Execució de processos

José María Alonso Cebrián
Jordi Gay Sensat
Antonio Guzmán Sacristán
Pedro Laguna Durán
Alejandro Martín Bailón
Jordi Serra Ruiz

PID_00196797

UOC
Universitat Oberta de Catalunya

Índex

Introducció

Durant l'execució d'un procés se succeeixen una sèrie d'accions que el sistema ha d'anar controlant per a la finalització correcta del programa. Durant l'execució d'aquest, el sistema ha de donar accés a la CPU a l'aplicació, accés als dispositius, gestionar les autoritzacions, gestionar la memòria assignada al procés, gestionar els arxius a què s'ha accedit...

Durant l'execució d'un programa es creen i utilitzen diferents zones de memòria per a desar cadascun dels components del programa que seran necessaris per a l'execució. Aquestes zones de memòria són les següents:

- Codi.
- Àrea de dades dinàmiques (*heap*).
- Àrea de dades estàtiques (*bss*).
- Pila.

1. Codi

Quan comença l'execució d'un programa, el sistema operatiu carrega el programa en una zona de memòria lliure. A més es creen les zones de dades enumerades anteriorment en altres zones de memòria que seran associades al programa en execució.

Normalment, la zona de codi té una mida prefixada. La zona de dades estàtiques tampoc no creix, ja que està predeterminada en temps de compilació. Les dues àrees que poden créixer són l'àrea de dades dinàmiques i la pila. La zona de dades dinàmiques creix envers adreces més altes de memòria i la pila envers adreces més baixes. Aquestes àrees s'han de posicionar en memòria de manera adequada, perquè no s'acabin superposant.

En l'inici del programa es prepara l'entorn perquè es pugui començar a executar la primera instrucció del programa. Això s'aconsegueix apuntant el registre EIP del processador (encarregat d'apuntar a la instrucció per executar) a l'adreça d'entrada del programa carregat (*entry point* o primera instrucció per executar). També és necessari inicialitzar adequadament els apuntadors a la pila i les àrees de memòria.

2. Àrea de dades dinàmiques (*heap*)

L'assignació dinàmica de memòria és el procés pel qual el sistema assigna zones de memòria lliures a diferents parts del programa executat. Aquestes requereixen zones de memòria lliure durant l'execució del programa sense haver fet una declaració explícita de mida en el codi font, per a poder fer la seva comesa. Cada llenguatge té les seves pròpies funcions per a demanar memòria al sistema. Per exemple, en C es disposa de `malloc`.

```
char *buf = malloc(SIZE)
```

Aquesta línia reservaria un espai de memòria de mida `SIZE`, al qual s'accediria per mitjà del punter assignat a la variable `buf`. Per a poder alliberar aquest espai de memòria s'hauria d'utilitzar la funció `free`.

```
free(buf)
```

Altres llenguatges de programació disposen d'altres funcions per a fer aquesta comesa. Una vegada s'ha assignat una porció de memòria a un programa, aquesta roman reservada fins que el programa l'allibera explícitament. Possibles errors en la programació de programes poden provocar que la memòria reservada no s'alliberi (*memory leak*), i així el sistema queda amb parts de memòria que no s'utilitzarien, ja que estarien reservades. Això podria provocar que els sistemes, després d'un cert període, quedessin inutilitzats pel fet de no disposar de memòria lliure, a causa que seria tota "usada" pels diferents processos del sistema.

Per a evitar aquest problema els sistemes operatius disposen de processos que es dediquen a buscar parts de memòria reservades que no tinguin cap referència des de cap dels programes en execució. Aquests processos es denominen *garbage collectors*. En cas de trobar espais de memòria que no són utilitzats per cap procés, es procedeix a alliberar-los. Cada sistema operatiu, o fins i tot cada llenguatge de programació (llenguatges interpretats com C# o Java), té la seva pròpia manera de gestionar la memòria.

