LABORATORI DE SISTEMES OPERATIUS

Jordi Garcia
Pedro Marcuello
Àlex Ramírez
Toni Cortés

Pràctica Zero: Introducció a l'entorn de treball

Inici

- En iniciar els ordinadors, sel.leccioneu la opció *Novell Network* del menú.
- Identifiqueu-vos com a usuari `fibs`. No us demanarà cap password.
- En el directori *w:/soft/assig/so/p0* trobareu els fitxers necessaris per fer la pràctica. També hi trobareu el fitxer `ip.bat`. Heu d'executar aquesta comanda cada cop que entreu a la xarxa (també per la pràctica d'Onion).
- Les unitats de disc estan protegides contra escriptura. Heu de fer la pràctica en el vostre disc flexible (podeu copiar el contingut del directori al vostre disc executant la comanda *xcopy /s *.* a:* des del directori *w:/soft/assig/so/p0*).

Contingut

1 Compilació de programes i gestió de llibreries

Directori de treball: 1_tlib

Coses a fer

1. Genereu el fitxer executable `main.exe`. Genereu primer tots els fitxers .OBJ fent servir el TCC. Genereu amb TCC el fitxer executable fent servir els fitxers objecte que heu generat (no useu TLINK). No genereu el fitxer executable directament a partir dels fitxers font.

   Examineu el contingut de `files.txt`. Podeu fer servir `@files.txt` en lloc d'especificar tots els fitxers en la línia de comandes. Això es útil per generar l'executable quan la línia de comandes es massa llarga per al DOS.

2. Genereu una llibreria anomenada `arithmet.lib` que contingui les rutines add, sub, multiply, power usant el gestor de llibreries TLIB. No podeu usar `@files.txt` amb el gestor de llibreries. Sols funciona amb el TCC.

3. Genereu un llistat del contingut de la llibreria `arithmet.lib`.

4. Genereu el fitxer executable usant la llibreria `arithmet.lib` en lloc dels deiferents fitxers .OBJ que hi hem posat.

5. Escriviu un fitxer `factoria.c` que contingui la rutina *factorial(a)* per calcular nombres factorials. Afegiu aquesta rutina a la llibreria `arithmet.lib` sense tornar a generar tota la llibreria.

6. Modifiqueu el fitxer `multiply.c` per tal que faci servir la multiplicació en C en lloc de la funció *add()*. Modifiqueu la funció *multiply()* de la llibreria sense tornar a generar tota la llibreria.

   Nota: donat que la rutina *multiply()* ja no fa servir les rutines *add()* i *sub()*, els fitxers de capçalera `add.h` i `sub.h` ja no son necessaris, i podem eliminar les directives *#include* del fitxer `multiply.c`

2 Importància dels fitxers de capçalera

Directori de treball: 2_header

Coses a fer

1. Genereu els fitxers `header.obj` i `add.obj`. Comproveu que el compilador no genera cap error, tot i que el fitxer `header.c` no fa servir la capçalera de la rutina add().

2. Genereu el fitxer executable `header.exe`. Comproveu que tampoc no es produeix cap error. Comproveu, però, que l'execució del programa no és correcte.

3. Torneu a compilar els fitxers `.obj` afegint la opció *-wpro* en la crida al compilador. Solucioneu els warning i els errors que apareixen.

4. Genereu l'executable, i comproveu que l'execució del programa és correcte.

   Nota: a partir d'aquest punt, es recomanable que useu el flag *-wpro* sempre que hagueu de compilar un fitxer. Tingueu en compte que TCC diferencia entre *rutina()* (rutina amb qualsevol nombre de parametres, sols serveix per definir el tipus que retorna la rutina), i *rutina(void)* (rutina sense cap parametre). Les capceleres del primer tipus seguiran generant un warning quan useu el flag *-wpro*, sense que aixo provoqui cap error.

3 Gestió de projectes (I)

Directori de treball: 3_make

Coses a fer

1. Executeu la comanda *make* per generar el fitxer executable `main.exe`. Examineu el contingut del fitxer Makefile, i raoneu quin es el procés seguit per la eina *make*.

