# Organización y Arquitectura de Computadoras
# Practica 8: Llamadas al sistema

Profesor: José Galaviz Casas
Ayudante de Laboratorio: Ricardo Enrique Pérez Villanueva

27 de abril de 2023

## 1. Objetivos

### 1.1. Generales

Introducir al alumno a los conceptos básicos de operaciones de entrada/salida, cubriendo la interacción entre el usuario, las rutinas, el sistema operativo y el hardware.

### 1.2. Particulares

El alumno continuara practicando la programación en el lenguaje ensamblador de MIPS y al finalizar la practica:

- Estará familiarizado con las llamadas al sistema.
- Sera capaz de manipular cadenas de caracteres con lenguaje ensamblador MIPS-32.

## 2. Requerimientos

### 2.1. Conocimientos Previos:

- Instrucciones aritméticas, lógicas y de manejo de flujo de programas del lenguaje ensamblador MIPS.
- Convención de llamadas a subrutinas.
- Manejo de archivos en sistemas POSIX.
- Codificacion ASCII.

### 2.2. Tiempo de realización sugerido: 5 horas

### 2.3. Numero de colaboradoras: Parejas (Equipos de 2 personas)

### 2.4. Software a utilizar.

- Java Runtime Environment versión 5 o superior.
- El paquete MARS.

## 3. Planteamiento

En una computadora moderna, el sistema operativo es el encargado de gestionar la comunicación con los dispositivos de entrada y salida; administrar la memoria, otorgando nuevos segmentos a los programas que lo requieran; y dirigir la ejecución de los programas, entre otras muchas cosas. Los programas solicitan estos servicios al sistema operativo a través de un mecanismo denominado llamada al sistema, una instrucción especial que transfiere el control del procesador al sistema operativo, este realiza la tarea que le fue solicitada y

al finalizar, regresa los resultados y el control al programa que lo solicito. En nuestro caso, al no contar con un sistema operativo en MARS, este simula un pequeño subconjunto de llamadas. En esta practica se desarrollara un interprete de comandos en donde se aprovecharan algunas de las llamadas al sistema simuladas por MARS. En la figura 1 se muestra un fragmento de código como ejemplo, el programa imprime un mensaje en la terminal de MARS.

```
        .data
cad:    .asciiz "Hola mundo"
        .text
        la      $a0, cad        # Cargar argumento
        li      $v0, 4          # Cargar código imprimir cadena
        syscall
        li      $v0, 10         # Cargar código terminar prog
        syscall
```

Figura 1: Ejemplo de llamada al sistema en MARS.

# 4. Desarrollo:

## 4.1. Llamadas al sistema en MIPS

Para solicitar un servicio en el simulador MARS, un programa debe cargar el código de la llamada en el registro $v0, los argumentos en los registros $a0-$a2 o $f12 en el caso de punto flotante y ejecutar la instrucción syscall. Si la llamada regresa algún valor, este se encontrara en el registro $v0 o $f0 para datos de punto flotante y dobles. En la tabla 1 se encuentran las llamadas necesarias para completar parte de la practica, el resto las puedes encontrar en el menú ayuda del simulador o en el siguiente enlace: `https://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html`.

| Código | Servicio | Argumeto | Retorno |
|---|---|---|---|
| 1 | Imprimir entero | `$a0` = Entero | |
| 2 | Imprimir flotante | `$f12` = Flotante | |
| 3 | Imprimir doble | `$f12` = Doble | |
| 4 | Imprimir cadena | `$a0` = Cadena* ** | |
| 5 | Leer entero | | `$v0` = Entero |
| 6 | Leer flotante | | `$f0` = Flotante |
| 7 | Leer doble | | `$f0` = Doble |
| 8 | Leer cadena | `$a0` = Buffer*<br>`$a1` = Tamaño | |
| 9 | Asignar memoria | `$a0` = Cantidad | `$v0` = Dirección |
| 10 | Terminar ejecución | | |
| 11 | Imprimir caracter | `$a0` = Caracter | |
| 12 | Leer caracter | | `$v0` = Caracter |
| 13 | Abrir archivo | `$a0` = Nombre* **<br>`$a1` = Banderas<br>`$a2` = Modo | `$v0` = Descriptor |
| 14 | Leer del archivo | `$a0` = Descriptor<br>`$a1` = Buffer*<br>`$a2` = Tamaño | `$v0` = Número de caracteres leidos |
| 15 | Escribir al archivo | `$a0` = Descriptor<br>`$a1` = Buffer*<br>`$a2` = Tamaño | `$v0` = Número de caracteres escritos |
| 16 | Cerrar archivo | `$a0` = Descriptor | |

* El argumento es un apuntador, es decir la dirección de memoria en donde comienza el dato.
** Cadena con carácter nulo al final.

## 4.2. Interprete de comandos

Un interprete de comandos es un programa que ejecuta interactivamente ordenes introducidas por un usuario. El interprete recibe una cadena de texto con la orden del usuario, la analiza, determina la tarea a realizar y la ejecuta. El interprete que simularemos, cumplirá con las siguientes carácterısticas:

1. La entrada y la salida sera por medio del simulador de terminal de MARS.

2. El interprete imprimirá en la terminal un prompt (un carácter o secuencia de caracteres) para indicar al usuario que espera un comando, el usuario podrá escribir un comando en la terminal, el interprete ejecutara dicho comando, mostrara el resultado en la terminal y repetirá el proceso.