3. Àrea de dades estàtiques (*bss*)

En aquesta àrea es desen totes les dades que són conegudes en temps de compilació. Habitualment són les variables globals, les variables definides com a *static* i les constants. Es podria dir que es desen en aquest espai dades que han de ser accessibles diverses vegades durant l'execució del programa i no solament durant l'execució d'una funció.

```
static char buf[SIZE]
```

La sentència anterior defineix una matriu de caràcters de manera estàtica, amb la qual cosa es reservaria espai en memòria per a aquest a la zona *bss*.

Les variables locals, no definides com a *static*, es desen en la pila. No cal accedir-hi després de l'execució de la funció, per la qual cosa poden ser destruïdes completament i perden els seus valors i definició. La pila és el lloc perfecte per a crear aquest comportament durant l'execució d'un programa.

De vegades es denomina aquesta zona de memòria *bss* (*block started by symbol*). Aquesta nomenclatura té l'origen en una pseudooperació en el FORTRAN *assembly program*, desenvolupat per IBM. Posteriorment aquesta nomenclatura es va anar heretant entre sistemes fins als nostres dies.

Els *exploits* que exploten errors en l'accés a aquesta àrea de memòria es denominen *bss-based overflows*. Si per contra exploten vulnerabilitats en l'accés a l'àrea de memòria del *heap* es denominen *heap-based overflows*. Com que la manera d'explotar totes dues regions de memòria s'assembla, moltes vegades s'ajunten en un sol nom: *heap/bss-based overflows*.

4. Pila

La pila és una zona de memòria reservada per a desar tot tipus de dades temporals de manera ordenada, mantenint la coherència de les dades per a poder fer crides a funcions i el retorn d'aquestes de manera correcta. La pila creix sempre envers adreces de memòria més baixes i comença en adreces de memòria altes.

En la pila s'introdueixen elements per a fer càlculs o per a fer crides a funcions, juntament amb els seus paràmetres i variables locals. D'aquesta manera la pila va creixent, i donat el cas, pot arribar a un límit en el qual el sistema operatiu detindrà el programa per haver sobrepassat la mida màxima de la pila (desbordament de pila). Hi ha *exploits* que provoquen aquest desbordament. Això es pot donar si es programa una funció recursiva amb un nombre massa elevat de crides, o si simplement està mal programada i no té especificada la condició de finalització de la recursivitat. Una altra possibilitat és que les variables locals utilitzades siguin massa grans. Això pot provocar que després de diverses crides se sobrepassi la mida màxima de la pila; per això és recomanable que les variables que necessitin quantitats grans de memòria reservin el seu espai en memòria de manera dinàmica. D'aquesta manera, tan sols es desarà un punter en la pila i les dades se situaran en el *heap*.

En mòduls anteriors hem vist, de manera esquemàtica, com es gestiona la pila. A continuació mostrarem un exemple d'un programa que fa diverses crides a funcions, i fa el seguiment de les dades de la pila. El programa que s'analitza és el següent:

```
#include <stdio.h>

int funcio2(int a) {
   int tmp2;

   printf ("Inici funcio2: %d\n", a);
   tmp2 = a+2;
   printf ("Funcio2: tmp2 = %d\n", tmp2);
   return tmp2;
}

int funcio1(int b) {
   int tmp1;

   printf ("Inici funcio1: %d\n", b);
   tmp1 = funcio2(b);
   tmp1 = tmp1 + 1;
   printf ("Funcio1: tmp1 = %d\n", tmp1);
   return tmp1;
}

int main() {
   int tmp_main;

   tmp_main = 7;
   printf ("Valor inicial = %d\n", tmp_main);
   tmp_main = funcio1(tmp_main);
   printf ("Resultat = %d\n", tmp_main);
   return 1;
}
```

