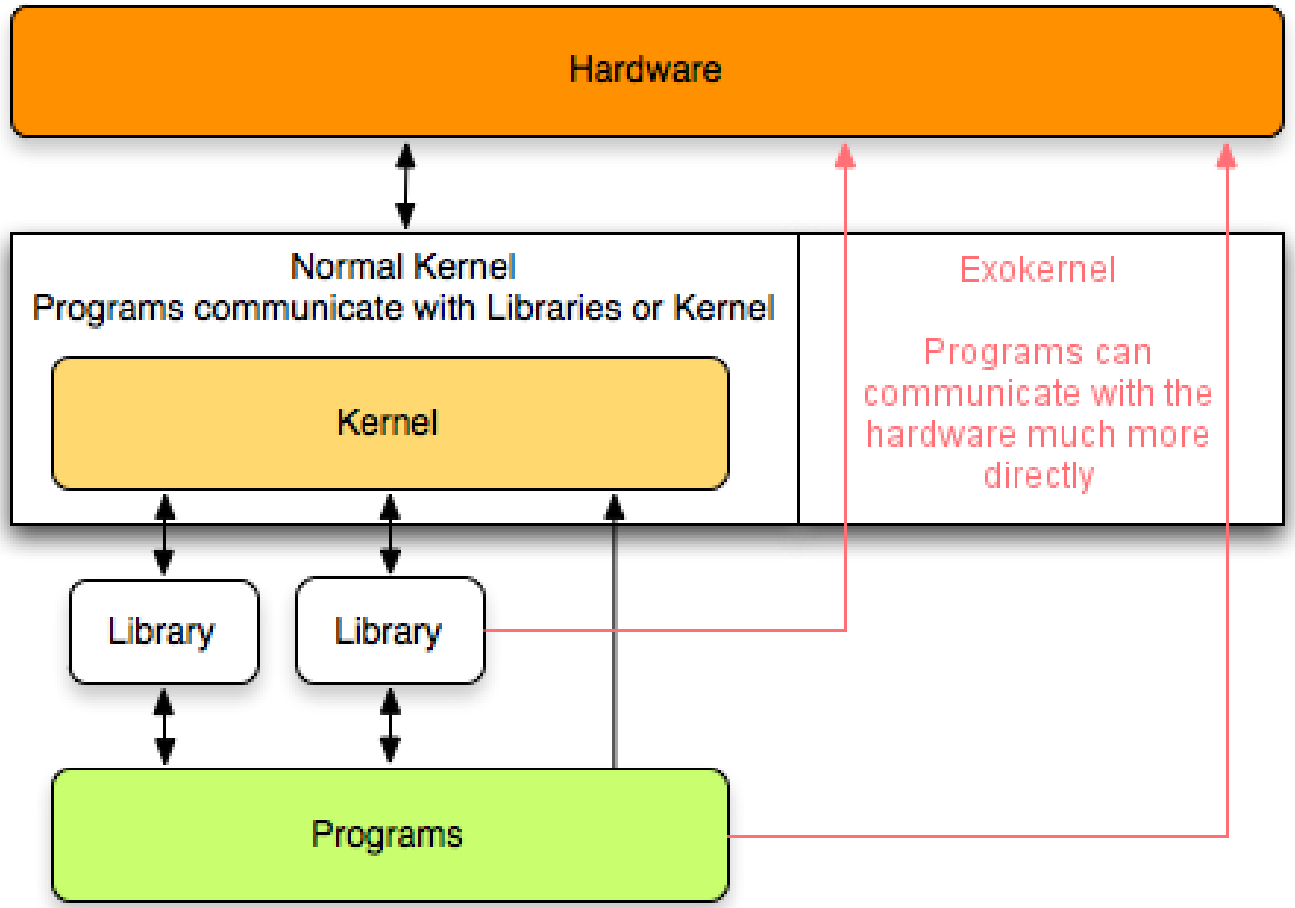
Tareas más importantes:

