Programación para aplicaciones de tiempo real

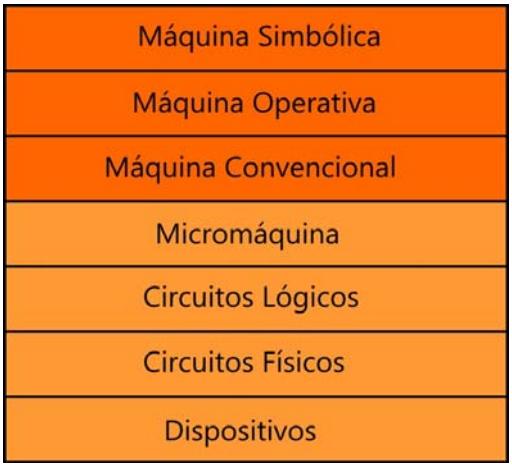
- Conceptos básicos sobre computadores y soporte físico
- Lenguajes de programación y sistemas operativos
- Conceptos de lenguaje C
- Normas POSIX para tiempo real

Definición de computador

- Computador:
  - "Máquina programable de propósito general que procesa datos"
  - El computador puede verse como una superposición de "máquinas virtuales"
  - Cada m.v. corresponde a un nivel de detalle, con su lenguaje y estructura

Niveles de máquina

- Diferentes niveles de detalle o abstracción
- Cada nivel tiene un lenguaje y una estructura



Niveles de máquina

- Máquina convencional (puede ser la “real”)
  - Lenguaje: lenguaje de máquina
  - Estructura: Modelo de Von Neumann
- Máquina operativa
  - Máquina convencional + sistema operativo
  - Lenguaje: Lenguaje de máquina + servicios s.o.
  - Estructura: Módulos del s.o.
- Máquina simbólica
  - Lenguaje y estructura: Lenguaje de alto nivel (C, Ada95) y modelo de máquina que implica

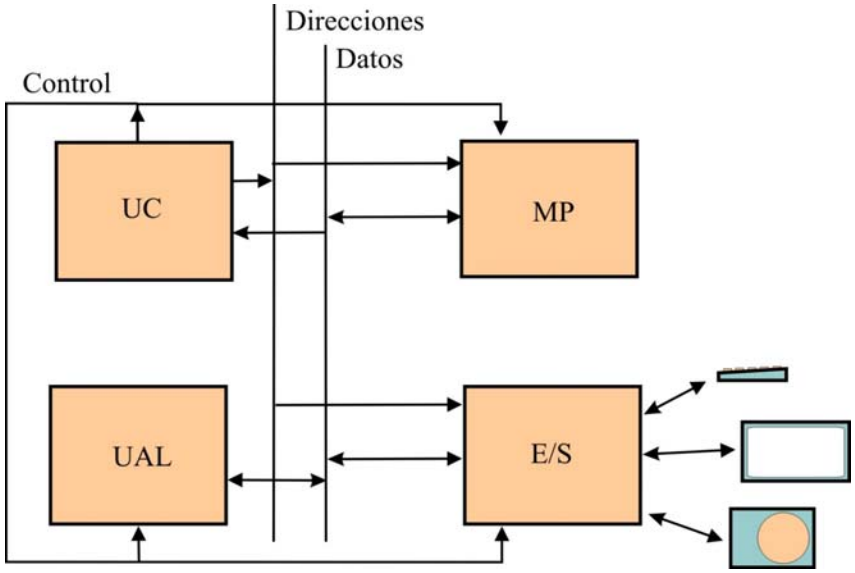
Nivel de Máquina Convencional

- Lenguaje: Lenguaje de máquina
  - Codificación binaria de programa y datos
  - Programa almacenado en memoria
- Estructura: Arquitectura de Von Neumann
  - Memoria principal (MP)
  - Unidad de control (UC)
  - Unidad Aritmético-lógica (UAL)
  - Unidad de Entrada/Salida (E/S)

Arquitectura de Von Neumann

- Memoria Principal
  - Contenido: datos o instrucciones
  - Conceptos básicos: Dirección, palabra, operaciones de lectura y escritura
- Unidad de Control
  - Lee e interpreta instrucciones
- Unidad Aritmetico-lógica
  - Dispone de circuitos lógicos para ejecutar instrucciones
- Unidad de Entrada/Salida
  - Comunica el computador con el exterior
  - Circuitos de interfaz específicos

Arquitectura de Von Neumann



Registros

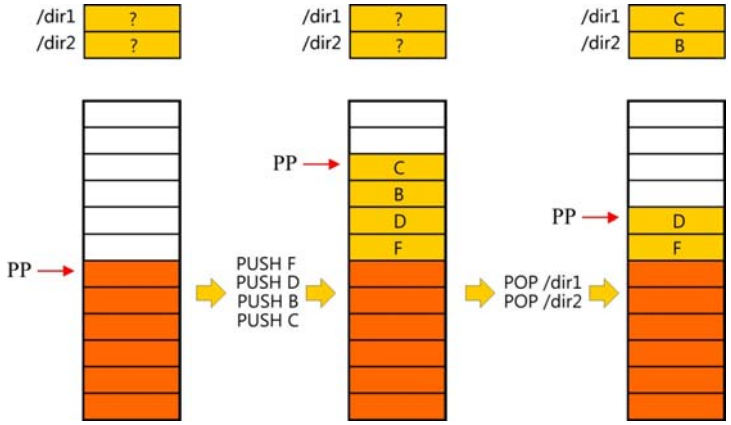
- Incluidos en UAL, UC
- Acceso diferente al de la memoria
- Registros aritméticos
- Registros de direccionamiento
- Contador de programa (CP)
- Puntero de pila (PP)
- Otros registros en dispositivos de E/S

Pila del sistema

- Problemas a resolver en las llamadas a función:
  - Saltar a la función (trivial)
  - Volver al punto de llamada (dirección de retorno variable…)
  - Paso de argumentos y resultados
- Pila (“stack”) de llamadas a función o pila del sistema:
  - Paso de la DDR a la función, y además…
  - Paso de argumentos y resultados
  - Creación de variables locales
  - Copia de registros