2. *make* fa servir la data de modificació dels fitxers per saber quin dels dos és més recent. La comanda *touch* modifica la data de modificació d'un fitxer, i la fixa a la data actual. Modifiqueu la data de modificació del fitxer `multiply.obj`, i torneu a executar *make*.

3. Comproveu que quan es modifica qualsevol dels fitxers font (sigui `.c` o `.h`) sols es compilen aquells fitxers que en depenen. Els fitxers `.obj` que no en depenen es poden fer servir tal i com estaven.

4. Perque cal recompilar `multiply.obj` quan hem modificat `add.h`?

5. Modifiqueu el fitxer `sub.h`, i torneu a executar *make*. Perque no hem compilat `sub.obj`?

6. Pot ser això un problema?

4 Gestió de projectes (II)

Directori de treball: 4_make

Coses a fer

1. Fent servir la opció *-Ipath* del compilador TCC, escriviu un makefile per tal de generar l'executable `main.exe`. No està permés canviar l'estructura de fitxers i directoris original (fitxers `.h` al directori `include`, i llibreries al directori `lib`). El vostre makefile ha de tenir el mateix funcionament i característiques que l'exemple de la secció anterior: només ha de compilar aquells fitxers que calguin, i ha de mantenir totes les dependncies entre fitxers `.c` i `.h`

2. Modifiqueu el vostre makefile per tal que generi la llibreria `arithmet.lib` directament en el directori `lib`. Només pot haver-hi una copia de la llibreria en acabar de generar-se l'executable.

3. Modifiqueu el vostre makefile fent servir la opció *-Lpath* del compilador TCC per tal que faci servir la llibreria `arithmet.lib` del directori `lib` en lloc dels fitxers `.obj`. El makefile ha de mantenir les mateixes característiques que al començament, es a dir, si es modifica algun dels fitxers `.c` o `.h`, caldrà recompilar els `.obj` que calguin, actualitzar la llibreria, i tornar generar l'executable.

   Nota: A partir d'aquest punt, es obligatori (i molt recomanable) que useu *make* sempre que hagueu de compilar fitxers, encara que siguin pocs. A la llarga sempre us serà més cómode.

5 Relació de rutines C i ASM

Directori de treball: 5_asm

Coses a fer

1. Genereu el fitxer objecte `main.obj` i l'executable `main.exe`.

2. Tal i com hem generat els fitxers, no es possible veure el codi assemblador que ha generat el compilador. Torneu a generar els fitxers, afegint la opció -S al compilador. Observeu que s'ha generat un fitxer `main.asm` que conte el codi assemblador.

3. Examineu el fitxer `main.asm`. Observeu com s'han declarat les variables globals a, b, c. Observeu també com s'han accedit aquestes variables dins el codi.

4. Quin efecte té sobre la declaració de les variables a, b i c l'opció de compil.lació `-r-`?

5. Modifiqueu el fitxer `main.c` per tal que les variables a, b, c passin a ser locals. Torneu a generar el fitxer `main.asm`. Observeu com es declaren ara les variables, i com s'accedeix a elles.

6. Observeu en el fitxer `main.asm` com es fa la crida a la rutina suma(). En particular, observeu els noms de les rutines en assemblador, com es passen els paràmetres a la rutina, i com es recull el resultat de la rutina.

7. Observeu com dins de la rutina suma() es recullen els paràmetres que se li han passat, com genera el resultat, i com el passa a la rutina que l'ha cridat.

8. Torneu a generar els fitxers afegint les opcions *-k -ml* al compilador. Observeu què ha canviat en el procés de crida a subrutines en afegir aquestes opcions.

   Nota: En aquest punt es important la distinció entre un procediment `near` i un procediment `far`. En un procediment `near`, la direcció de retorn es composa únicament del desplaçament (registre IP), mentre que en un procediment `far` calen tant el segment com el desplaçament (registres CS i IP). Els flags *-k -ml* obliguen al TCC a considerar que tots els procediments son `far`, cosa que ens fa més fàcil combinar programes en C amb rutines en ASM.

   A partir d'aquest punt, useu els flags *-k -ml* sempre que hagueu de compilar un fitxer C (no tenen cap efecte si estem compilant un fitxer ASM).

