İşletim Sistemlerine Giriş

Sistem Çağrıları(System Calls)

Sistem Çağrıları(System Calls)

İşletim sistemi ile kullanıcı programları arasında tanımlı olan arayüz, işletim sistemi tarafından tanımlanan bir prosedürler kümesidir.

İşletim sistemi tarafından tanımlanan bu prosedürlere sistem çağrıları(system calls) denilir.

Tanımlı olan işletim sisteminin sistem çağrıları kümesi işletim sistemlerinde farklı olabilir.

İsim olarak farklı olmasına rağmen arka planda gerçekleştirilen işlemler benzerdir.

Sistem Çağrıları(System Calls)

Kullanıcı programları donanım ile ya da özel işlemler için bu tanımlı sistem çağrılarını kullanırlar.

UNIX,System V,BSD,Minix,Linux türevi işletim sistemlerinde bulunması gerekli olan sistem çağrıları IEEE tarafından POSIX standartları olarak belirlenmiştir.

Bir sistem çağrısında gerçekleştirilecek olan işlemler makineye tamamen bağımlıdır bu yüzden sıklıkla assembly programlama dili ile tanımlanırlar.

C ve diğer programlama dillerinden bu prosedürleri kullanmak için prosedür kütüphaneleri tanımlanır.

Sistem Çağrıları(System Calls)

İşlenci aynı anda tek komut(instruction) çalıştırabilir.

Eğer kullanıcı kipinde çalışan bir kullanıcı programı bir servise ihtiyaç duyarsa, örneğin bir dosyayı okumak isterse, bir sistem çağrısı komutu(system call instruction) çalıştırarak kontrolü işletim sistemine vermelidir.

İşletim sistemi programın ne istediğini parametreleri inceleyerek belirler, sistem çağrısını yerine getirir ve kontrolü sistem çağrısını çağıran programa geri verir.

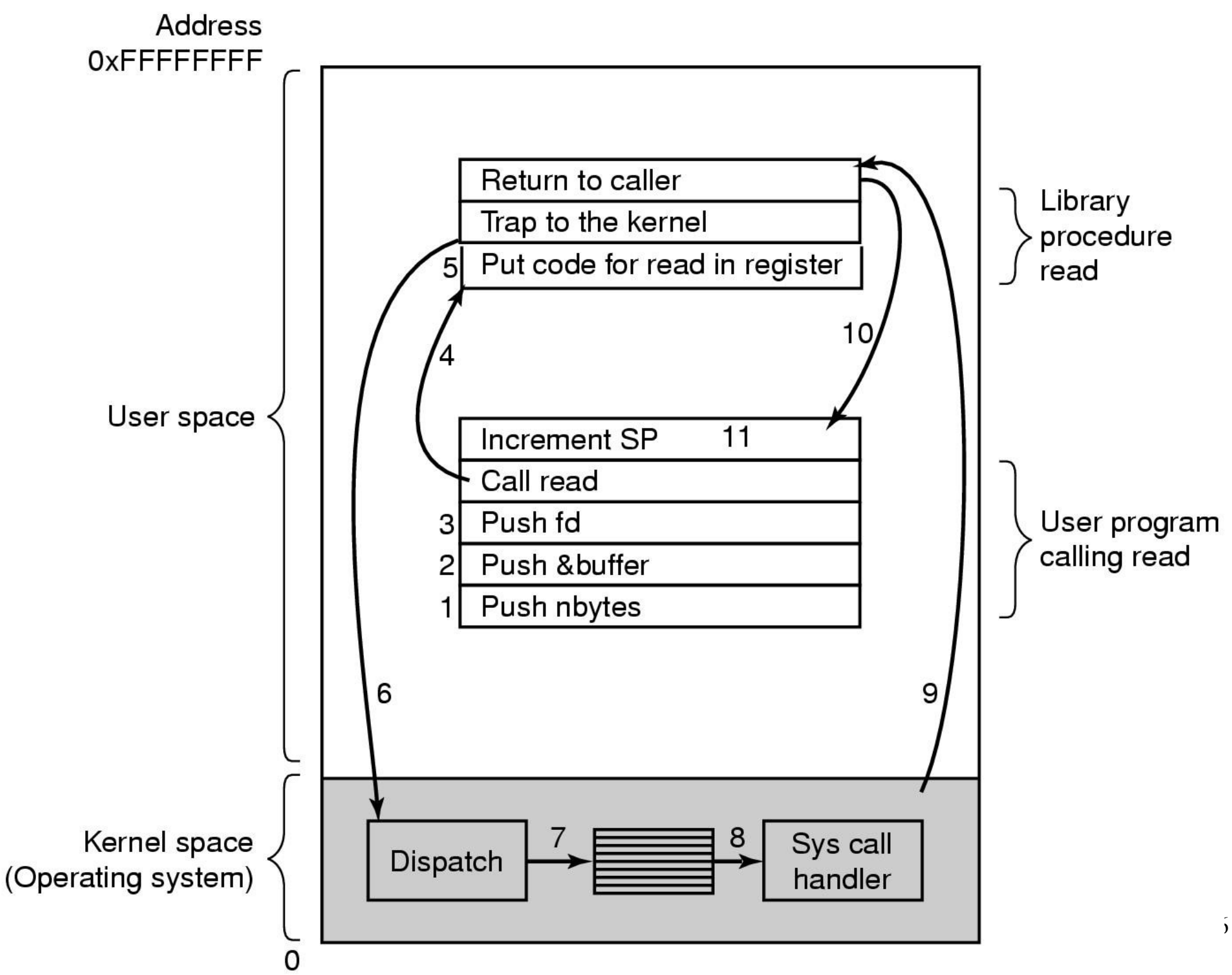
Sistem çağrısını çağırmayı, özel prosedürleri çağırma gibi düşünebiliriz.