Pila del sistema

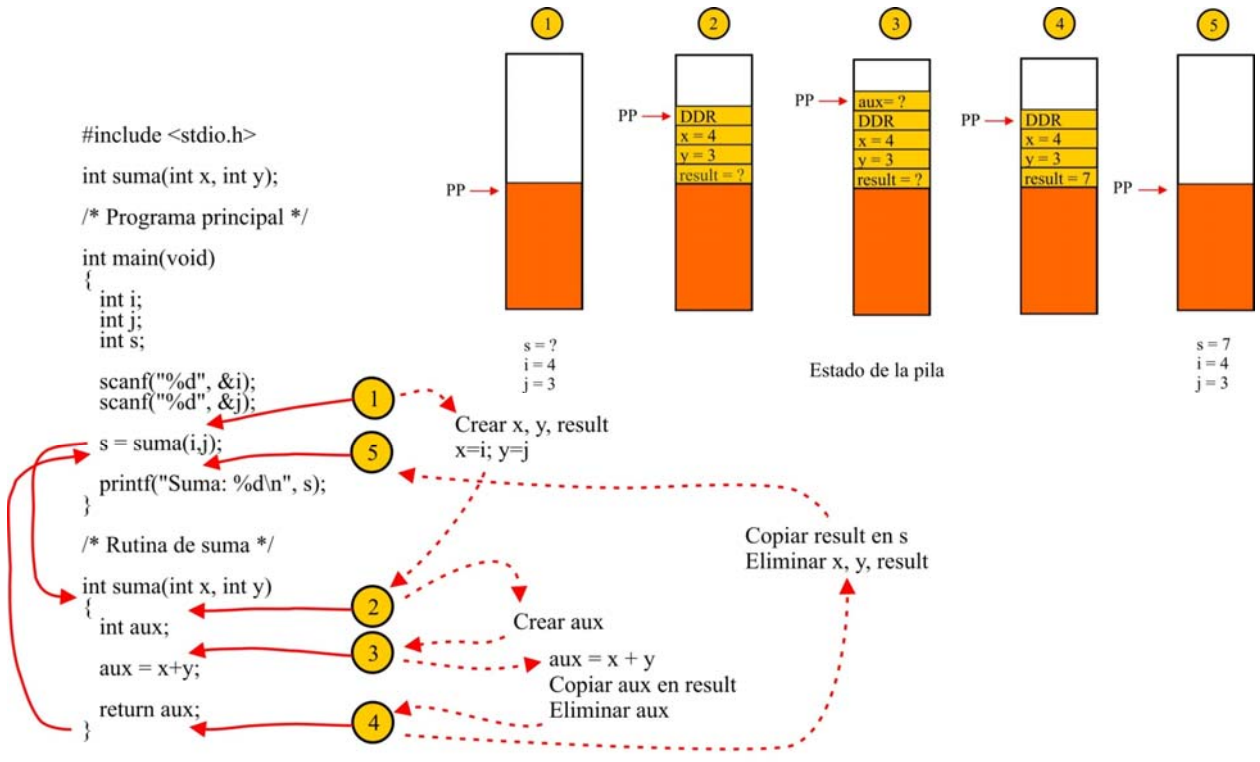
- Implementación de la pila (“stack”):
  - Zona de memoria reservada
  - Registro dedicado (puntero de pila - PP) para definir la cima
  - Instrucciones de máquina que utilizan PP
  - Diferentes opciones



Pila del sistema

- Almacenamiento y extracción de datos:
  - PUSH <valor>
    - PP = PP – 1
    - M(PP) = <valor>
  - POP /dir
    - M(/dir) = M(PP)
    - PP = PP + 1
- Subrutinas - funciones:
  - CALL /dir:
    - Almacena la DDR en la pila y salta a /dir
    - PP = PP - 1
    - M(PP) = DDR
    - CP = /dir
  - RET:
    - Recupera la DDR de la pila y salta a DDR
    - CP = M(PP)
    - PP = PP + 1

Pila del sistema



Pila del sistema

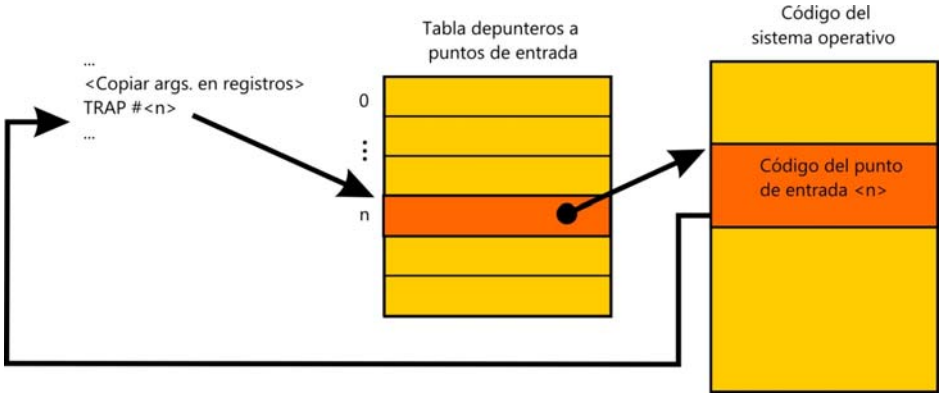
- Marco de pila:
  - Cada función que es invocada crea una estructura de datos propia ("stack-frame" – marco de pila): Argumentos, DDR, variables locales
  - Al salir, los marcos de pila van desapareciendo por orden
  - Esta solución permite recursividad
- Para crear actividades concurrentes es fundamental conservar (entre otros):
  - El contador de programa
  - El puntero de pila. Si no, no pueden mantenerse secuencias de llamadas a función separadas

Sistema operativo

- Máquina extendida
  - Añade instrucciones a la máquina convencional ("llamadas" o servicios del S.O.)
  - Ofrece soluciones de alto nivel para problemas complejos (E/S, gestión de archivos)
  - Permite multitarea creando varias "máquinas virtuales"
- Administrador de recursos
  - Gestiona los recursos del computador: Tiempo de CPU, acceso a E/S, memoria
  - Multiplexa (reparte) recursos en el espacio y en el tiempo

Sistema operativo

- Implementación habitual
  - Parte crítica (núcleo) asociada a un nivel de funcionamiento “privilegiado” del hardware
  - Llamadas mediante instrucciones de máquina especiales (TRAP, INT…) que cambian de nivel de privilegio y de entorno controladamente



Sistemas operativos y tiempo real

- Varias opciones de implementación:
  - Construir un sistema de propósito específico
  - Usar un lenguaje de alto nivel especializado y su sistema de soporte en tiempo de ejecución (por ejemplo ADA)
  - Usar los servicios de un S.O: mas un lenguaje convencional
- Los sistemas operativos deben proporcionar:
  - Soporte para concurrencia (multitarea)
  - Servicios de planificación (“scheduling”)
  - Servicios de temporización
  - Servicios de comunicación y sincronización
- La mayor parte de los sistemas operativos parecen proporcionar lo necesario; ¿puede usarse cualquiera de ellos?