- Comunicación entre aplicaciones y hardware
- Gestión de tareas (calendarización)
- Gestión del hardware (memoria, procesador, periférico, forma de almacenamiento, entre otros)

Tipos de Kernel

- **Monolítico:** todos los servicios se ejecutan en espacio de memoria del kernel. - Linux
- **Microkernel:** conjunto mínimo de servicios, permite que se implementen otros en espacio de usuario. - QNX
- **Nanokernel:** el código mínimo, generalmente el kernel se reduce a drivers. El SO se implementa en espacio de usuario.
- **Exokernel:** implementa llamadas a sistema de un sistema operativo para simular otro o acceder a recursos de HW más eficientemente.

MIT- Exokernel OS



API en sistemas operativos

Conjunto de rutinas, estructuras de datos, protocolos y servicios del sistema operativo que pueden ser utilizados por aplicaciones.

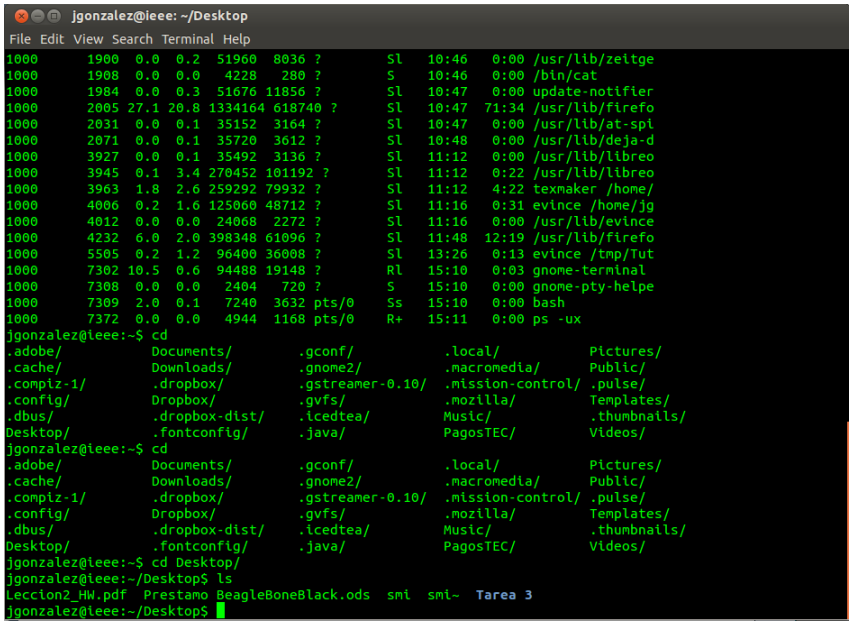
Permite que una aplicación pueda ser ejecutada en un determinado sistema operativo. Entre las funciones se encuentran:

- Depuración y manejo de errores
- E/S de dispositivos
- Manejo de memoria
- Interfaz de usuario

Shells

Parte del software del sistema operativo que le permite al usuario acceso a los servicios del kernel

- Intérprete de líneas de comandos
- 2 tipos:
  - Shells de líneas de comandos .
  - Shells con Interfaz gráfica de usuario (GUI).



Sistema de archivos

Método de almacenamiento y **organización** de datos cuyo objetivo es facilitar el acceso a ellos.

Diferentes tipos:

- de disco
- *flash***
- de red
- de propósito especial -Ej. swap

Ejemplos Sistema de Archivos

Sistema Operativo	Sistemas de Archivos
Mac 0SX	HFS Plus
Linux	Familia ext* (ext2, ext3) XFS JFS ReiserFS
Linux (Memoria flash)	JFFS2 YASSFS CRAMFS ROMFS
Windows	Familia FAT NTFS

Procesos

Un proceso es una abstracción de un programa dentro de otro (aplicación). Se constituye de:

- Código - C
- Estado - S

Un proceso además tiene su propio:

- Contador de Programa
- Espacio de memoria: **No** permite compartir memoria con otros procesos / aplicaciones

