

# CSD İŞLETİM SİSTEMİNDE SİSTEM FONKSİYONLARININ ÇAĞIRILMASI

07.07.2003

Burak DEMİRKOL – Atılım BOY

[v 1.0]

*Özet : Bu makalede CSD İşletim sisteminin sistem fonksiyonlarının user mod'dan (ring 3) kernel mod'a (ring 0) geçilerek çağırılma mekanizması anlatılacaktır.*

## 1. SİSTEM FONKSİYONU KAVRAMI

Sistem fonksiyonları, işletim sisteminin dış dünya ile bağlantısını sağlayan aşağı seviyeli fonksiyonlardır. *UNIX/LINUX* terminolojisinde kernel moda geçilerek çalıştırılan *API* fonksiyonlarına *sistem fonksiyonu* denilmektedir.

İşletim sisteminin aşağı seviyeli çalışma şekli ve organizasyonu, sistemin kararlılığı ve güvenilirliği açısından önemlidir. Her şey işletim sisteminin kontrolü altında, güvenli ve kararlı bir şekilde yürütülmelidir. Bütün bu koruma mekanizmalarına rağmen işletim sistemi dış dünya ile iletişimini kaybetmemeli ve çeşitli isteklere cevap verebilmelidir. İşletim sisteminin kendi belirlediği sınırlar içerisinde bu isteklere cevap verebilmesi için programcılara sunduğu fonksiyonlar ise sistem fonksiyonlarıdır. Bir işletim sisteminde olması beklenen sistem fonksiyonları *POSIX* standartlarında belirtilmiştir. Bu standartlara uygun olarak yazılan işletim sistemlerindeki *API* fonksiyonları aynı olacağı için *POSIX API* fonksiyonları kullanılarak yazılan programlar taşınabilir olacaktır. *CSD* işletim sistemi de *POSIX* standartlarına uygun bir işletim sistemi olacaktır.

Bütün bunlara ek olarak her işletim sisteminin kendisine özgü, programcılara sunduğu bazı sistem fonksiyonları olabilir. Bunlar da o işletim sisteminin karakteristik özelliklerini belirler. Örneğin *GUI* tabanlı bir işletim sistemi, kendine özgü grafik ara yüzü üzerinde her türlü işlemin kolayca yapılabilmesi için çeşitli sistem fonksiyonları sunabilir. Buradan da anlaşıldığı gibi sistem fonksiyonları, işletim sistemi üzerinde yazılan programların taşınabilirliği, kolay uygulama geliştirilebilir olması, sağlamlığı ve güvenilirliği açısından önemlidir.

## 2. SİSTEM FONKSİYONLARININ ÇAĞIRILMA MEKANİZMASI

*CSD* işletim sisteminde, bir *API* fonksiyonu çağırıldığında çağırılan *API* fonksiyonu sistem fonksiyonunu çağırır. Buradan da anlaşılacağı gibi aslında programcının çağırıldığı *API* fonksiyonu sarma (wrapper) bir fonksiyondur. *CSD* kaynak kodlarında sistem fonksiyonlarının isimlerinin başında *SYS\_* öneki vardır. Örneğin *exit()* *API* fonksiyonu kendi içerisinde sistem fonksiyonunu aşağıdaki gibi çağırır:

```
mov    eax, 1
xor    ebx, ebx
int    80h
```

Yukarıdaki kod'da sırayla, *eax'a* çağrılacak gerçek sistem fonksiyonu numarası, *ebx'e* birinci parametre koyulur. Eğer varsa diğer parametreler sırayla *ecx, edx, ...* yazmaçlarına aktarılır. Daha sonra *int 80h* kesmesi çağrılır. Bu kesmeye işletim sistemi tarafından bir tuzak kapısı (trap gate) yerleştirilmiştir. Bu kesme çağırıldığında tuzak kapısı yolu ile akış işletim sisteminin belirlediği bir noktaya gelir. Bu nokta *CSD* işletim sisteminde *SystemCall\_ASM* fonksiyonudur.

### 3. SİSTEM FONKSİYONU ÇAĞIRILDIĞINDA NE OLUR?

Bir API fonksiyonu çağırıldığında tuzak kapısı yolu ile akış işletim sisteminin belirlediği noktaya gelir. Örneğin *UNIX/LINUX* işletim sistemlerinde 80h kesmesine bir tuzak kapısı yerleştirilmiştir. 80h kesmesi çağırıldığında tuzak kapısı yolu ile akış işletim sisteminin belirlediği noktaya gelir. *CSD* işletim sisteminde bu nokta *entry.asm* içerisindeki *SystemCall\_ASM* fonksiyonudur. Bu noktaya gelindiğinde artık *CPU* ring 0'da çalışmaktadır. *syscall.asm* dosyasının içerisindeki kodun tamamı makalenin bir sonraki bölümünde verilecektir. Bu bölümde *entry.asm* dosyasındaki kodlar incelenecektir.

```
SystemCall_ASM
push    eax          ; save system call number
```

