Tutoriat 2

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout this text are generic

System Call Implementation

- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value — function name — parameters

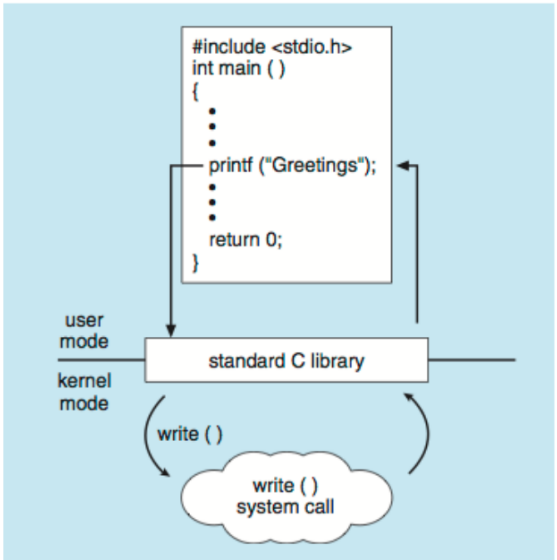
A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

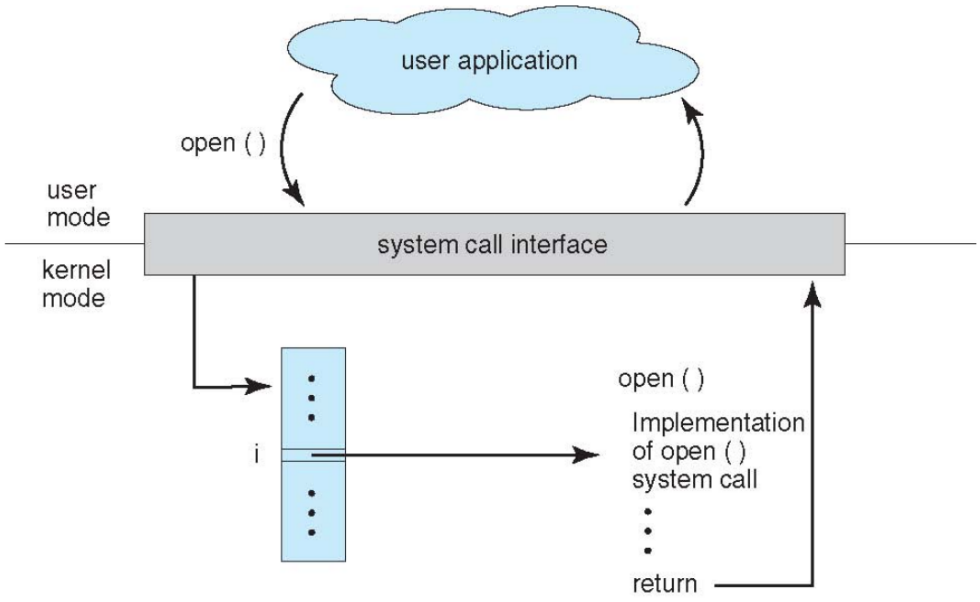
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

Standard C Library Example

- C program invoking printf() library call, which calls write() system call



API – System Call – OS Relationship



Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <sys/stat.h>

/**
* Functii de sistem
*  - Folosite pentru a accesa servicii de sistem
*  - Manual functii sistem: man 2 <syscall>
*  Tipuri de functii de sistem:
*   - Process control
*   - File management:
*    - int open(const char *path, int flags , ...);
*    - ssize_t read(int d, void *buf, size t nbytes);
*    - ssize_t write(int fildes, const void *buf, size_t nbyte);
*    - int stat(const char *path, struct stat *sb);
*   - Device management
*   - Information maintenance
*   - Communications
*   - Protection
*/


/// 1. Scrieti un program ce sa afiseze pe ecran 'sisteme de operare'
folosind functii de sistem
/// ssize_t write(int fildes, const void *buf, size_t nbyte);
//int main(){
//    write(2, "sisteme de operare", 18);
//}


