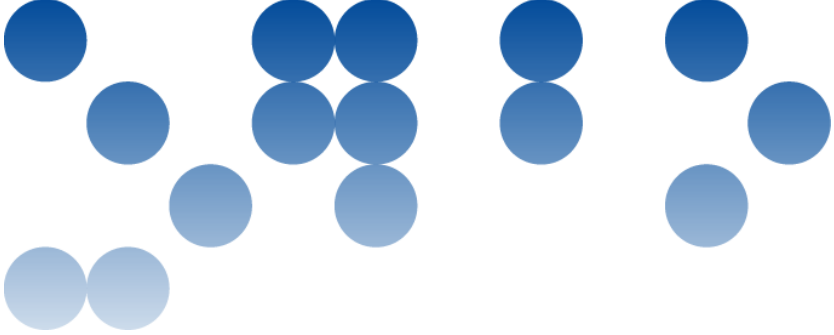

PRÁCTICA 1a parte: Buscaminas

Estructura de Computadores

Grado en Ingeniería Informática

set19-feb20

Estudios de Informática, Multimedia y Telecomunicaciones

EIMT.UOC.EDU

Universitat Oberta de Catalunya

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: El Buscaminas

La práctica consiste en implementar el juego del “BuscaMinas” que consiste en encontrar donde están las mines en un tablero de 10 x 10 casillas sin abrir ninguna casilla que contenga una mina. Se pueden marcar las casillas donde creemos que hay una mina. Si se abre una casilla que no tiene mina se indicará cuantas minas hay en las 8 casillas de alrededor, con esta información tenemos que ser capaces de encontrar donde están todas las minas. Si se abre una casilla que tiene una mina, se pierde la partida. El funcionamiento es parecido al “BuscaMinas” de Windows.

La práctica consta de un programa en C, que os damos hecho y que NO TENÉIS QUE 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.

La práctica se ha dividido en dos partes:

PRIMERA PARTE OBLIGATORIA:

Para esta primera parte os proporcionamos dos archivos: BMp1c-es.c y BMp1-es.asm. El código C (BMp1c-es.c) no lo tenéis que modificar sólo tenéis que implementar en ensamblador en el archivo BMp1-es.asm las funcionalidades necesarias.

Para dividir la implementación del juego en cada una de las subrutinas que tenéis que implementar y facilitaros la comprobación de su funcionamiento, el programa C genera un menú principal con 10 opciones:

- posCurScreen, llama a la subrutina posCurScreeenP1. Posiciona el cursor en la pantalla dentro del tablero, en función del índice de la matriz (indexMat), posición del cursor dentro del tablero.
- showMines, llama a la subrutina showMinesP1. Convierte el valor del número de minas que quedan por marcar, numMines, valor entre 0 i 99, en dos caracteres ASCII. Muestra en la parte inferior del tablero las mines que quedan por marcar, inicialmente quedan 18.
- updateBoard, llama a la subrutina updateBoardP1. Actualiza el contenido del tablero de juego con los datos de la matriz *marks* y el número de minas que queden por marcar llamando a la subrutina showMinesP1.
- moveCursor, llama a la subrutina moveCursorP1. Actualiza la posición del cursor en el tablero que tenemos indicada con la variable (indexMat), en función de la tecla pulsada que tenemos en la variable (charac). Si se sale fuera del tablero no actualizar la posición del cursor.
- mineMarker, llama a la subrutina mineMarkerP1. Marca/desmarca una mina en la matriz (marks) en la posición actual del cursor, indicada en la variable (indexMat).

- checkMines, llama a la subrutina checkMinesP1. Verifica si hemos marcado todas las mines. Si numMines es 0, cambia el estado del juego para indicar que hemos ganado.
- Play Game, llama a la subrutina playP1. Permite probar todas les funcionalidades llamando a las subrutinas de ensamblador que hay que implementar en esta primera parte, con la tecla 'ESC' se puede salir del juego y volver al menú.
- Play Game C, permite probar todas las funcionalidades llamando a las funciones de C que os damos hechas en esta primera parte, con la tecla 'ESC' se puede salir dels juego y volver al menú, **con esta opción podéis ver el funcionamiento del juego**.
- Exit, finalizar el programa.

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 Buscaminas no estará completamente implementado. 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 Buscaminas.

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: BMp2c-es.c y BMp2-es.asm. De forma parecida a la primera parte, el código C no lo tendréis que 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 29 de noviembre 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 8 de noviembre 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 29 de noviembre 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 29 de noviembre 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 08-11-2019	**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 29-11-2019	**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, 8 de noviembre de 2019 a las 23:59

Fecha límite de la segunda entrega:

Viernes, 29 de noviembre de 2019 a las 23:59

Criterios de valoración

La práctica tiene que funcionar completamente para considerarse superada, y hay que implementar todas las funcionalidad pedidas en el enunciado, l**a 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 modas 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: [marks+eax], [marks+edi]. No se pueden utilizar índices con valores fijos para acceder a los vectores.

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

```
mov BYTE [marks+0], 0

mov BYTE [marks+1], 0

mov BYTE [marks+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:

```
//=====================================================
posCurScreenP1();
//posCurScreenP1_C();
//=====================================================
```

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

```
//=====================================================
//posCurScreenP1();
posCurScreenP1_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`
`playP1`

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

`posCurScreenP1`
`showMinesP1`
`updateBoardP1`
`moveCursorP1`
`mineMarkerP1`

`checkMinesP1`

Consultad en el archivo de código ensamblador (BMp1.asm) cual debe ser el funcionamiento de cada una de las subrutinas que tenéis que implementar y que variables utilizan.