Bu noktada *eax* içerisindeki değer *stack'te* saklanmıştır. Bir önceki konuda anlatıldığı gibi *eax* yazmacı içerisinde çağrılacak sistem fonksiyonunun numarası vardır.

```
SAVE_ALL          ; save es, ds, eax, ebp, edi, esi, edx, ecx, ebx
```

Bu bir makrodur. Açılımı ise aşağıdaki gibidir;

```
%macro SAVE_ALL    0
    push    es
    push    ds
    push    eax
    push    ebp
    push    edi
    push    esi
    push    edx
    push    ecx
    push    ebx
    mov    edx, KERNEL_DATA_SEL
    mov    ds, edx
    mov    es, edx
%endmacro
```

Bu noktada tüm yazmaçlar ters sırada stack'de saklanmıştır. Bir sistem fonksiyonu çağrılmadan önce tüm yazmaçlar stack'de saklanmalı, sistem fonksiyonundan çıktıktan sonra akış user mod'a geçerken bu yazmaçlar geri yüklenmelidir. *SAVE\_ALL* makrosu tüm yazmaçların saklanması işini yapar. Makalenin bir önceki bölümünde sistem fonksiyonunun çağrılma mekanizması anlatılırken yazmaçların içerisinde sistem fonksiyonunun parametreleri olduğu anlatılmıştı. Buradan da anlaşılacağı gibi tüm yazmaçların saklanması demek aynı zamanda parametre olarak kullanılan yazmaçların da saklanması demektir.

Bu noktada dikkat edilmesi gereken önemli bir durum daha vardır o da *SAVE\_ALL* makrosu içerisinde parametrelerin stack'e ters sırada push edilmesidir. Bilindiği gibi assembler içerisinde *\_cdecl* çağrılma biçimine sahip bir C fonksiyonunu çağırarak için parametrelerin ters sırada push edilmesi gerekmektedir. Buradan da anlaşılacağı gibi aslında *SAVE\_ALL* makrosu ile saklanan yazmaçlar birazdan çağırılacak sistem fonksiyonuna parametre olarak geçirilmiştir.

Özetlemek gerekirse *SAVE\_ALL* makrosunda yapılan işlemin amacı şunlardır;

Sistem fonksiyonunu çağırılmadan önce tüm yazmaçları stack üzerinde saklanması, (Böylece sistem fonksiyonunun parametresi olarak kullanılan yazmaçların da stack üzerinde saklanması)

Bütün yazmaçların stack üzerine ters sırada push edilmesi ile bu yazmaçların birazdan çağırılacak C fonksiyonlarına parametre olarak geçirilebilmesi de sağlanmıştır.

```
cmp    eax, NSYS_CALLS
jae    BAD_SYSCALL
```

Yukarıdaki kod'da çağırılması istenen sistem fonksiyonu numarasının, toplam sistem fonksiyonundan büyük olup olmadığı kontrol edilmiştir. Eğer geçersiz bir sistem fonksiyonu numarası girilmiş ise akış *BAD\_SYSCALL* fonksiyonuna gidecektir. *BAD\_SYSCALL* fonksiyonu ise aşağıdaki gibidir;

```
BAD_SYSCALL:
mov    eax, -ENOSYS
jmp    RETURN_FROM_SYSCALL
```

Burada *ENOSYS* hata kodu *eax* yazmacına negatif olarak yerleştirilmiştir ve *RETURN\_FROM\_SYSCALL* fonksiyonu çağırılmıştır.

*RETURN\_FROM\_SYSCALL*:

```
RESTORE_ALL    ; restores ebx, ecx, edx, esi, edi, ebp, eax, ds, es
add    esp, 4
ret    ; return to user mode!
```

Bu noktaya gelindiğinde *RESTORE\_ALL* makrosu çağırılarak stack üzerinde saklanan yazmaçlar aynı sırada *pop* edilmiştir ve stack dengelenerek *ret* makine kodu ile tekrar user mod'a geri dönmüştür.