3. Si el usuario solicita un comando no implementado, el interprete debe mostrar un mensaje indicando el error.

4. Los comandos que se deben implementar:

   - **help**: Imprime información de los comandos disponibles y sus opciones. Si se llama sin argumentos, imprime una lista de los comandos disponibles.

     Sinopsis: `help [arg]` Argumentos: `arg` - Imprime la descripción y opciones del comando arg.
   - **joke**: Usando la llamada a sistema 41 o 42, muestra un chiste al azar. Joke debe ser capaz de generar al menos 3 chistes distintos.[1] Sinopsis: `joke` Argumentos: Ninguno
   - **song**: Usando la syscall 31 o 33, se debe de generar una canción corta de al menos 10 notas (10 llamadas a syscall). [2] Sinopsis: `song` Argumentos: Ninguno
   - **rev**: Imprime la reversa de una cadena. Si no se especifica un archivo, se utiliza la entrada estándar (solo una linea). Sinopsis: `rev [file]`

     Argumentos: file - Archivo de texto a revertir.
   - **cat**: Concatena 2 archivos y los muestra en pantalla. Sinopsis: `cat file [file]`

     Argumentos: file - Archivo a concatenar.
   - **exit**

     Termina la ejecución del interprete de comandos, terminando asi la simulación de MARS.

5. Si el usuario comete un error al dar los argumentos de un comando, la subrutina del comando sera la encargada de informar al usuario el error.

6. **Su programa debe comparar toda las cadenas introducidas**, no solo el primer char. Para esto pueden usar las operaciones sub, stl o el siguiente código:

```
1  # Your code here!
2  # Loop que compara cada char de una cadena
3  # Funciona comparando cada byte en un registro, de forma similar
4  #a comparar cada elemento guardado en un arreglo
5  cmploop:
6      lb      $t2,($s2)               # Siguiente char de la primera cadena
7      lb      $t3,($s3)               # Siguiente char de la segunda cadena
8      bne     $t2,$t3,cmpne           # Si son diferentes, saltamos a cmpne
9
10     beq     $t2,$zero,cmpeq         # Si llegamos al final del string, salta
11
12     addi    $s2,$s2,1               # Apuntamos al siguiente char
13     addi    $s3,$s3,1               # Apuntamos al siguiente char de la otra
14     j       cmploop
15
16 # Las cadenas no son iguales. Mandamos mensaje
17 cmpne:
18     la      $a0,errormsg
19     li      $v0,4
20     syscall
21     j       main
22
23 # Las cadenas son iguales, mandamos mensaje
24 cmpeq:
25     la      $a0,print_command
26     li      $v0,4
27     syscall
28     j       main
```

[1] No, entregar esta practica mal no cuenta como chiste.
[2] No, la mayonesa no es un instrumento.

# 5. Entrada

El interprete sera interactivo, el usuario introducirá la entrada o comando en el simulador de terminal de MARS. El tamaño máximo de los archivos de enterada sera de 1 KB. **NO manden su archivo txt junto a la practica**

# 6. Salida

El interprete mostrara el resultado en el simulador de terminal de MARS.

# 7. Variables Libres

El alumno deberá definir 2 comando para el interprete. Ademas, la cancion y los chistes quedan a discreción del alumno.

# 8. Procedimiento

Escribe las rutinas necesarias en lenguaje ensamblador de MIPS respetando los siguientes lineamientos:

- El programa debe terminar con la llamada al sistema exit.
- Deberás entregar un único archivo de código fuente.
- No olvides documentar el código.

# 9. Ejercicios

1. Desarrolla los 6 comandos descritos en la sección 4.2.
2. Agrega 2 comandos extra, debe ser de una complejidad similar a los otros. No olvides documentarlo para el comando help.

## 9.1. Puntos extra:

- (1 punto extra) La canción para el comando **song** debe durar al menos 15 segundos, y debe sonar relativamente bien a pesar de hacerla en Ensamblador.
- (2 puntos extra) Usar subrutinas para cada el proceso de cada comando descrito arriba y para comparar las cadenas. Si se usan subrutinas, usa la convención de llamadas a subrutinas vista en la practica pasada. Agrega a la rutina main código para que se carguen de la memoria los argumentos en los registros adecuados y se llame a la subrutina que resuelva el comando.
- (-99999 puntos) Si su chistes son malos los repruebo.

# 10. Preguntas

1. ¿Que es `shell`? Da un pequeño resumen sobre que es Shell, para que se usa y como funciona.
2. ¿Cual es el propósito de un interprete de comandos para cualquier sistema?
3. Si tuviéramos que implementar el interprete de comandos en una sistema operativo. ¿Porque es importante usar subrutinas?
4. Considerando los componentes básicos de una computadora moderna, además de las llamadas al sistema descritas en la practica, ¿que otras llamadas al sistema resultarıan necesarias?

5. De forma general, ¿Que comandos son básicos para un interprete de comandos de un sistema operativo?

6. Investiga y describe con tus propias palabras ¿como resuelve una llamada al sistema el sistema operativo?