Aquest programa fa primer una crida a la `funcio1` i aquesta, al seu torn, crida a la `funcio2`. El codi en assemblador de les diferents funcions del programa anterior, compilat amb el GCC, és el següent:

```
Funció principal main

0x08048421 <main+0>:          lea        0x4(%esp),%ecx
0x08048425 <main+4>:          and        $0xfffffff0,%esp
0x08048428 <main+7>:          pushl      -0x4(%ecx)
0x0804842b <main+10>:         push       %ebp
0x0804842c <main+11>:         mov        %esp,%ebp
0x0804842e <main+13>:         push       %ecx
0x0804842f <main+14>:         sub        $0x24,%esp
0x08048432 <main+17>:         movl       $0x7,-0x8(%ebp)
0x08048439 <main+24>:         mov        -0x8(%ebp),%eax
0x0804843c <main+27>:         mov        %eax,0x4(%esp)
0x08048440 <main+31>:         movl       $0x8048594,(%esp)
0x08048447 <main+38>:         call       0x80482d8 <printf@plt>
0x0804844c <main+43>:         mov        -0x8(%ebp),%eax
0x0804844f <main+46>:         mov        %eax,(%esp)
0x08048452 <main+49>:         call       0x80483de <funcio1>
0x08048457 <main+54>:         mov        %eax,-0x8(%ebp)
0x0804845a <main+57>:         mov        -0x8(%ebp),%eax
0x0804845d <main+60>:         mov        %eax,0x4(%esp)
0x08048461 <main+64>:         movl       $0x80485a8,(%esp)
0x08048468 <main+71>:         call       0x80482d8 <printf@plt>
0x0804846d <main+76>:         mov        $0x1,%eax
0x08048472 <main+81>:         add        $0x24,%esp
0x08048475 <main+84>:         pop        %ecx
0x08048476 <main+85>:         pop        %ebp
0x08048477 <main+86>:         lea        -0x4(%ecx),%esp
0x0804847a <main+89>:         ret

Funcio1

0x080483de <funcio1+0>:      push       %ebp
0x080483df <funcio1+1>:      mov        %esp,%ebp
0x080483e1 <funcio1+3>:      sub        $0x18,%esp
```

```
0x080483e4 <funcio1+6>:      mov        0x8(%ebp),%eax
0x080483e7 <funcio1+9>:      mov        %eax,0x4(%esp)
0x080483eb <funcio1+13>:     movl       $0x804856a,(%esp)
0x080483f2 <funcio1+20>:     call       0x80482d8 <printf@plt>
0x080483f7 <funcio1+25>:     mov        0x8(%ebp),%eax
0x080483fa <funcio1+28>:     mov        %eax,(%esp)
0x080483fd <funcio1+31>:     call       0x80483a4 <funcio2>
0x08048402 <funcio1+36>:     mov        %eax,-0x4(%ebp)
0x08048405 <funcio1+39>:     addl       $0x1,-0x4(%ebp)
0x08048409 <funcio1+43>:     mov        -0x4(%ebp),%eax
0x0804840c <funcio1+46>:     mov        %eax,0x4(%esp)
0x08048410 <funcio1+50>:     movl       $0x804857f,(%esp)
0x08048417 <funcio1+57>:     call       0x80482d8 <printf@plt>
0x0804841c <funcio1+62>:     mov        -0x4(%ebp),%eax
0x0804841f <funcio1+65>:     leave
0x08048420 <funcio1+66>:     ret

Funcio2

0x080483a4 <funcio2+0>:      push       %ebp
0x080483a5 <funcio2+1>:      mov        %esp,%ebp
0x080483a7 <funcio2+3>:      sub        $0x18,%esp
0x080483aa <funcio2+6>:      mov        0x8(%ebp),%eax
0x080483ad <funcio2+9>:      mov        %eax,0x4(%esp)
0x080483b1 <funcio2+13>:     movl       $0x8048540,(%esp)
0x080483b8 <funcio2+20>:     call       0x80482d8 <printf@plt>
0x080483bd <funcio2+25>:     mov        0x8(%ebp),%eax
0x080483c0 <funcio2+28>:     add        $0x2,%eax
0x080483c3 <funcio2+31>:     mov        %eax,-0x4(%ebp)
0x080483c6 <funcio2+34>:     mov        -0x4(%ebp),%eax
0x080483c9 <funcio2+37>:     mov        %eax,0x4(%esp)
0x080483cd <funcio2+41>:     movl       $0x8048555,(%esp)
0x080483d4 <funcio2+48>:     call       0x80482d8 <printf@plt>
0x080483d9 <funcio2+53>:     mov        -0x4(%ebp),%eax
0x080483dc <funcio2+56>:     leave
0x080483dd <funcio2+57>:     ret
```