Un poco de historia

- 1ª Generación (1945-55, válvulas): Sólo existe el nivel de máquina convencional
- 2ª Generación (1955-65, transistores):
  - Primeros lenguajes: FORTRAN
  - Proceso en tandas: s.o. elemental. Objetivo: Eficiencia
- 3ª Generación (1965-80, SSI, MSI)
  - Multiprogramación (OS-360 de IBM). Objetivo: Eficiencia (aprovechar tiempo de E/S)
  - Tiempo compartido (MULTICS, UNIX). Objetivo: Reparto equitativo de la potencia de cálculo entre usuarios
- 4ª Generación (1980-?, LSI)
  - Microprocesadores y computadores personales
  - Interconexión en red → s.o. en red y distribuidos

Sistemas operativos y tiempo real

- Objetivos de diseño de los sistemas operativos convencionales:
  - Eficiencia del uso del computador (maximizar computación por unidad de tiempo, o “throughput”)
  - Reparto equilibrado de la máquina entre aplicaciones y usuarios
- Objetivo de los sistemas informáticos de tiempo real:
  - Cumplir especificaciones temporales (un tiempo de respuesta suficientemente breve)
  - Por tanto, comportamiento temporal predecible
  - Lo más urgente tiene prioridad
  - Las tareas no se tratan de manera equitativa
  - La eficiencia no es el fin principal
- Conclusión: No todos los sistemas operativos son adecuados para tiempo real

Sistemas operativos y tiempo real

- ¿Qué puede ser un problema en los s.o. convencionales?
  - No es realmente de un problema de funcionalidad
  - La estrategia de planificación puede ser inadecuada - las tareas más urgentes no tienen prioridad efectiva
  - El tiempo de respuesta del propio sistema operativo puede ser impredecible
  - Falta de adecuación para sistemas empotrados (necesidad elevada de memoria, dificultad para trabajar sin los periféricos habituales…)
- Algunos s.o. convencionales: Windows, UNIX, Linux
- Algunos s.o. pensados para aplicaciones de tiempo real:
  - QNX
  - VxWorks (Tornado)
  - FreeRTOS
  - RTLinux

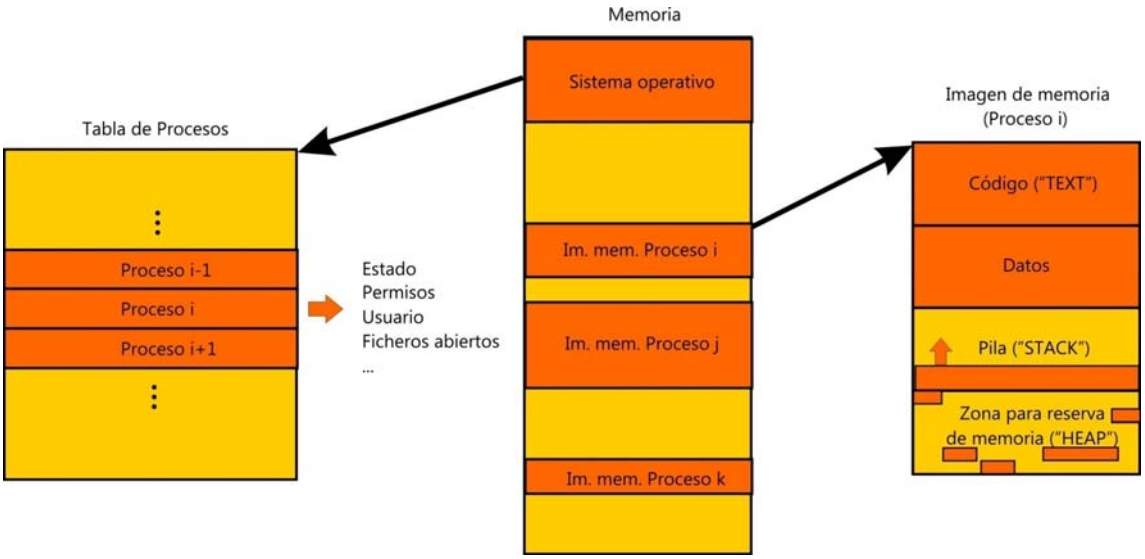
Algunos conceptos básicos sobre S.O.

- Proceso
- Interfaz de usuario
- Memoria virtual y direccionamiento paginado
- Archivos y directorios
- Llamadas al sistema

Procesos

- Proceso
  - Nombre típico para una unidad de concurrencia en la mayor parte de los sistemas operativos
  - No es un programa. Es un programa en ejecución
  - El mismo programa puede utilizarse en varios procesos
- Dos componentes:
  - Imagen de memoria
    - Creada a partir del programa ejecutable
    - Contiene instrucciones y datos
  - Entrada en la tabla de procesos
    - Estado de ejecución (p.e. copia de los registros)
    - Recursos utilizados (archivos, "sockets"…)
    - Datos administrativos (permisos, usuario…)

Procesos



- Intercambio de memoria ("memory swapping")
  - Las imágenes de memoria no siempre están en memoria principal
  - Se vuelcan a disco imágenes de procesos inactivos
  - Permite mantener más procesos de los que caben en MP

Interfaz de usuario

- Interfaz de usuario:
  - Intérprete de comandos (“shell”) en modo texto
  - Interfaz gráfica de usuario (servidor de pantalla, gestor de ventanas)
- Hay que tener en cuenta que
  - Son el aspecto externo del s.o., no deben identificarse con él
  - Normalmente son procesos que utilizan los mismos servicios que los del usuario
  - Puede haber varias opciones para escoger

Direccionamiento virtual y paginado

- Algunos problemas a resolver en un s.o.:
  - Gestión de memoria (buscar sitio para un proceso)
    - Cuando entran y salen procesos, la memoria es ocupada y liberada según un patrón aleatorio
    - La memoria libre podría quedar demasiado dispersa para ser útil
  - Protección de datos (crear “barreras” entre procesos)
  - Reubicación (¡un proceso puede cambiar de dirección!)
    - Los procesos deben utilizar el espacio disponible; no pueden crearse para direcciones específicas
    - Si vuelven de disco, la memoria disponible puede haber cambiado
- Una medio muy extendido de resolver estos problemas: Direccionamiento virtual y paginado

