



PRÁCTICA 1a parte: Mastermind

Estructura de Computadores
Grado en Ingeniería Informática
feb19-jun19

Estudios de Informática, Multimedia y Telecomunicaciones

Presentación

La práctica que se describe a continuación consiste en la programación en lenguaje ensamblador x86_64 de un conjunto de subrutinas, que se tienen que poder llamar desde un programa en C. El objetivo es implementar el juego del MasterMind.

La **PRÁCTICA** es una **actividad evaluable individual**, por lo tanto no se pueden hacer comentarios muy amplios en el foro de la asignatura. Se puede hacer una consulta sobre un error que tengáis a la hora de ensamblar el programa o de algún detalle concreto, pero no se puede poner el código de una subrutina o bucles enteros.

Competencias

Las competencias específicas que persigue la PRÁCTICA son:

- [13] Capacidad para identificar los elementos de la estructura y los principios de funcionamiento de un ordenador.
- [14] Capacidad para analizar la arquitectura y organización de los sistemas y aplicaciones informáticos en red.
- [15] Conocer las tecnologías de comunicaciones actuales y emergentes y saberlas aplicar convenientemente para diseñar y desarrollar soluciones basadas en sistemas y tecnologías de la información.

Objetivos

Introducir al estudiante en la programación de bajo nivel de un computador, utilizando el lenguaje ensamblador de la arquitectura Intel x86-64 y el lenguaje C.

Recursos

Podéis consultar los recursos del aula pero no podéis hacer uso intensivo del foro.

El material básico que podéis consultar es:

- Módulo 6: Programación en ensamblador (x86_64)
- Documento “Entorno de trabajo”

La Práctica: MasterMind

La práctica consiste en implementar un juego del MasterMind. El juego consiste en introducir una combinación secreta de 6 números entre 0 y 9 y entonces ir introduciendo combinaciones de 6 números (jugadas) hasta descubrir la combinación secreta o agotar el número máximo de jugadas. Cada vez que introducimos una jugada se tiene que comparar la jugada con la combinación secreta.

En la 1a parte de la práctica sólo se compara toda la jugada con la combinación secreta indicando si se ha acertado o no la combinación.

En la 2a parte de la práctica hay que indicar para cada jugada cuantos números se han acertado de la combinación secreta, y si son aciertos en su sitio o fuera de su sitio.

Nos tenemos que poder mover de izquierda a derecha mientras introducimos tanto la combinación secreta como las jugadas.

La práctica consta de un programa en C, que os damos hecho y que NO PODÉIS MODIFICAR, y un programa en ensamblador que contiene algunas subrutinas ya hechas y otras que tenéis que implementar vosotros. Más adelante encontraréis la información detallada sobre la implementación de la práctica.

El archivo C contiene una versión completa de la práctica para que os sirva de guía a la hora de implementar las subrutinas en ensamblador. También os permite ejecutar el juego para ver cómo tiene que funcionar.

La práctica se ha dividido en dos partes:

PRIMERA PARTE OBLIGATORIA:

Para esta primera parte os proporcionamos dos archivos: MMp1c-es.c y MMp1-es.asm. El código C (MMp1c-es.c) no lo podéis modificar: sólo tenéis que implementar en ensamblador en el archivo MMp1-es.asm las funcionalidades siguientes:

- PrintTries: Mostrar en pantalla los intentos que quedan (variable *tries*).
- GetSecretPlay: Leer una jugada i guardarla en un vector de 6 posiciones (*vplay*). Mientras se introducen los valores se tiene que poder mover el cursor: 'j' (izquierda) , 'k' (derecha) controlando que al mover el cursor no salga fuera del espacio asignado y mostrar los valores de la jugada en pantalla. Al pulsar 'ENTER' se aceptará la combinación secreta, y se pasará a leer la primera jugada, en cambio si se pulsa 'ESC' se saldrá del programa.
- GetSecretPlay: Leer la combinación secreta y guardarla en un vector de 6 posiciones (*vSecret*). Mientras se introducen los valores se tiene que poder mover el cursor: 'j' (izquierda) , 'k' (derecha) controlando

que al mover el cursor no salga fuera del espacio asignado, pero ahora en pantalla no se mostrará el valor. En su lugar se mostrará un asterisco ' * '. Al pulsar 'ENTER' se aceptará la combinación secreta, y se pasará a leer la primera jugada, en cambio si se pulsa 'ESC' se saldrá del programa.