Sistem Çağrıları(System Calls)

Örneğin, Unix sistemlerde bulunan ***read*** sistem çağrısını inceleyelim. Üç adet parametre alır. Birinci dosyayı belirtir, ikinci parametre tamponu belirtir ve üçüncü parametrede okunacak byte sayısını verir.

sayi = read(fd,tampon,okunacakByteSayısı);

Eğer sistem çağrısı doğru çalışmamışsa hata kodu ***errno*** isimli global değişkene yazılır ve geriye -1 çevrilir.


Address
0xFFFFFFFF
User space
Kernel space
(Operating system)
0
Return to caller
Trap to the kernel
5
Put code for read in register
4
10
Increment SP
11
Call read
3
Push fd
2
Push &buffer
1
Push nbytes
6
9
Dispatch
7
8
Sys call
handler
Library
procedure
read
User program
calling read

Sistem Çağrıları(System Calls)

1. Parametreler yığına(stack) konulur.
2. read sistem çağrısı için read kütüphane fonksiyonu çağrılır.
3. yazmaça read sistem çağrısına karşılık gelen değer konulur.
4. TRAP ile kontrol çekirdeğe verilir.
5. çekirdek hangi sistem çağrısının gerçekleştirileceğine parametre ve yazmaçdaki değeri inceleyerek karar verir. Uygun sistem çağrısı işleyicisini çalıştırır. Sistem çağrısı bittikten sonra çağıran kütüphane fonksiyonunu geri dönülür.
6. Kütüphane fonksiyonuda kendisini çağıran kullanıcı programına geri döner.

Sistem Çağrıları(System Calls)

POSIX tarafından tanımlanan, birkaç sistem çağrısını inceleyelim.

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Sistem Çağrıları(System Calls)

Bu isimler sistemlerdeki sistem çağrılarının isimlerini bire bir karşılamayabilir, POSIX sistemde bulunması gerekli olan çağrıları tanımlar, isimlerini tanımlamaz.

UNIX ve türevi olan sitemler için sistem çağrılarını biraz inceleyelim.

Süreç Yönetimi için Sistem Çağrıları

fork() sistem çağrısından başlayabiliriz. fork() sistem çağrısı UNIX sistemlerde yeni bir süreç oluşturmak için tek yoldur.

Bu fonksiyon asıl sürecin bire bir kopyasını oluşturur (dosya tanınlayıcıları, yazmaçlar,... herşey).

Kopyalama işleminden sonra iki süreç (ana ve çocuk) birbirlerinden tamamen ayrılırlar.

Kullandıkları veriler kendilerine özgü olur. (Fakat programın text kısmı aynı olduğu için iki süreç tarafından paylaşılır)

Süreç Yönetimi için Sistem Çağrıları

fork() fonksiyonu geriye bir değer döndürür.