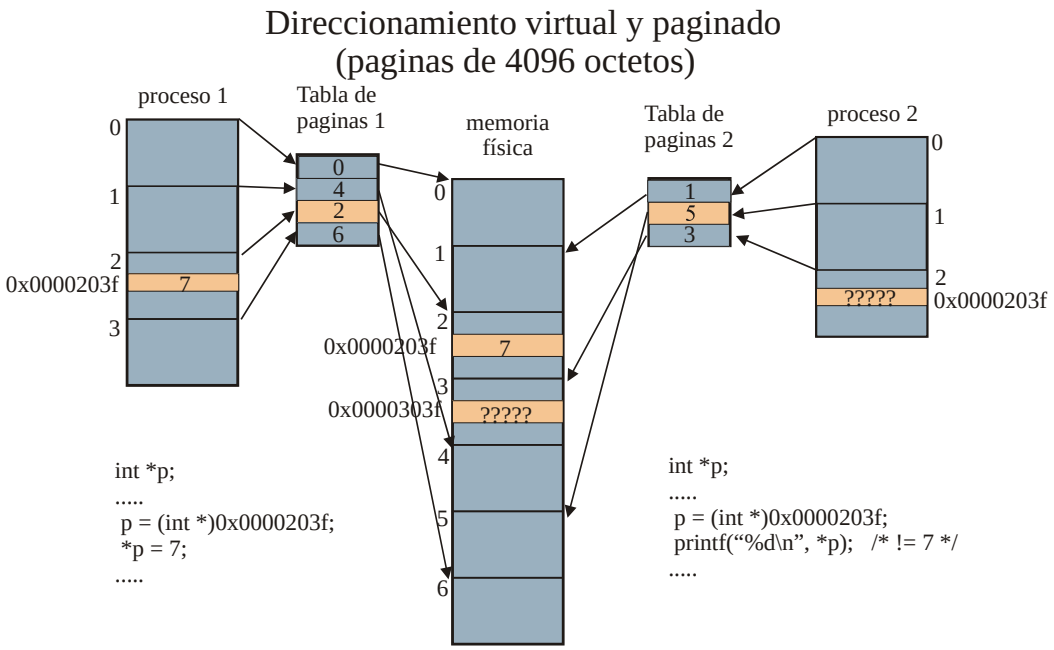
Direccionamiento virtual y paginado

- Dos modos de direccionamiento combinados:
  - Direccionamiento virtual:
    - Cada proceso posee un espacio de direcciones privado
    - Se necesita traducción a direcciones reales
  - Direccionamiento paginado:
    - Memoria dividida en páginas de tamaño igual a una potencia de 2
    - Direcciones compuestas de número de página y desplazamiento
- Direccionamiento virtual y paginado:
  - La traducción de direcciones se hace utilizando una tabla de páginas
  - Cada página virtual se “mapea” como una unidad sobre una página real

Direccionamiento virtual y paginado

- Tabla de páginas
  - Una entrada para cada página virtual
  - Como mínimo: número de pagina real asociada
  - Otros datos: Bits de protección, indicador de página en disco
  - Excepción de fallo de página
    - Si la página no está en MP, el s.o. debe traerla de la memoria de intercambio
    - Es transparente al proceso (aunque tarda más de lo normal…)
- Conclusiónes:
  - Solución de los problemas de gestión de memoria, protección y reubicación
  - Es necesaria circuitería especializada (MMU) para que sea viable

Direccionamiento virtual y paginado



- Una tabla de páginas por proceso
- ¡No es inmediato compartir memoria!

Sistema de archivos "tipo UNIX" y llamadas al sistema operativo

- Archivos y directorios
  - Un solo árbol de directorios (los discos individuales están ocultos por los puntos de montaje)
  - Toda los dispositivos de E/S están en el árbol como archivos
  - "Path name" y directorio de trabajo
  - Permisos tipo UNIX
  - Entrada y salida standard, salida de error
- Llamadas al sistema
  - Al nivel más bajo se ejecutan con instrucciones especiales
  - Habitualmente se utilizan funciones de biblioteca:
  - Interfaz adecuada al lenguaje
  - Operaciones a nivel de usuario
  - Una o más llamadas al sistema
  - Posibles bloqueos para sincronización (esperar datos)
  - Ejemplo: cuenta = read(archivo, buffer, nbytes);

Software de bajo nivel

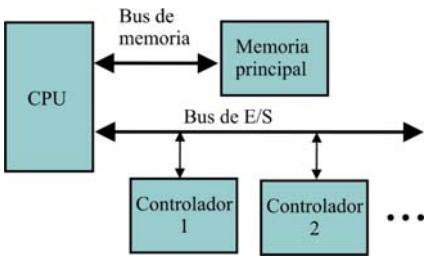
- S.T.R: Dispositivos de E/S no standard. Puede ser necesario programar a bajo nivel
- Se prefiere hacerlo desde un lenguaje de alto nivel
- Requerimientos:
  - Acceso a registros de E/S
  - Tratamiento de interrupciones
  - Ejecución de instrucciones de lenguaje máquina

Software de bajo nivel

- Arquitectura de E/S
  - Espacio de direcciones propio, con instrucciones especializadas:

    in R1, <d. puerto E/S>

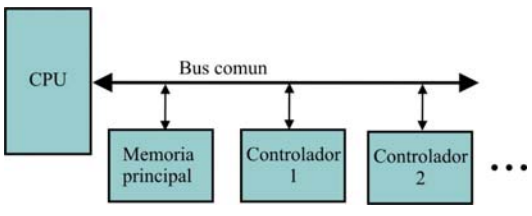
    out <d. puerto E/S>,R1



  - Espacio de direcciones compartido con memoria; instrucciones comunes:

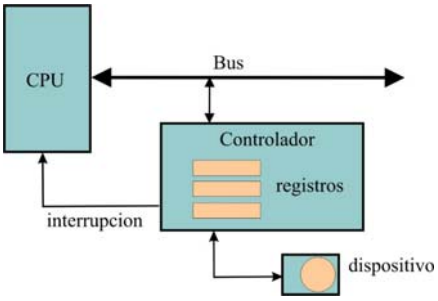
    move R1, <d. mem.>

    move R1, <d. puerto>



Software de bajo nivel: Controlador

- Dispositivo físico
  - Compatible con interfaz del bus
  - Capaz de controlar uno o varios dispositivos de E/S
- Comunicación con la CPU
  - Registros: Transmisión de información
    - Control
    - Estado
    - Datos
  - Interrupciones: Sincronización



Software de bajo nivel: Sincronización

- Necesidad de sincronización
  - Las acciones de E/S no son inmediatas
  - Normalmente el programa debe esperar al dispositivo (sincronización)
- Métodos:
  - Espera activa o consulta (“polling”):
    - La CPU muestrea repetidamente el estado del controlador hasta detectar la condición
    - Simple pero poco eficiente: Ejecuta instrucciones inútilmente
    - Debe incluir retrasos para ser admisible
  - Interrupciones:
    - El controlador envía una interrupción a la CPU
    - En respuesta se ejecuta una acción (rutina de servicio o manejador de interrupción)
    - Más eficiente pero con coste difícil de cuantificar
  - Acceso directo a memoria (DMA):
    - El controlador se encarga de almacenar los datos en memoria
    - Requiere interrupciones de sincronización, pero menos frecuentes
    - En principio más eficiente que las interrupciones

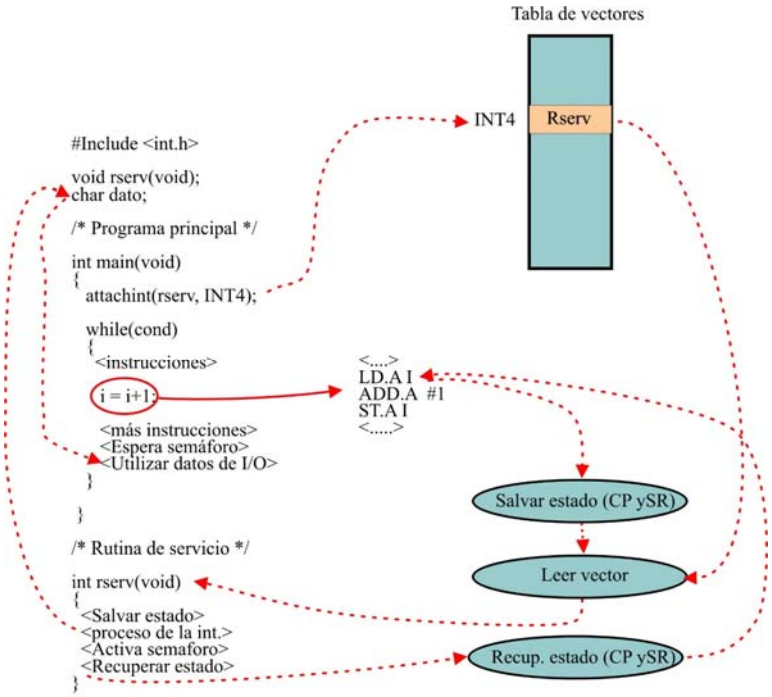
Software de bajo nivel: Tratamiento de interrupciones