/// 2. Scrieti un program care sa primeasca la intrare un fisier sursa si
sa afiseze continutul sau concatenat cu un text dat
/// Ex: cat2 hello again -> 'Hello world again'

// gcc main.c -o cat2
// ./cat2 hello again
```

```c
int openFile(char* fileName){
   int ff = open(fileName, O_RDONLY);
   printf("%d", ff);
   if(ff==-1){
       printf("Cannot open file!\n");
       return -1;
   }
   return ff;
}

int main(int argc, char** argv){
   if(argc!=3){
       printf("Wrong format, please use: ./cat2 <source> <text>\n");
       return -1;
   }
   char* fromFile = argv[1];
   int ff = openFile(fromFile);
   if(ff==-1)
       return -1;
   /// Folosind functia stat
   struct stat* buf = malloc(sizeof (struct stat));
   stat(fromFile, buf);
   char* content = malloc(sizeof (char) * buf->st_size);
   int readResult = read(ff, content, buf->st_size);
   if(readResult>0)
       write(1, content, readResult);
   /// Citind constant cate 1024 bytes
   //    const int DMAX = 1024;
   //    char* content = malloc(sizeof(char) * DMAX);
   //    int readResult;
   //    while((readResult = read(ff, content, DMAX))>0)
   //        write(1, content, readResult);
   write(1, argv[2], strlen(argv[2]));
   if(readResult==-1){
       printf("Error no: %d\n", errno);
       return -1;
   }
   close(ff);
   return 0;
```

```
}
```

Adaugati o functie de sistem noua cu antetul (*const char* src, char* dst, size_t len, size_t repeat)* astfel incat la adresa *dst* sa se scrie sirul de la adresa *src* de *repeat* ori. Sa se returneze marimea noului sir si sa se afiseze din kernel numarul de bytes copiati in *dst*.

/sys/kern/syscalls.master:

```
  GNU nano 2.9.4                /sys/kern/syscalls.master                  Modified

324     STD             { int sys_symlinkat(const char *path, int fd, \
                            const char *link); }
325     STD             { int sys_unlinkat(int fd, const char *path, \
                            int flag); }
326     OBSOL           t32_utimensat
327     OBSOL           t32_futimens
328     OBSOL           __tfork51
329     STD NOLOCK      { void sys___set_tcb(void *tcb); }
330     STD NOLOCK      { void *sys___get_tcb(void); }
331     STD             { int sys_repeat_str(const char* src, \
                            char* dst, size_t len, size_t repeat);}
```

/sys/kern/sys_generic.c:

```
int sys_repeat_str(struct proc *p, void *v, register_t *retval)
{
        struct sys_repeat_str_args *args = v;
        const char *src = SCARG(args, src);
        char *dst = SCARG(args, dst);
        size_t len = SCARG(args, len);
        size_t repeat = SCARG(args, repeat);

        char *k_src = malloc(len, M_TEMP, M_WAITOK);
        size_t done;
        copyinstr(src, k_src, len, &done);
```

```
        char *k_dst = malloc(len * repeat, M_TEMP, M_WAITOK);
        for(int i = 0; i < repeat; i++){
                for(int j = 0; j < len; j++){
                        k_dst[i * len + j] = k_src[j];
                }
        }
        copyoutstr(k_dst, dst, len * repeat, &done);
        *retval = done;
        free(k_src, M_TEMP, len);
```

```
        free(k_dst, M_TEMP, len * repeat);
        printf("Hi from kernel %zu\n", done);
        return 0;
}
```

Userland:

```
int main()
{
        char *str1 = "ceva"; // a se citi la runtime
        int rep = 3; // a se citi la runtime
        char *str2 = (char *)malloc(sizeof(char) * strlen(str1) * rep);
        int ret = syscall(331, str1, str2, strlen(str1), rep);
        printf("ret = %d\nresult string = %s\n", ret, str2);
        free(str2);
        return 0;
}
```

```
openbsd# gcc test.c -o test
openbsd# ./test
Hi from kernel 12
ret = 12
result string = cevacevaceva
openbsd#
```