Ejemplo creación de proceso

```c
#include <sys/types.h> /* pid_t */
#include <sys/wait.h>  /* waitpid */
#include <stdio.h>     /* printf, perror */
#include <stdlib.h>    /* exit */
#include <unistd.h>    /* _exit, fork */

int main(void)
{
   pid_t pid = fork();

   if (pid == -1) {
      // When fork() returns -1, an error happened.
      perror("fork failed");
      exit(EXIT_FAILURE);
   }
   else if (pid == 0) {
      // When fork() returns 0, we are in the child process.
      printf("Hello from the child process!\n");
      _exit(EXIT_SUCCESS);  // exit() is unreliable here, so _exit must be used
   }
   else {
      // When fork() returns a positive number, we are in the parent process
      // and the return value is the PID of the newly created child process.
      int status;
      (void)waitpid(pid, &status, 0);
   }
   return EXIT_SUCCESS;
}
```

Hilos - Threads

Un hilo también es una abstracción de un programa, mediante una ejecución concurrente, que **permite** compartir memoria.

- Los hilos pueden accesar variables de otros hilos

¿Por qué utilizar hilos?

- Su creación y terminación es más rápida
- Existe posibilidad de compartir el espacio de memoria
- Existe posibilidad de explotar un mejor paralelismo
- Posibilidad de hacer un mejor uso de los multiprocesadores

Sistema Operativo de Tiempo Real - RTOS

Sistema operativo que permiten la construcción o implementación de sistemas / aplicaciones en tiempo real.

- Un RTOS está diseñado para cumplir tareas en **deadlines** específicos.

Ejemplos:

- **FreeRTOS**
- LynxOS
- QNX
- VxWorks

Middleware

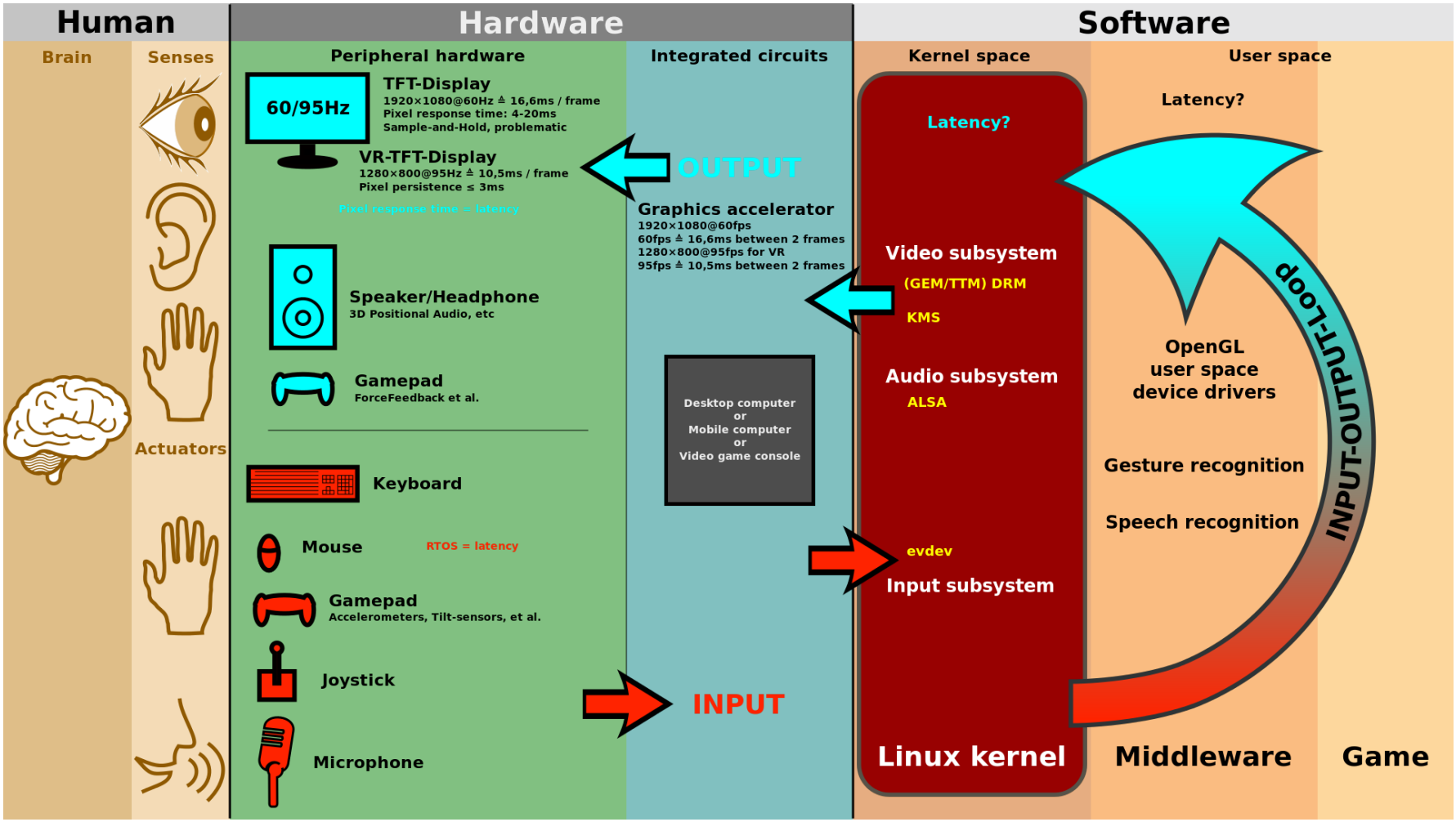
Capa intermedia (**bibliotecas**) entre el sistema operativo y la aplicación de software .