ATENCIÓN: La subrutina para leer la jugada y leer la combinación secreta es la misma. Según sea un caso u otro se tienen que hacer cosas diferentes. La variable (*state*) nos permite diferenciar cada caso.

- PrintSecret: Mostrar la combinación secreta (*vSecret*) en la parte superior del tablero cuando finaliza el juego.
- CheckSecret: Verificar que la combinación secreta (*vSecret*) no tenga espacios (es el valor con el que se inicializa el vector antes de leer).
- CheckPlay: Comparar el vector de la jugada (*vPlay*) con el vector de la combinación secreta (*vSecret*). Si son iguales, posición a posición, incrementar el contador de aciertos (*hitsX*). Si todas las posiciones son iguales habremos ganado el juego; si no continuaremos jugando.

Os damos implementada la subrutina principal del juego, *playP1*. Esta subrutina inicializa las variables del juego, ejecuta un bucle para leer la combinación secreta hasta que la combinación secreta sea correcta. A continuación, ejecuta otro bucle para leer las jugadas que se repite mientras queden jugadas (actualiza la variables de los intentos restantes) o hasta que se acierta la combinación secreta. Finalmente muestra la combinación secreta en pantalla y un mensaje indicando el motivo por el cual ha finalizado el juego.

Vosotros sólo tenéis que implementar las subrutinas que se llaman desde *playP1*. No tenéis que modificar esta subrutina.

Para dividir la implementación del juego en cada una de las subrutinas que hay que realizar y facilitaros la comprobación de su funcionamiento, el programa C genera un menú principal con diferentes opciones.

Os recomendamos que vayáis desarrollando el código siguiendo el orden de estas opciones del menú hasta llegar a la opción que permite jugar utilizando todas las funcionalidades anteriores. Cada opción permite comprobar el funcionamiento de cada subrutina de forma independiente.

En esta primera parte el juego del MasterMind no estará completamente implementado. Faltará por implementar la funcionalidad que muestra cuantos aciertos en su sitio y fuera de sitio hay en cada jugada. En la segunda parte opcional se implementarán las funcionalidades necesarias para tener un juego totalmente funcional.

SEGUNDA PARTE OPCIONAL:

En la segunda parte se tendrán que implementar las funcionalidades adicionales necesarias para completar todas las funcionalidades del juego del MasterMind.

Además, habrá que trabajar el paso de parámetros entre las diferentes subrutinas, modificando la implementación hecha en la primera parte.

Os proporcionaremos también dos archivos para esta segunda parte: MMp2c-es.c y MMp2-es.asm. De forma parecida a la primera parte, el código C no lo podéis modificar. Sólo tendréis que implementar en ensamblador nuevas funcionalidades y habrá que modificar las subrutinas que habréis hecho en la primera parte para trabajar el paso de parámetros entre las subrutinas de ensamblador y también con las funciones de C.

El 3 de mayo de 2019 os daremos los programas .c y .asm correspondientes a la 2ª parte opcional.

Fechas de entrega y formato

La **primera parte** se puede entregar antes de las **23:59 del día 12 de abril de 2019** para obtener una puntuación de prácticas que puede llegar a una B. Si en esta fecha se ha hecho la entrega de la primera parte de manera satisfactoria se puede entregar la segunda parte antes de las **23:59 del 17 de mayo de 2019** para poder llegar a una puntuación de A en las prácticas.

En cambio, si no se ha podido hacer la primera entrega o esta primera entrega no ha sido satisfactoria se puede hacer una **segunda entrega** antes de las **23:59 del 17 de mayo de 2019**. En esta segunda fecha de entrega se puede entregar la primera parte, para obtener una calificación máxima de C+, o ambas partes (la práctica completa), para obtener una calificación máxima de B.

Este esquema de entregas y calificación se puede resumir en la siguiente tabla.

Primera Entrega 12-04-2019	Primera parte Superada	Primera parte NO Superada NO Presentada	Primera parte Superada	Primera parte NO Superada NO Presentada	Primera parte NO Superada NO Presentada
Segunda Entrega 17-05-2019	Segunda parte NO Superada NO Presentada	Primera parte Superada	Segunda parte Superada	Primera parte Superada Segunda parte Superada	Primera parte NO Superada NO Presentada
Nota Final Práctica	B	C+	A	B	D/N

Los alumnos que no superen la PRÁCTICA tendrán un suspenso (calificación 2-3) o N si no se ha presentado nada, en la nota final de prácticas y con esta nota no se puede aprobar la asignatura, por este motivo la PRÁCTICA es obligatoria.