- Una interrupción dispara una rutina de servicio
  - Es necesario salvar el estado del procesador y recuperarlo luego
- Identificación de la causa:
  - Vector de interrupción: La CPU consulta automáticamente una tabla de “vectores” (punteros a función)
  - Estado: Es posible conocer el número de interrupción mediante un registro de estado; la acción arranca por programa
  - Consulta: Es necesario examinar cada dispositivo para identificar la causa
- Con frecuencia, el procedimiento es mixto:
  - Varias causas por señal de interrupción: Un vector por señal, y consulta para identificar la causa
  - Varias causas por dispositivo: Vector que identifica el dispositivo y consulta para identificar la causa

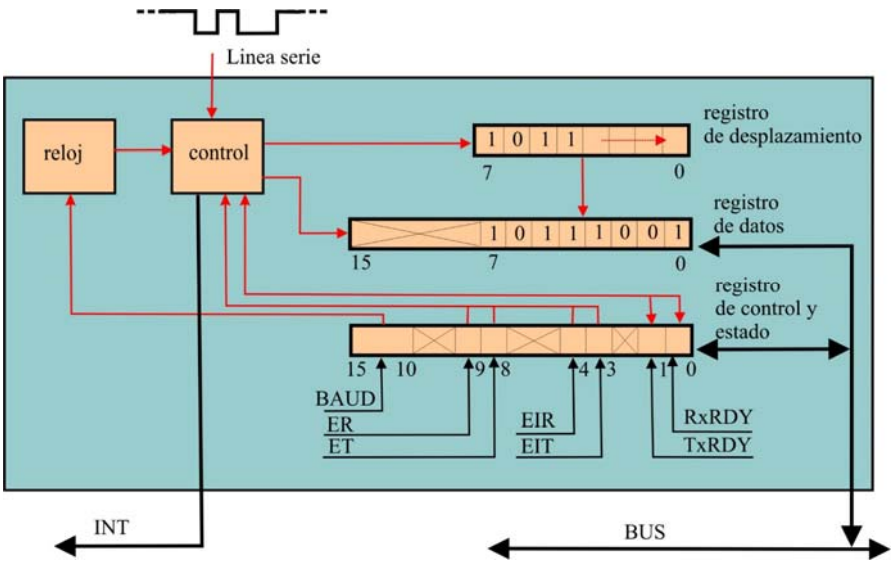
Software de bajo nivel: Tratamiento de interrupciones

- Prioridad:
  - Pueden existir niveles de prioridad, para diferentes grados de “urgencia”
  - Ante varias causas pendientes, se atiende a la de mayor prioridad
  - Prioridad estática o dinámica
  - “Anidamiento” de interrupciones
- Máscaras de interrupción
  - Una señal de interrupción puede ignorarse temporalmente (señal enmascarada)
  - Varios niveles:
    - Habilitación global de interrupciones (CPU)
    - Máscaras de señales individuales (CPU/Contr, de Int.)
    - Máscaras de causas individuales (Bits de control en los controladores de dispositivos)
  - La prioridad puede entenderse como enmascaramiento selectivo

Software de bajo nivel: Interrupción vectorizada



Ejemplo de controlador: Línea serie (recepción)



- Campos del RCE:
  - RxRDY, TxRDY: Carácter recibido o transmitido.
  - EIR, EIT: Habilitar interrupción en recepción y transmisión
  - ER, ET: Habilitar recepción y transmisión
  - BAUD: Selección de velocidad de transmisión/recepción

Software de bajo nivel y lenguaje

- Requerimientos para un lenguaje de alto nivel:
  - Representación de registros
  - Acceso a puertos de E/S o direcciones físicas
  - Programación de manejadores de interrupción
  - Excepcionalmente, insertar código máquina
- En ADA 95:
  - Cláusulas de representación de registros, incluyendo detalles de bajo nivel
  - Asociación de manejadores a interrupciones, incluyendo sincronización con el resto del código (package Ada.Interrupts)
  - Inclusión de instrucciones de lenguaje máquina

Software de bajo nivel y lenguaje

- Software de bajo nivel en C
  - Acceso a registros fácil si el espacio de direcciones es compartido con memoria:
    - Punteros para acceder a registros.
    - Operaciones lógicas bit a bit para manipular campos, o también estructuras con “campos de bits”
    - Problema: portabilidad
  - Manejadores de interrupción:
    - Funciones C sin parámetros ni resultado
    - Modificación directa de la tabla de vectores
    - Es necesario resolver sincronización
    - Funcionalidad dependiente del entorno, no portable
  - Inclusión de código máquina:
    - Sentencias “asm” (catalogadas como “extensiones” de C).
    - Enlazar funciones en ensamblador, respetando las especificaciones de llamada y retorno del sistema concreto

Software de bajo nivel y lenguaje

- Software de bajo nivel en C

```
#define HAB_RX 0x200
#define HAB_TX 0x100
#define INT_RX  0x10
#define INT_TX   0x8
#define RRDY     0x1
#define TRDY     0x2
#define BAUD     0xA
#define DIR_RCE 0xffff1020
(….)
unsigned short int *reg_ce, prog;
/* Programar solo lectura con interrupción */

reg_ce = (unsigned short int *)DIR_RCE;
prog = (BAUD << 10) |  HAB_RX | INT_RX;
*reg_ce = prog;        /* Registro escrito */
(…)
```


Software de bajo nivel y lenguaje

- Software de bajo nivel en C

```
#define RRDY     0x1
#define DIR_RCE 0xffff1020
#define DIR_RD   0xffff1022
(….)
unsigned short int *reg_ce, short int *reg_dat, valor, dato;
reg_ce = (unsigned short int *)DIR_RCE;
reg_dat = (unsigned short int *)DIR_RD;

/* Bucle de espera activa */
do {   valor = *reg_ce;
       if( (valor & RRDY) == 0)  espera_1ms();
    }  while( (valor & RRDY) == 0);
(…)
/* Leer dato */
dato = (*reg_dat)&0xff;
(...)
```

Software de E/S

- Funciones del software de E/S:
  - Interfaz independiente del dispositivo
  - Nombres uniformes en el sistema de archivos
  - Tratamiento de errores
  - Sincronización del código con las actividades de E/S
  - Gestión de “buffers” intermedios
  - Control de acceso

Software de E/S

- Niveles del software de E/S:
  - Software de E/S en nivel de usuario
  - Software de E/S en nivel de kernel, independiente del dispositivo
  - Controladores de dispositivos (“drivers”)
  - Manejadores de interrupción