Bu değer *çocuk süreçte 0* , *ana süreçte çocuk sürecin süreç numarası*dır.

Sistemdeki her sürecin bir süreç numarası bulunur. Bu süreç numarasına ***PID***(Process Identifier) denilir.

Birçok durumda çocuk süreçin farklı işler görmesi gereklidir.

Örneğin; bir kabuk(shell) da bir komut girdiğimizde kabuk ayrı, girilen komut ise ayrı işler gerçekleştirir.

Süreç Yönetimi için Sistem Çağrıları

ana süreç çocuk sürecin bitmesini beklemek isterse ya da beklemesi gerekli ise ***waidpid()*** isimli sistem çağrısını kullanır.

Linux sistemlerde ***pid*** ler 16 bit sayılardır ve Linux tarafından otomatik olarak atanır. Her sürecin mutlaka bir ana süreci vardır. (***init*** süreci Linux un ilk çalışan sürecidir, diğer tüm süreçler bunun çocuk süreçleridir)

Linux'da pid ler için, <***sys/types.h***> dosyasında tanımlı olan ***pid_t*** veri tipi kullanılır.
fork() gibi çeşitli süreç yönetim fonksiyonları <***unistd.h***> kütüphanesinde tanımlıdır.

Örnek (Linux)

print-pid.c

```
#include <stdio.h>
#include <unistd.h>

int main(){
      printf("süreç numarası : %d\n", (int)getpid() );
      printf("Ana sürecin süreç numarası:%d\n", (int)getppid() );
}
```

-getpid(): geriye sürecin süreç numarasını çevirir.
*-getppid():*çalışan sürecin ana sürecinin süreç numarasını geri çevirir. (get_parent_program_id)

Linux'da çalışan süreçleri ***ps*** komutu ile öğrenebilirsiniz.
$ps -e -o pid,ppid,command

fork() ve exec() in kullanımı

fork() : mevcut sürecin birebir kopyasını oluşturur, iki süreçde fork() fonksiyonundan sonraki satırdan itibaren kendi başlarına çalışmaya devam eder.

exec(): fonksiyonları kümesi, mevcut sürecin çalıştırmak istenilen başka bir programın sürecine dönüşmesini sağlar.

fork() ve exec() in kullanımı

fork.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
        pid_t cocuk_pid;

        printf("Ana sürecin pid = %d\n", (int)getpid() );

        cocuk_pid=fork();
        if (cocuk_pid!=0){
                printf("burası ana sürectir, süreç id pid=%d\n",(int)getpid());
                printf("çocuk sürecin idsi pid = %d\n",(int)cocuk_pid);
        }else{
                printf("burası çocuk süreçtir, pid=%d\n", (int)getpid());
        }
        return 0;
}
```

fork-exec.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
//fonksiyon yeni bir programı yumurtlar(spawn)
//yani calistirir, mevcut süreci bu programa cevirir
int spawn(char *program,char** arg_list){
    pid_t cocuk_pid;

    cocuk_pid =fork();
    if (cocuk_pid!=0){
          return cocuk_pid;
    }else{
          execvp(program,arg_list);
          //eger hata olmus ise alt kisim calisir
          fprintf(stderr,"execvp de hata olustu\n")
          abort();
    }
}
int main(){
    char * arg_list[]={"ls", "-l","/", NULL};
    spawn("ls",arg_list);
    printf("basarili olarak ana program bitti\n");
    return 0;
}
```