- Android
- OpenGL
- OpenMP
- MPI

Puesto que se localizan por encima del SO, quiere decir que pueden ser independientes del mismo, y por tanto del hardware que yace por debajo.

(¡Portabilidad!)

Ejemplo

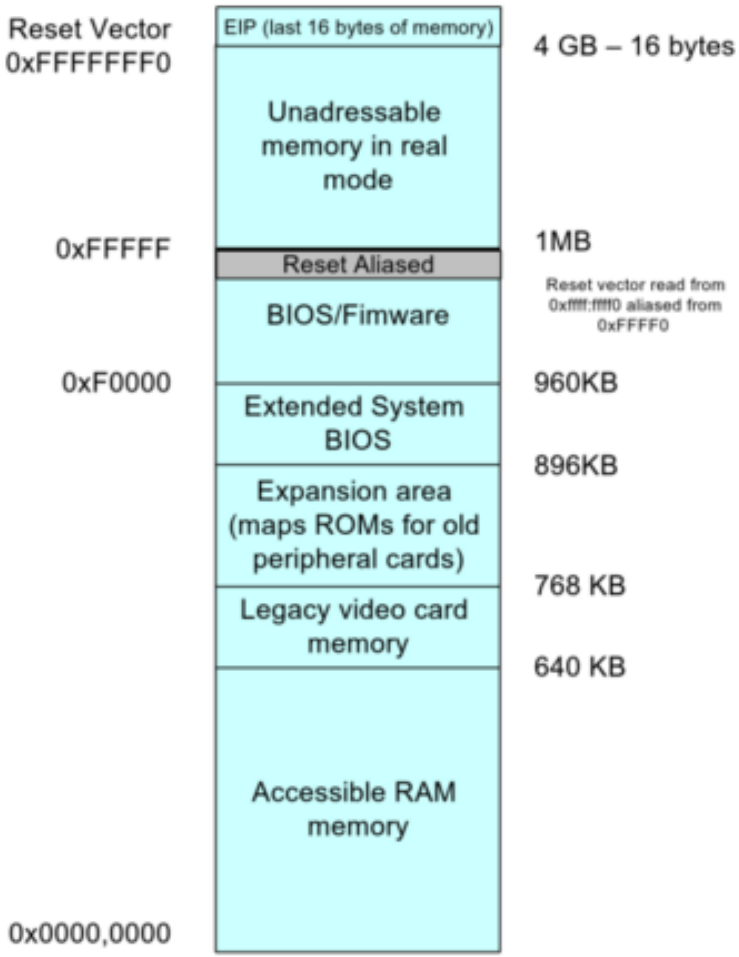

Human
Brain
Senses
Actuators
Hardware
Peripheral hardware
Integrated circuits
60/95Hz
TFT-Display
1920×1080@60Hz ≙ 16,6ms / frame
Pixel response time: 4-20ms
Sample-and-Hold, problematic
VR-TFT-Display
1280×800@95Hz ≙ 10,5ms / frame
Pixel persistence ≤ 3ms
Pixel response time = latency
OUTPUT
Graphics accelerator
1920×1080@60fps
60fps ≙ 16,6ms between 2 frames
1280×800@95fps for VR
95fps ≙ 10,5ms between 2 frames
Speaker/Headphone
3D Positional Audio, etc
Gamepad
ForceFeedback et al.
Desktop computer
or
Mobile computer
or
Video game console
Keyboard
Mouse
RTOS = latency
Gamepad
Accelerometers, Tilt-sensors, et al.
Joystick
Microphone
INPUT
Software
Kernel space
User space
Latency?
Latency?
Video subsystem
(GEM/TTM) DRM
KMS
Audio subsystem
ALSA
evdev
Input subsystem
Linux kernel
OpenGL
user space
device drivers
Gesture recognition
Speech recognition
Middleware
Game
INPUT-OUTPUT-LOOP

Arranque

Secuencia de arranque:

- Reset
- Firmware / BIOS
- Bootloader
- Sistema operativo
- Aplicación

Mapa de Memoria



Conceptos

La secuencia de booteo de sistemas embebidos varía poco entre diferentes dispositivos. Algunos conceptos clave en sistema multiprocesador son:

- **Hilo** (Thread): procesador lógico que comparte recursos con otro procesador lógico en un núcleo. Ejemplo: Intel SMT, suele tener 2 hilos por núcleo.
- **Núcleo** (core): Un núcleo (físico) puede existir con otros núcleos en un solo paquete. Los recursos no suelen compartirse entre núcleos a excepción de *cache*.
- **Paquete**: Chip que contiene uno o más núcleos.