Després d'executar el programa, s'inspecciona la pila just abans de la primera crida a la funció `printf` (`main+38`). L'estat de les dades que s'hi troben és el següent:

```
0xbfa597d0: 0x08048594      ESP (Paràmetre 2 printf)
0x00000007                  (Paràmetre 1 printf)
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  EBP (Conté antic EBP)
```

El paràmetre 2 de `printf` conté un punter a una adreça de memòria on hi ha desada la cadena de caràcters que se li passa com a format de sortida. Si s'inspecciona la memòria en aquesta adreça, es pot veure el següent:

```
0x8048594: "Valor inicial = %d\n"
```

Es continua amb l'execució i es para just abans de la crida a la funció 1. L'anàlisi de la pila en aquest moment és:

```
0xbfa597d0: 0x00000007      ESP (Paràmetre 1 funcio1)
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  EBP (Conté antic EBP)
```

S'observa com s'ha preparat la pila per a executar la crida a la `funcio1`. S'ha col·locat en el *top* de la pila el paràmetre necessari perquè la `funcio1` s'executi correctament. Es fa la crida a la `funcio1` i es torna a analitzar la pila. S'avança fins a abans de la crida a la `funcio2` per a analitzar el contingut de la pila en aquest moment.

```
0xbfa597b0: 0x00000007      ESP (Paràmetre 1 funcio2)
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0xb7f1dff4                  Variable local tmp1 (sense valor)
0xbfa597f8                  EBP (antic EBP) F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funcio1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

Es pot observar l'adreça de retorn a la funció `main` que s'usarà quan acabi la `funcio1` per a retornar l'execució al punt següent on s'havia interromput. Es veu també on se situa la variable local `tmp1`. Igual que la preparació de la crida a la `funcio1`, es poden observar els paràmetres que es passaran a la `funcio2`. Es procedeix a continuació a entrar en la `funcio2` i avançar fins al moment en què s'ha d'incrementar el valor del paràmetre en 2 unitats.

```
0xbfa59790: 0x08048540      ESP (antic paramet. a printf)
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x0804856a                  Espai reservat i no usat
0xbfa597a0: 0xbfa597b8      Espai reservat i no usat
0xb7f1dff4                  Variable local tmp2 (sense valor)
0xbfa597c8                  EBP (antic EBP) F2
0x08048402                  Adreça de retorn a funcio1
0xbfa597b0: 0x00000007      Paràmetre 1 funcio2
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0xb7f1dff4                  Variable local tmp1 (sense valor)
0xbfa597f8                  Antic EBP F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funcio1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

En cada crida a funció s'observen els antics paràmetres a la funció `printf`. Això succeeix perquè el compilador ha decidit reservar un espai en la pila per a les variables locals i altres usos, entre els quals s'inclou el pas de paràmetres a funcions. Com que el programa té espai lliure en la pila per a posar aquests paràmetres no es fa cap *push* en la pila d'aquests, i simplement s'escriuen en les posicions de memòria que haurien d'ocupar en la pila en fer la crida. Després de la crida a la funció aquests paràmetres continuen existint, ja que no es destrueixen aquestes posicions de memòria fins que s'acaba cada funció.

Es passa ara a incrementar el valor del paràmetre i a desar-lo en la variable local `tmp2`. Això es fa amb l'execució de les línies de codi:

```
0x080483c0 <funcio2+28>:        add $0x2,%eax
0x080483c3 <funcio2+31>:        mov %eax,-0x4(%ebp)
```

El resultat de l'execució d'aquest codi és l'actualització de la variable local en la pila.

```
0xbfa59790: 0x08048540      ESP (antic paràm. a printf)
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x0804856a                  Espai reservat i no usat
0xbfa597a0: 0xbfa597b8      Espai reservat i no usat
0x00000009                  Variable local tmp2 (nou valor)
0xbfa597c8                  EBP (antic EBP) F2
0x08048402                  Adreça de retorn a funcio1
0xbfa597b0: 0x00000007      Paràmetre 1 funcio2
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0xb7f1dff4                  Variable local tmp1 (sense valor)
0xbfa597f8                  Antic EBP F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funció1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
Antic EBP Main
```