Software de E/S

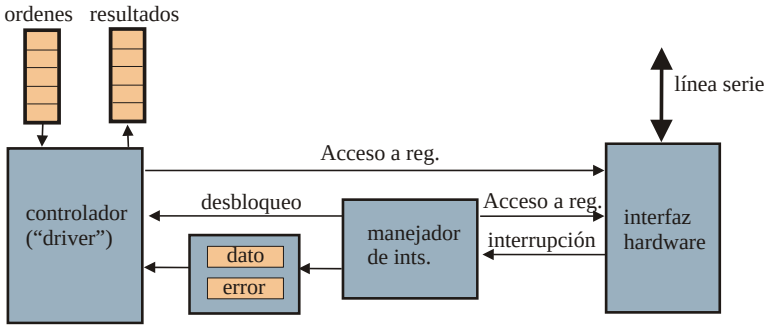
- Manejador de interrupciones
  - Pasos necesarios:
    - Guardar contexto del proceso actual y preparar contexto del manejador (pila, tabla de páginas...)
    - Operaciones comunes ligadas al hardware (reconocer interrupción, habilitar interrupciones...)
    - Procedimiento de servicio de interrupciones
    - Escoger nuevo proceso y preparar nuevo contexto
    - Pasar control a nuevo proceso
  - Normalmente: operaciones sencillas y breves
  - Puede desbloquear al controlador

Software de E/S

- Controlador (“driver”)
  - Específico para el tipo de dispositivo
  - Ofrece al resto del s.o. una determinada interfaz
  - Funciones:
    - Desarrollar peticiones en forma de una secuencia de comandos
    - Se bloquea a la espera de interrupciones
    - Realiza tratamiento de errores
  - Dificultades:
    - Peculiaridades en la interacción con el resto del s.o.
    - Puede necesitarse funcionamiento reentrante
    - Los recursos y condiciones de funcionamiento pueden cambiar

Software de E/S

Controlador y manejador de interrupciones



Leer n datos en buffer

Habilitar lectura e int.
Desde i =1 hasta n
  Esperar interrupción
  Si error, devolver cod. error
  Copiar dato en buffer
Fin desde
Inhibir int. asociada
Devolver resultado correcto

Atender int.

Salvar contexto
Leer registro de error
Si ha habido error,
  error = 1
si no,
  dato = reg. de recepción
fin si
Desbloquear controlador
Gestionar paso a nuevo proceso

Software de E/S

- Software de E/S independiente del dispositivo (nivel de kernel)
  - Realiza operaciones comunes y ofrece una interfaz única
  - Funciones:
    - Correspondencia entre nombres y dispositivos
    - Gestión de buffers para transferencia de datos
    - Tratamiento de errores
    - Gestión de acceso a dispositivos
    - Ocultar diferencias entre dispositivos

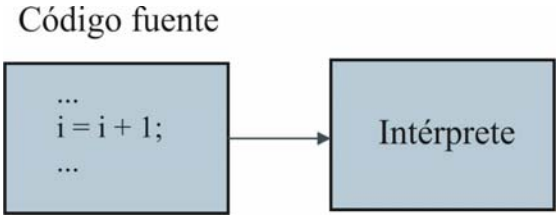
Software de E/S

- Software de E/S en nivel de usuario
  - Bibliotecas de E/S y procesos del sistema
  - Funciones:
    - Ofrecer la interfaz requerida por el lenguaje de programación
    - Sincronización con operación de E/S (E/S bloqueante)
    - Formateo de datos
    - Spooling (un proceso dedicado gestiona el acceso a un dispositivo)

Máquina simbólica

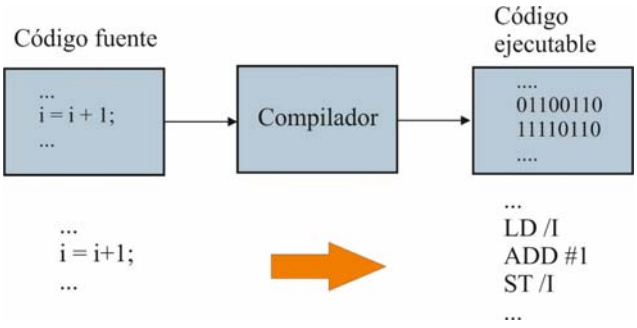
- Máquina simbólica:
  - Máquina virtual sobre la Máquina Operativa que "entiende" lenguajes de alto nivel
  - La máquina real no puede hacerlo
  - El programa de alto nivel es el código fuente
- Opciones de implementación:
  - Interpretación: Un intérprete lee el código fuente como datos y lo ejecuta
  - Compilación: Un compilador traduce el código fuente a código máquina

Interpretación



- El intérprete procesa el código fuente como datos
- El interprete siempre se está ejecutando cuando se ejecuta el programa
- Muy portable pero habitualmente muy ineficiente si se compara con la compilación
- Ejemplos: archivos “m” de MATLAB, JAVA
  - JAVA se compila primero a “bytecode” (lenguaje intermedio)
  - La máquina virtual de JAVA, creada para un sistema específico, interpreta el “bytecode”

Interpretación

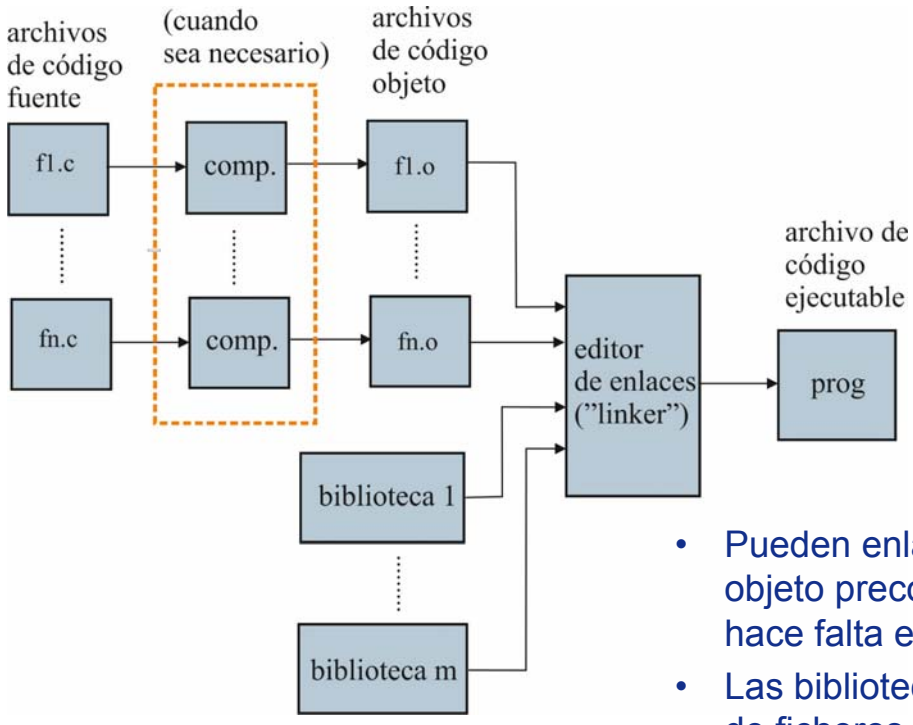


- El compilador traduce el código fuente a un código ejecutable equivalente
- Una vez compilado, el código ejecutable puede ejecutarse tantas veces como sea necesario
- El compilador no ejecuta el código fuente; su papel acaba con la compilación
- Ejemplos: C, C++, ADA