- **Sistema en Chip** (SoC): Paquete que contiene CPUs/núcleos junto con controladores de memoria, cache, coprocesadores, controladores E/S, etc.

Secuencia de arranque en multiprocesadores

La secuencia contiene los siguientes pasos:

- Arbitraje del hardware
- Inicialización dispositivos (BSP)
- Inicialización dispositivos (AP)
- Ejecución BIOS/firmware
- Arranque S.O

Arbitraje del HW

Al generarse una señal de reset se inicia un proceso de arbitraje de hardware. Para facilitar el proceso de arranque, los procesadores se dividen en:

- Procesador *Bootstrap* (BSP): principal encargado de arranque de sistema
- Procesador de aplicación (AP): el resto de los procesadores

Inicialización de dispositivos- BSP

Al ser seleccionado, el BSP **busca** las instrucciones del vector de reset:

- Establece bit de BSP en controlador local de interrupciones.
- Arranca inicialización de estructuras globales para BIOS/firmware.
- Inicializa controlador de interrupciones general (APIC).
- AP's realizan *self-tests* y esperan inicio (wait for start up inter processor interrupt (WAIT-for-SIPI)
- Al finalizar la inicialización, el BSP envía un mensaje SIPI (IPC) a los APs con los vectores de inicialización de BIOS de cada procesador.

Inicialización de dispositivos - AP's

Al recibir el mensaje, cada AP debe

- Tomar semáforo de inicialización de BIOS.
- Agregar ID de controlador de interrupciones a tablas de información de sistema ACPI (Advanced Configuration and Power Interface) - nivel de firmware.
- Aumenta contador de número de procesadores.
- Liberar semáforo de inicialización de BIOS.
- Limpiar su controlador de interrupciones
- Apagarse

Ejecución BIOS/firmware

Una vez que cada AP termina su inicialización. El BSP procede a ejecutar las rutinas del BIOS para la inicialización y prueba de los dispositivos E/S, memoria, controladores, interfaces, recursos de SoC, etc.