9. Esborreu la rutina suma() del fitxer `main.c`. Escriviu un nou fitxer `suma.c` que tan sols contingui la rutina que acabem d'esborrar. Torneu a generar els fitxers. Examineu el codi assemblador, i comproveu si ha canviat alguna cosa.

10. Esborreu el fitxer `suma.c`, i escriviu un fitxer `suma.asm` que implementi la rutina suma(). Feu que el fitxer `main.c` cridi a la rutina que heu implementat en assemblador.

    Podeu utilitzar el fitxer `plantill.asm` com a base per al fitxer `suma.asm` que heu de generar.

11. Modifiqueu la vostre rutina suma() per tal que cridi a una nova rutina add() que serà la que realment farà la suma. Programeu aquesta rutina en assemblador, dins del propi fitxer `suma.asm`.

12. Genereu un tercer fitxer `add.c` que contindrà una versió en C de la rutina add(). Feu que la rutina suma() (programada en assemblador) cridi a la rutina add() que heu programat en C.

    Nota: L'ordre dels fitxers en la línia de comandes pot ser important per al funcionament del programa.

6 Depuració de programes

Directori de treball: 6_debug

Coses a fer

1. Escriviu un Makefile per tal de generar el fitxer `test.exe`.

2. Executeu el debugger (TD) passant com a paràmetre l'executable que voleu depurar (en aquest cas `test.exe`). Comproveu que la informació presentada d'aquesta manera no ens es gaire útil.

3. Modifiqueu el Makefile i afegiu el flag *-v*. Torneu a generar tots els fitxers `.obj`, així com l'executable. Torneu a executar el debugger. El flag *-v* ha afegit informació a l'executable que permet al debugger ensenyar-nos el codi font (sigui en C o ASM).

   Nota: el debugger es una eina molt util en el desenvolupament de la pràctica d'Onion. Es molt recomanable que useu el flag *-v* sempre que hagueu de compilar un fitxer.

4. Executeu el programa amb la opció RUN (o be amb F9). Canvieu al mode visió de pantalla (Alt+F5) per veure la sortida per pantalla del programa.

5. Reinicieu l'estat del debugger (opció RESET, o Ctrl-F2).

6. Executeu una sola línia de codi fent un STEP (F8). Podeu executar el codi pas a pas d'aquesta manera. Executeu el codi línia a línia fins arribar a la crida a la rutina outer(). Si ara feu un altre STEP, passareu a la següent línia de codi sense que el debugger entri dins la subrutina.

7. En lloc de fer un STEP, feu un TRACE (F7). El debugger passa a executar la següent línia de codi, però entrant dins de la subrutina. Podem seguir executant el programa pas a pas sense entrar a les subrutines fent un STEP, entrant dins les subrutines fent un TRACE, o be completament amb un RUN.

8. Obriu el mòdul `inner.c` (Open-Module, o Alt+F3). Ara podem veure el codi font del fitxer, tot i que l'execució del programa encara no hi ha arribat. Avanceu el cursor (la línia ressaltada) fins la primera línia de codi dins del bucle mes intern. Afegiu un BREAKPOINT (F2). La línia de codi ha quedat marcada.

9. Executeu el programa amb RUN (F9). Comproveu que en comptes d'executar el programa sencer, l'execució d'atura en la línia de codi que hem marcat amb un BREAKPOINT. Per avançar mes ràpidament en l'execució del programa podem afegir tants BREAKPOINTS com ens facin falta, i anar passant d'un a l'altre executant el programa amb RUN (F9).

   Nota: Afegir un BREAKPOINT (F2) es tot sovint la única manera pràctica d'arribar fins a una subrutina. En aquest cas, el nombre de comandes STEP que caldrien fins arribar a la rutina inner() es molt elevat. Durant la pràctica d'Onion, un BREAKPOINT es la única manera de fer que l'execució arribi fins a l'interior d'una rutina de servei a la interrupció (RSI).

10. Examineu el valor de la variable `acum` afegint un WATCH (Ctrl-F7). Mitjançant un WATCH podeu controlar el valor de qualsevol variable del programa, fins i tot vectors, estructures, i apuntadors. Genereu un programa de proves que tingui diferents tipus de dades (array, struct, union, array de struct, etc), i afegiu un WATCH per cadascun d'ells.