Eğer *eax* içerisindeki değer geçerli bir sistem fonksiyonu numarası ise akış aşağıdaki çağrı yapılarak devam edecektir;

```
call    dword [ga_systemCallTable + eax * 4]
```

*ga\_systemCallTable* bir fonksiyon gösterici dizisidir. Bütün sistem fonksiyonlarının fonksiyon pointer'ları bu dizinin bir elemanıdır ve dizinin her elemanı 4 byte uzunluğundadır. *ga\_systemCallTable* bu dizinin başlangıç adresi olduğuna göre her bir fonksiyon göstericisine;

*n*, sistem fonksiyon numarası olmak üzere:

```
ga_systemCallTable + n * 4
```

şeklinde ulaşırız. *call* makine komutu ile erişilen pointer'ı kullanarak sistem fonksiyonu çağırılmıştır. Sistem fonksiyonundan çıkıldığında akış aşağıdaki noktaya gelir.

```
mov     [esp + REG_EAX], eax      ; save return value in EAX
```

Sistem fonksiyonundan elde edilen geri dönüş değeri stack üzerindeki *eax* yazmacına yazılmıştır. Bilindiği gibi stack üzerine push edildiğinde *esp* yazmacının gösterdiği adres azalmaktadır. *REG\_* ile başlayan sabitler ise hangi yazmacın *esp* den ne kadar uzaklıkta olduğunu gösterir.

Akış aşağıya doğru devam ettiğinde *RETURN\_FROM\_SYSCALL* fonksiyonuna girmektedir ve user moda yani ring 3'e geri dönülmektedir.

## 4. İSKELET PROGRAM

Bu bölümde *CSD* sistem fonksiyon çağırma mekanizmasına ilişkin iskelet kodlar verilmektedir.

```
; -----
; FILE          : syscall.asm
; AUTHOR        : Kaan ASLAN
; LAST UPDATE   : 01/05/2003
; PLATFORM      : Win32/Linux
; TRANSLATOR    : nasm
;
;   Sekelton CSD System Call Implementation File
;
;   Copyleft (c) 1993 by C and System Programmers Association (CSD)
;   All Rights Free
; -----

[BITS 32]

#include "kernel.inc"
#include "port.inc"

;-----
; Symbolic Definitions
; -----

NSYS_CALLS EQU 128
```

```
ENOSYS      EQU      38
```

```
; Register Stack Offsets
```

```
REG_EBX     EQU      0x00  
REG_ECX     EQU      0x04  
REG_EDX     EQU      0x08  
REG_ESI     EQU      0x0C  
REG_EDI     EQU      0x10  
REG_EBP     EQU      0x14  
REG_EAX     EQU      0x18  
REG_DS      EQU      0x1C  
REG_ES      EQU      0x20
```

```
;-  
-
```

```
; Macro Definitions
```

```
;-
```

```
%macro SAVE_ALL      0  
    push    es  
    push    ds  
    push    eax  
    push    ebp  
    push    edi  
    push    esi  
    push    edx  
    push    ecx  
    push    ebx  
    mov     edx, KERNEL_DATA_SEL  
    mov     ds, edx  
    mov     es, edx  
%endmacro
```

```
%macro RESTORE_ALL   0  
    pop     ebx  
    pop     ecx  
    pop     edx  
    pop     esi  
    pop     edi  
    pop     ebp  
    pop     eax  
    pop     ds  
    pop     es  
    add     esp, 4  
%endmacro
```

```
;-
```

```
; System Call Routines
```

```
;-
```

```
[SECTION .text]
```

```
; --- Global Declarations --- ;
```

```
GLOBAL_C SystemCall_ASM  
GLOBAL_C ga_systemCallTable  
GLOBAL_C SetSystemCall_ASM
```

```
; --- System Call Entry Point --- ;
```

```

SystemCall_ASM:
    push    eax                ; save system call number
    SAVE_ALL    ; save es, ds, eax, ebp, edi, esi, edx, ecx, ebx
    cmp     eax, NSYS_CALLS
    jae    BAD_SYSCALL
    call   dword [ga_systemCallTable + eax * 4]
    mov    [esp + REG_EAX], eax    ; save return value in EAX

RETURN_FROM_SYSCALL:
    RESTORE_ALL ; restores ebx, ecx, edx, esi, edi, ebp, eax, ds, es
    iret     ; return to user mode!

BAD_SYSCALL:
    mov    eax, -ENOSYS
    jmp    RETURN_FROM_SYSCALL

;--- Set System Call --- ;

SetSystemCall_ASM:
    push    ebp
    mov    ebp, esp

    mov    ecx, [ebp + 8]
    mov    eax, [ebp + 12]
    shl   ecx, 2
    add   [ecx + ga_systemCallTable], eax

    pop    ebp
    ret

; --- NULL System Call --- ;

SYS_LINUX_ni_syscall:
    ret

[SECTION .data]

ga_systemCallTable:
    dd     SYS_LINUX_ni_syscall
    times (NSYS_CALLS - ($ - ga_systemCallTable) / 4 ) dd 0

```

## **Kaynaklar:**

1. CSD İşletim Sistemi Kernel Grubu Toplantıları, Kaan ASLAN – 2003
2. Assembly Programlama Dili Kurs Notları, Kaan ASLAN – 2003