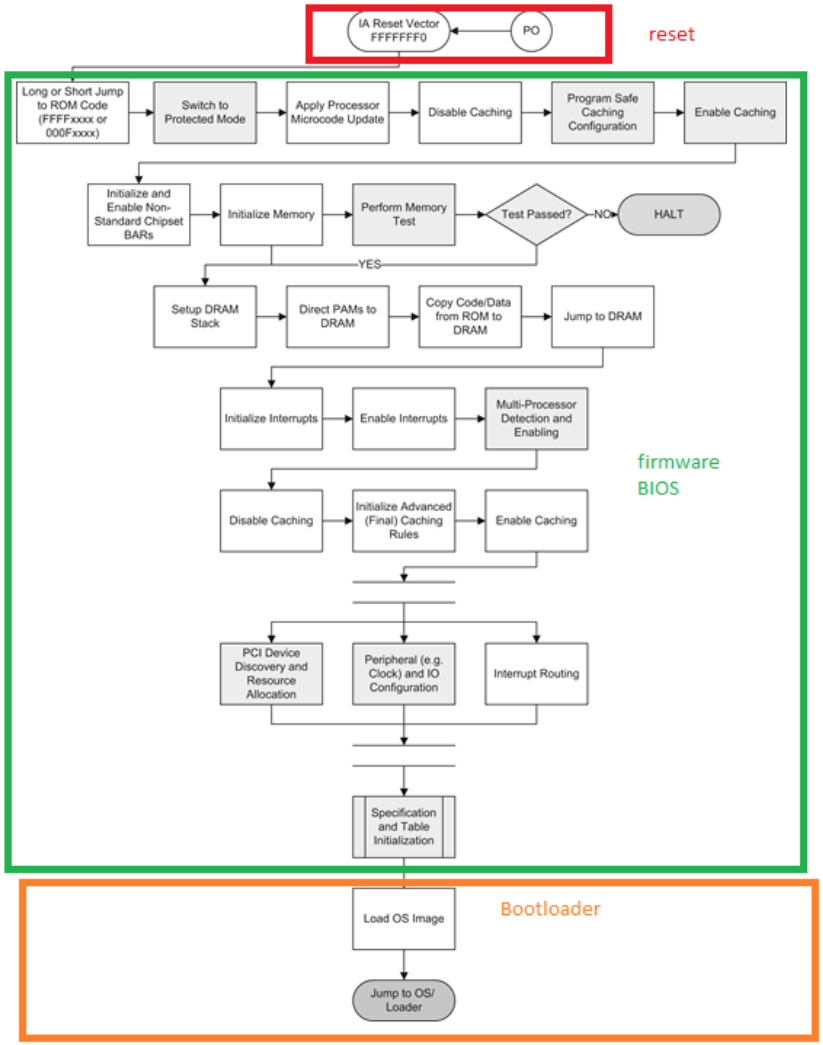
- Al finalizar, se inicia la ejecución del S.O.

Arranque de S.O

Cuando el sistema está listo para arrancar el S.O (por medio del bootloader):

- El BSP inicia los demás procesadores (modo WAIT-for-SIPI), por medio de mensajes IPC.
- En este punto, el S.O toma control de los procesadores para la ejecución de sí mismo y demás aplicaciones.

Secuencia de arranque detallada



Referencias

Peter Barry and Patrick Crowley

Modern Embedded Computing: Designing conected, prevasive, media-rich systems

Miguel Angel Aguilar U. (2009)

Material de clase: Introducción a los Sistemas Embebidos

María Haydeé Rodríguez B. (2014)

Material de clase: Sistemas Empotrados

Kaashoek, M. Fransz B. (1995)

Exokernel: An Operating System Architecture for Application-Level Resource Management