La entrega se tiene que hacer a través de la aplicación **Entrega y registro de EC** del aula. Se tiene que entregar sólo un fichero con el código ensamblador bien comentado.

Es importante que el nombre de los ficheros tenga vuestros apellidos y nombre con el formato:

apellido1_apellido2_nombre_P1.asm

Fecha límite de la primera entrega:

Vienes, 12 de abril de 2019 a las 23:59

Fecha límite de la segunda entrega:

Viernes, 17 de mayo de 2019 a las 23:59

Criterios de valoración

La práctica tiene que funcionar completamente para considerarse superada, y hay que implementar todas las funcionalidades pedidas en el enunciado. **La opción del menú correspondiente al juego completo en ensamblador tiene que funcionar correctamente.**

Las otras opciones del menú son sólo para comprobar individualmente cada una de las subrutinas que se tienen que implementar.

No es suficiente para aprobar la práctica que las opciones correspondientes a las subrutinas individuales funcionen.

Dado que se trata de hacer un programa, sería bueno recordar que las dos virtudes a exigir, por este orden son:

- **Eficacia:** que haga el que se ha pedido y tal como se ha pedido, es decir, que todo funcione correctamente según las especificaciones dadas. Si las subrutinas no tienen la funcionalidad pedida, aunque la práctica funcione correctamente, se podrá considerar suspendida.
- **Eficiencia:** que lo haga de la mejor forma. Evitar código redundante (que no haga nada) y demasiado código repetido (que el código sea compacte pero claro). Usar modos de direccionamiento adecuados: los que hagan falta. Evitar el uso excesivo de variables y usar registros para almacenar valores temporales.

IMPORTANTE: Para acceder a los vectores en ensamblador se tiene que utilizar direccionamiento relativo o direccionamiento indexado: [vSecret+eax], [vSecret+edi]. No se pueden utilizar índices con valores fijos para acceder a los vectores.

Ejemplo de lo que **NO** se puede hacer:

```
mov BYTE [vSecret+0], 0
mov BYTE [vSecret+1], 0
mov BYTE [vSecret+2], 0
...
```

Y repetir este código muchas veces.

Otro aspecto importante es la documentación del código: que clarifique, dé orden y estructura, que ayude a entenderlo mejor. No se tiene que explicar que hace la instrucción (se da por supuesto que quién la lee sabe ensamblador) si no que se tiene que explicar porque se usa una instrucción o grupo de instrucciones (para hacer qué tarea de más alto nivel, relacionada con el problema que queremos resolver).

Implementación

Cómo ya hemos dicho, la práctica consta de una parte de código en C que os damos hecho y **NO PODÉIS MODIFICAR** y un programa en ensamblador que contiene algunas subrutinas ya implementadas y las subrutinas que tenéis que implementar. Cada subrutina de ensamblador tiene una cabecera que explica que hace esta subrutina y cómo se tiene que implementar, indicando las variables, funciones de C y subrutinas de ensamblador que se tienen que llamar.

No tenéis que añadir otras variables o subrutinas.

Para ayudaros en el desarrollo de la práctica, en el fichero de código C encontraréis implementado en este lenguaje las subrutinas que tenéis que hacer en ensamblador para que os sirvan de guía durante la codificación.

En el código C se hacen llamadas a las subrutinas de ensamblador que tenéis que implementar, pero también encontraréis comentadas las llamadas a las funciones de C equivalentes. Si queréis probar las funcionalidades hechas en C lo podéis quitando el comentario de la llamada de C y poniéndolo en la llamada a la subrutina de ensamblador.

Por ejemplo, en la opción 1 del menú hecho en C hay el código siguiente:

```
//=====
printTriesP1();
//printTriesP1_C();
//=====
```

El código llama a la subrutina de ensamblador `posCurScreenrP1()`, podemos cambiar el comentario y llamar a la función de C.

```
//=====
//printTriesP1();
printTriesP1_C();
//=====
```

Recordad volver a dejar el código como estaba para probar vuestras subrutinas.

Las subrutinas siguientes de la Primera Parte ya están implementadas y NO las tenéis que modificar:

gotoxyP1
printchP1
getchP1
printMessageP1
playP1

Las subrutinas que hay que implementar en ensamblador para la Primera Parte son:

printTriesP1
getSecretPlayP1
checkSecretP1
checkPlayP1
printSecretP1

Consultad en el archivo de código ensamblador (MMp1.asm) cual debe ser el funcionamiento de cada una de las subrutinas que tenéis que implementar y que variables utilizan.