Compilación separada

- Necesaria para modularidad
  - Permite construir el programa a partir de módulos
  - Cada módulo se compila separadamente
- Dos fases:
  - Compilación a objeto (sólo para los módulos que han cambiado):
    - Traduce código fuente para general código objeto reubicable
    - No pueden asignarse direcciones definitivas
    - Las referencias externas (a elementos de otros módulos) están indefinidas
    - El código fuente puede adaptarse más tarde a direcciones arbitrarias
  - Edición de enlaces o "linking" (para todos los módulos)
    - Asignación definitiva de direcciones a cada módulo
    - Reubicación del código de todos los módulos
    - Creación de una tabla de símbolos globales
    - Resolución de referencias externas

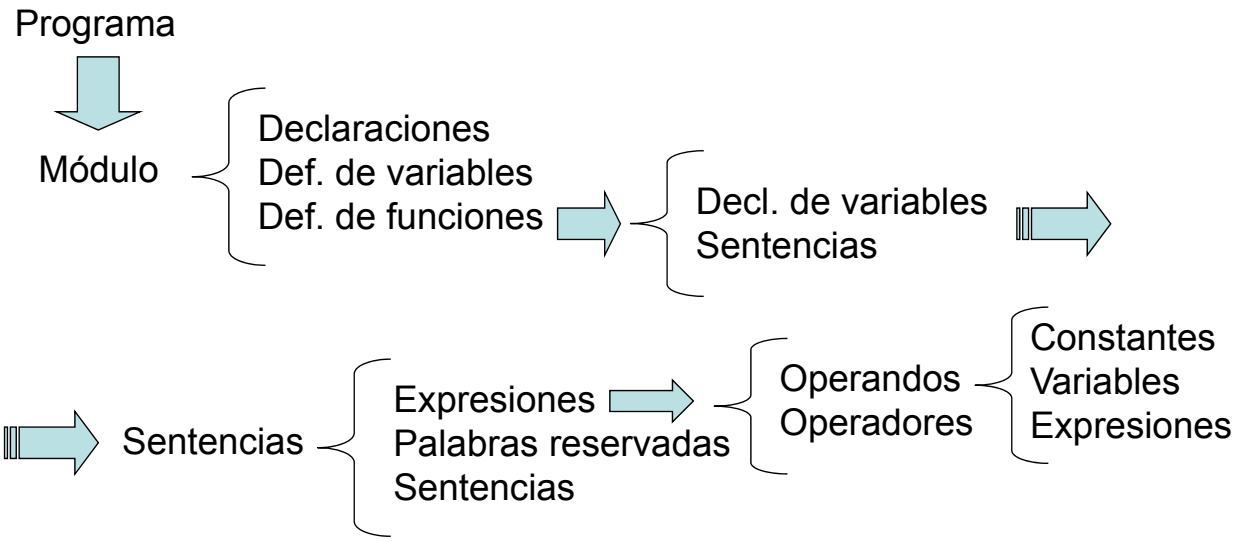
Compilación separada



- Pueden enlazarse ficheros objeto precompilados; no hace falta el código fuente
- Las bibliotecas son conjuntos de ficheros objeto

Conceptos de lenguaje C

- Estructura del lenguaje



Conceptos de lenguaje C

- Variables, constantes, operadores, expresiones
  - Tipos básicos
  - Visibilidad y tipo de almacenamiento
  - Funciones y argumentos
  - Vectores y matrices
  - Inicialización
  - Constantes
  - Sentencias y expresiones
  - Operadores
  - Precedencia y asociatividad
  - Conversión de tipo implícita
  - Orden de evaluación

Visibilidad

```
/* modulo1.c */

#include <stdio.h>
void f1(void);
int fext( int y);
int privada(int x);

/* Programa principal */

extern  int externa1;
extern  int externa3;

int main(void)
{
  int i;
  i = externa2;
  printf("externa1: %d\n", externa1);
  printf("externa3: %d\n", externa3);
  f1();
  f1();
  printf("fext: %d\n", fext(4));
  privada(i);
}


void f1(void)
{
  int aux;
  int externa3;
  static int estatica;
  externa3 = 7;
  aux = i;
  printf("externa1: %d\n", externa1);
  printf("externa3: %d\n", externa3);
  printf("estatica: %d\n", estatica);
  estatica = estatica +1;
}
```

```
/* modulo2.c */

int externa1 = 27;
static  int externa3 = 18;
int  externa2;

int fext(int y)
{
  externa2 = y;
  return externa3;
}

static void  privada(int x)
{
  externa3 = x;
}
```

Después de corregir errores:

```
> gcc -c modulo1.c
> gcc -c modulo2.c
> gcc -o prog.exe modulo1.o modulo2.o
> prog
27
27
7
0
27
7
1
18
```

Símbolos globales de modulo1:

```
main
f1
```

Símbolos globales de modulo2:

```
externa1
externa2
fext
```


Operadores de C

TABLE 2-1. PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

OPERATORS	ASSOCIATIVITY
`( )  [ ]  ->  .`	left to right
`! ~  ++  --  +  -  *  &  (type)  sizeof`	right to left
`*  /  %`	left to right
`+  -`	left to right
`<<  >>`	left to right
`<  <=  >  >=`	left to right
`==  !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`=  +=  -=  *=  /=  %=  &=  ^=  |=  <<=  >>=`	right to left
`,`	left to right

Unary +, -, and * have higher precedence than the binary forms.

Conceptos de lenguaje C

- Sentencias de control
  - Programación estructurada
  - Bloques en secuencia
    - Sentencia compuesta
  - Operaciones condicionales
    - Sentencia **if-else**
    - Sentencia **switch**
  - Operaciones iterativas
    - Sentencia **for**
    - Sentencia **while**
    - Sentencia **do-while**
  - Saltos
    - Sentencia **break**
    - Sentencia **continue**
    - Sentencia **return**
    - Función **exit()**
    - Etiquetas y sentencia **goto**

Conceptos de lenguaje C

- Sentencia compuesta

```
{
    <sentencia 1>
    <sentencia 2>
    ……..
}
```

- Sentencia **if-else**

```
if(<expresión>)
  <sentencia 1>
else
  <sentencia 2>
```

El "**else**" acompaña al "**if**" más cercano

- Sentencia **switch** (forma habitual)

```
switch(<expr. con resultado entero>)
{
  case <constante 1>
  <sentencia 1>
  break;
  case <constante 2>
  <sentencia 2>
  break;
  …….
  default:
  < sentencia n>
  break;
}
```

Conceptos de lenguaje C

- Sentencia **for**

  for(<expresión inicial>; <expresión de mant.>; <expresión final (actualización)>)
  <sentencia>