Süreçlerin Bellek Görünümü

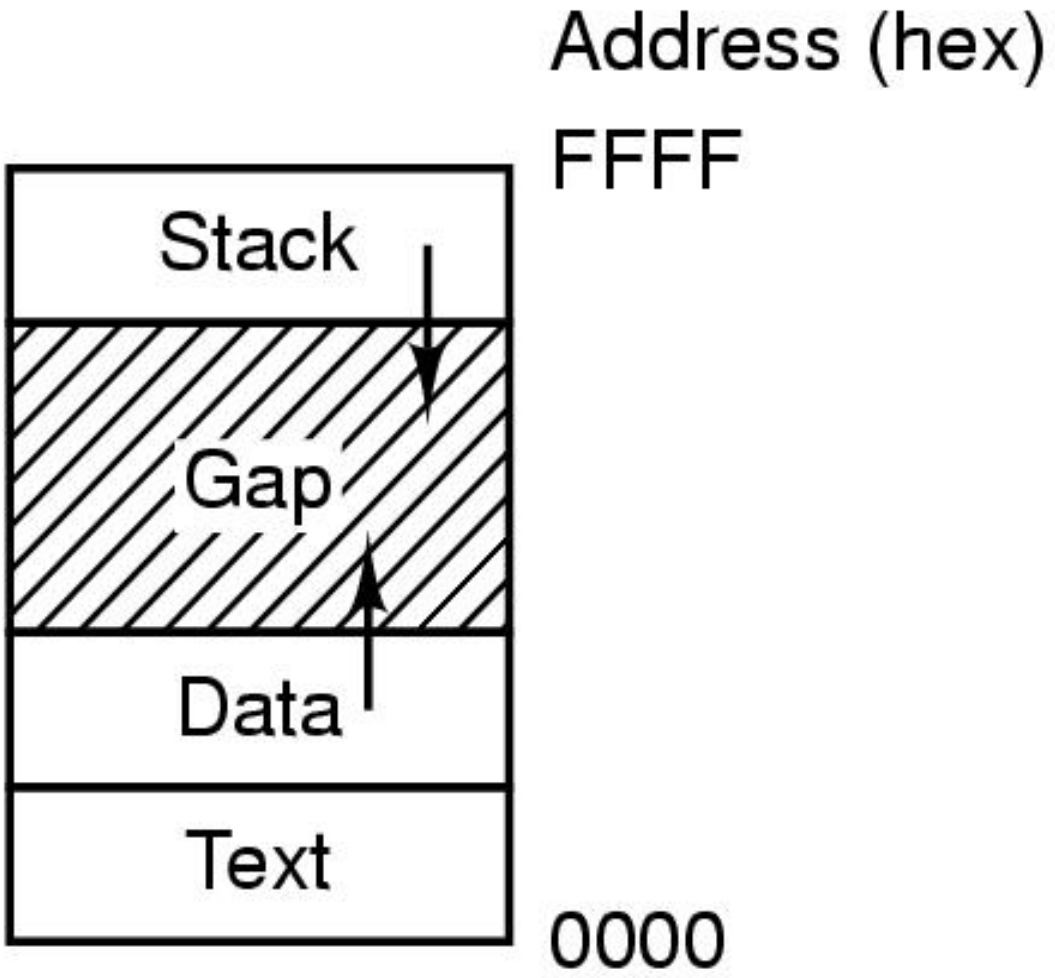
UNIX temelli sistemlerde bellek üç kesime(segment) parçalanmıştır.

Veri Kesimi(Data Segment) : değişkenler

Stack Kesimi(Stack Segment): malloc() ile ayrılan yerler

Yazı Kesimi(Text Segment): Program kodunun olduğu bellek bölümü

Süreçlerin Bellek Görünümü



Windows Win32 API

Programlama modelleri Windows ve Unix türevi işlerim sistemlerinin farklıdır.

Bir UNIX programı belirli bir görevi yerine getiren ve bu görevi yerine getirirken sistem çağrılarını kullanan bir programdır.

Bir Windows programı ise olay temellidir.Ana program belirli olayların gerçekleşmesini bekler. Bu olay gerçekleştiğinde de, olayı işleyecek(handle) olan prosedürü çalıştırır.

Windows Win32 API

Windowsunda sistem çağrıları bulunmaktadır. Unix sistemlerde genellikle sistem çağrısı ile çağrılacak olan kütüphane fonksiyonunun ismi aynıdır. Bu isimler POSIX tarafından tanımlanmıştır.

Windowsda durum bu şekilde değildir. Microsoft Win32 API (Application Programin Interface) adını verdiği bir prosedür kümesi tanımlamıştır. Programcılar işletim sisteminin servislerini kullanmak için bu prosedürleri kullanırlar. Bu arayüz tüm Windows işletim sistemleri tarafından kısmi olarak desteklenmektedir.

Yeni windows sistemlerinde bu prosedürler ve kullanımları farklılaştırılmaktadır.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

İşletim Sistemlerine Giriş

Sistem Çağrıları(System Calls)