A partir d'aquest moment es torna a fer una crida a `printf` i finalment es prepara el retorn del valor calculat a la funció anterior. En aquest cas el valor calculat es retorna en el registre EAX. El contingut del registre s'actualitza amb la sentència següent:

```
0x080483d9 <funcio2+53>:      mov 0x4(%ebp),%eax
```

Aquest codi accedeix a la pila, on hi ha situada la variable local, i assigna el valor desat en aquesta posició de memòria al registre EAX. Finalment tan sols queda la finalització de la funció i devolució de l'execució al punt posterior de la crida feta en la funció pare. Això es fa amb les instruccions `leave` i `ret`. Aquestes instruccions s'expliquen amb més detall en propers mòduls però, bàsicament, `leave` serveix per a alliberar l'espai reservat en la pila per la funció cridada (variables locals) i recuperar el valor d'EBP anterior. Aquesta instrucció seria equivalent a executar:

```
mov %ebp, %esp
pop %ebp
```

En fer això, en la part superior de la pila quedaria l'adreça de retorn, que seria usada per la instrucció `ret`. L'execució de `ret` equivaldria a executar:

```
pop %eip
```

Encara que aquesta instrucció no és vàlida en assemblador, serveix per a explicar què és el que fa la instrucció `ret`. Es passa ara a executar el codi fins a la instrucció `leave`.

```
0xbfa59790: 0x08048555      ESP (antic paràm. a printf)
0x00000009                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x0804856a                  Espai reservat i no usat
0xbfa597a0: 0xbfa597b8      Espai reservat i no usat
0x00000009                  Variable local tmp2 (nou valor)
0xbfa597c8                  EBP (antic EBP) F2
0x08048402                  Adreça de retorn a funcio1
0xbfa597b0: 0x00000007      Paràmetre 1 funcio2
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0xb7f1dff4                  Variable local tmp1 (sense valor)
0xbfa597f8                  Antic EBP F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funcio1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

Aquest estat de la pila canvia just després d'executar la instrucció `leave`.

```
0xbfa597ac: 0x08048402      ESP (Dir. de retorn a fun. 1)
0xbfa597b0: 0x00000007      Paràmetre 1 funcio2
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0xb7f1dff4                  Variable local tmp1 (sense valor)
0xbfa597f8                  Antic EBP F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funcio1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

L'adreça de retorn queda en la part superior de la pila. La instrucció *ret* agafa el que hi hagi en el *top* de la pila i retorna l'execució del programa a aquesta adreça. L'estat de la pila després de la instrucció `ret` és exactament igual que just abans d'haver fet la crida a la `funcio2`. Aquest estat ja s'ha vist anteriorment. El resultat retornat per la `funcio2` es troba en el registre EAX. A continuació la `funcio1` assigna el resultat a la variable local `tmp1` que es troba en la pila. La instrucció que fa això és:

```
0x08048402 <funcio1+36>:      mov %eax,-0x4(%ebp)
```

i l'estat final de la pila després d'aquesta acció és el següent:

```
0xbfa597b0: 0x00000007      ESP (Paràmetre 1 funcio2)
0x00000007                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0x00000009                  Variable local tmp1 (nou valor)
0xbfa597f8                  EBP (antic EBP) F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funcio1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

A continuació, s'incrementa aquest valor en una unitat i es prepara EAX per a retornar el resultat a la funció que ha fet la crida. Seguidament, es mostra la pila en l'estat en què queda abans de cridar la funció `leave`.