- Sentencia **while**

  while(<expresión (cond. de mantenimiento>)
  <sentencia>

  La sentencia **puede no llegar a ejecutarse** nunca

- Sentencia **do-while**

  do
  <sentencia>
  while(<expresión (cond. de mantenimiento>)

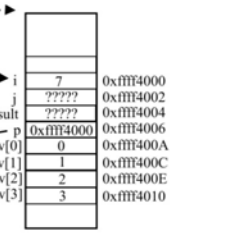
  La sentencia se ejecuta **al menos una vez**

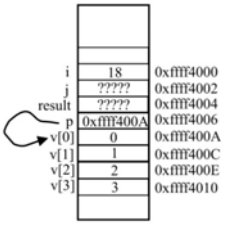
Conceptos de lenguaje C

- Punteros y tablas (“arrays”)
  - Punteros y direcciones
  - Punteros y tablas
  - Aritmética de punteros
  - Inicialización de punteros
  - Cadenas de caracteres
  - Punteros a puntero
  - Punteros a función
  - Argumentos de la línea de comando

Punteros

```
#include <stdio.h>

void suma(int x, int y, int *res);

/* Programa principal */

int main(void)
{
 /* Declaraciones */

  int i, j, result;
  int *p;
  int v[4] = {0, 1, 2, 3};

 /* Sentencias */

  i = 7;
  p = &i;

  printf("i: %d, p: %p\n", i, p);
  *p = 18;
  printf("i: %d, p: %p\n", i, p);

  p = v;

  printf("v[1]: %d, p[1]: %d,  *(p+1): %d, p: %p, p+1: %p\n",
          v[1], p[1], *(p+1), p, p+1);

 i = 4; j = 7;
  suma(i, j, &result);

  printf("i: %d, j:  %d, result %d\n", i,j,result);
}

/* Rutina de suma */

void suma(int x, int y, int *p)
{
  *p = x + y;
}
```






```
Al ejecutar el programa:
> prog
i: 7, p: ffff4000
i: 18, p: ffff4000
v[1]: 1, p[1]: 1, *(p+1): 1, p: ffff400A, p + 1: ffff400C
I: 4, j: 7, result: 11
```


Cadenas de caracteres

- Caracteres almacenados en posiciones sucesivas
- Cadena definida por una dirección y un terminador (‘\0’)
- Funciones:

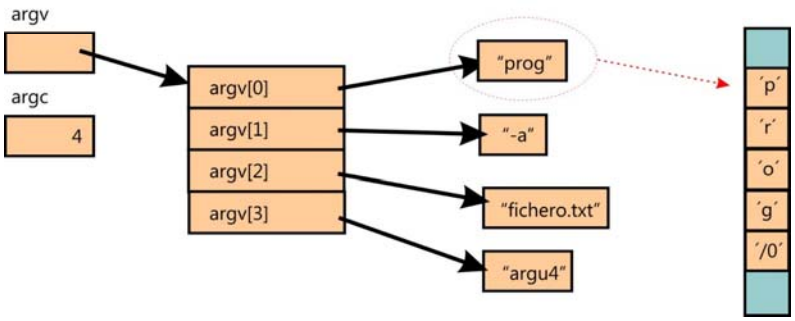
```
#include <string.h>
size_t strlen(char *cadena);
char *strcpy(char *destino, char *origen);
char *strcat(char *destino, char *origen);
int strcmp(char *cad1, char *cad2);
```

Cadenas de caracteres

- Función **strlen**:
  - Devuelve el número de caracteres de la cadena, sin contar el terminador
- Función **strcpy**:
  - Copia la cadena origen en la cadena destino
  - Devuelve un puntero a la cadena destino
- Función **strcat**:
  - Añade la cadena origen al final de la cadena destino
  - Devuelve un puntero a la cadena destino
- Función **strcmp**:
  - Compara las cadenas 1 y 2
  - Devuelve 0 si son iguales, >0 si la cadena 1 debe ir detrás de la 2 y <0 si es al contrario

Argumentos de la línea de comando

```
int main(int argc, char **argv)
{
   ....
}

prog -a fichero.txt argu4
```



Conceptos de lenguaje C

- Estructuras
  - Definición y declaración
  - Inicialización
  - Acceso a elementos
  - Tablas de estructuras
  - Estructuras anidadas
  - Estructuras como argumentos de funciones
  - Estructuras con autoreferencia

Conceptos de lenguaje C

- Otros conceptos de C
  - Uniones
  - Definición de tipos
  - Tipos enumerados
  - Comandos del preprocesador
  - Campos de bits

Conceptos de lenguaje C

- Entrada-salida y ficheros
  - Conceptos básicos
  - Apertura y cierre de ficheros (**fopen**, **fclose**)
  - Lectura y escritura de caracteres aislados (**fgetc**, **fputc**, **getchar**, **putchar**)
  - Lectura y escritura de cadenas de caracteres (**fgets**, **fputs**, **gets**, **puts**)
  - Lectura y escritura formateada (**fprintf**, **fscanf**, **printf**, **scanf**)
  - Lectura y escritura de bloques (**fread**, **fwrite**)
  - Acceso aleatorio **(fseek**, **ftell**)
  - Detección de error y fin de archivo (**ferror**, **feof**)

Normas POSIX para tiempo real

- Introducción
  - Definidas por el IEEE (“Institute of Electrical and Electronics Engineers”)
  - Estandarizadas por ANSI e ISO
  - Otras normas IEEE: 802.3, (Ethernet) 802.11 (Wifi)…
- POSIX:
  - “Portable Operating System Interface”
  - Familia de normas que definen aspectos básicos de sistemas operativos: IEEE 1003.x
  - IEEE 1003.1 define las llamadas al sistema

Normas POSIX para tiempo real

- Resumen de las normas POSIX
  - POSIX 1003.1a (antes .1):
    - UNIX normalizado. Funciones básicas: Concurrencia, entrada/salida, temporización…
    - Se solapa con ANSI-C por definición
  - POSIX 1003.1b (antes .4):
    - Funciones “de tiempo real”: Servicios más avanzados de planificación, temporización, comunicación…
    - Numerosos bloques opcionales
  - POSIX 1003.1c (antes .4a): Hilos y llamadas asociadas
- Hay varias ediciones, y un cierto número de componentes opcionales

Normas POSIX para tiempo real

- Comprobación de opciones
  - En tiempo de compilación: Basada en cabeceras

```
#define _POSIX_C_SOURCE 199309L
#include <unistd.h>
#include <limits.h>
```

  - En tiempo de ejecución: Basada en funciones

```
#include <unistd.h> /* long sysconf(int name); */
result = sysconf(_SC_SEMAPHORES);
```

Normas POSIX para tiempo real

- Idea fundamental: Hacer posible la portabilidad entre diferentes sistemas operativos
- Soportadas por un número creciente de sistemas operativos: Linux, QNX, VxWorks…
- Habitualmente implementadas como una biblioteca que provee servicios POSIX a partir de los nativos
- Las normas POSIX especifican sobre todo requerimientos funcionales, no de tiempos de respuesta
- Soportar las normas POSIX no implica que el sistema operativo sea apropiado para tiempo real