```
0xbfa597b0: 0x0804857f      ESP (antic param. a printf)
0x0000000a                  Antic paràmetre a printf
0xb7f1e4e0                  Espai reservat i no usat
0x08048594                  Espai reservat i no usat
0xbfa597c0: 0xbfa597d8      Espai reservat i no usat
0x0000000a                  Variable local tmp1 (nou valor)
0xbfa597f8                  EBP (antic EBP) F1
0x08048457                  Adreça de retorn a main
0xbfa597d0: 0x00000007      Paràmetre 1 funcio1
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x00000007      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

S'observa que ja s'ha incrementat el valor de la variable `tmp1`, i que en el *top* de la pila hi ha els últims paràmetres que s'han passat a l'anomenada funció `printf` perquè mostri per pantalla el resultat de les operacions després de l'execució del codi que modifica les variables.

Per acabar, es mostra com queda la pila en finalitzar l'execució de la `funcio2` i retornar l'execució a la funció principal `main`.

```
0xbfa597d0: 0x00000007      ESP (Paràmetre 1 funcio1)
0x00000007                  Antic paràmetre a printf
0xbfa597e8                  Espai reservat i no usat
0x080482a4                  Espai reservat i no usat
0xbfa597e0: 0xb7f1dff4      Espai reservat i no usat
0x080496a4                  Espai reservat i no usat
0xbfa59808                  Espai reservat i no usat
0x080484a9                  Espai reservat i no usat
0xbfa597f0: 0x0000000a      Variable local tmp_main
0xbfa59810                  Antic valor d'ECX
0xbfa59868                  Antic EBP Main
```

S'observa que ja s'ha actualitzat el valor de la variable local i tot l'espai de la pila que s'havia reservat per a cridar les diferents funcions ja s'ha alliberat. El mètode per seguir per a acabar el programa és exactament el mateix. Primer s'allibera l'espai ocupat per les variables locals i finalment es retorna l'execució a un altre punt.

S'ha pogut veure, pas per pas, com va evolucionant la pila juntament amb els paràmetres i les variables locals definides. Com a regla general es pot dir que l'accés a les variables locals i els paràmetres rebuts per una funció es fa utilitzant el registre EBP com a base, i per a passar paràmetres a una altra funció (maneig del *top* de la pila) s'usa el registre ESP. L'esquema de la memòria assignada a la pila seria:

paràmetres ...	← Esp
... variables locals ...	
EBP anterior	← EBP
Adreça de retorn (EIP)	
paràmetres ...	
... variables locals ...	
EBP anterior	
Adreça de retorn (EIP2)	
paràmetres ...	

Conèixer detalladament el funcionament de la pila (i de les estructures de memòria), és molt important a l'hora de crear o usar *exploits*, ja que la majoria aprofiten vulnerabilitats d'aplicacions que no controlen bé aquestes estructures.

5. Espai d'usuari i espai de sistema

Hi ha un conjunt de tasques que han de ser fetes pel sistema operatiu, com és l'accés a determinats recursos de manera controlada. Entre aquests recursos es pot trobar l'accés al disc o accés a dispositius d'entrada/sortida. És tasca del sistema operatiu controlar aquest accés i garantir que les diferents aplicacions puguin utilitzar els recursos.

Aquest conjunt de tasques s'ha de fer de manera totalment transparent a l'aplicació. Dins d'aquests entorns d'execució es troben dos modes d'execució:

- Mode supervisor (espai de sistema).
- Mode usuari (espai de sistema).

Els processos s'executen normalment en mode usuari, i quan necessiten accedir a determinats recursos als quals no tenen accés per defecte (gestionats pel sistema), fan una crida al sistema per aconseguir l'operació desitjada. Les crides al sistema són operacions que ofereix el sistema operatiu a les aplicacions perquè puguin utilitzar els recursos disponibles. La manera de gestionar aquestes crides és totalment diferent en sistemes Windows i sistemes Linux. Quan es fa la crida, el sistema s'encarrega que l'operació sol·licitada s'executi correctament i retorna el resultat a l'aplicació que ha fet la sol·licitud. D'aquesta manera, es garanteix que l'accés als diferents recursos serà feta de manera correcta, per part d'una màquina utilitzada per múltiples usuaris, ja que només hi ha un àrbitre que controla l'accés als recursos: el sistema operatiu.