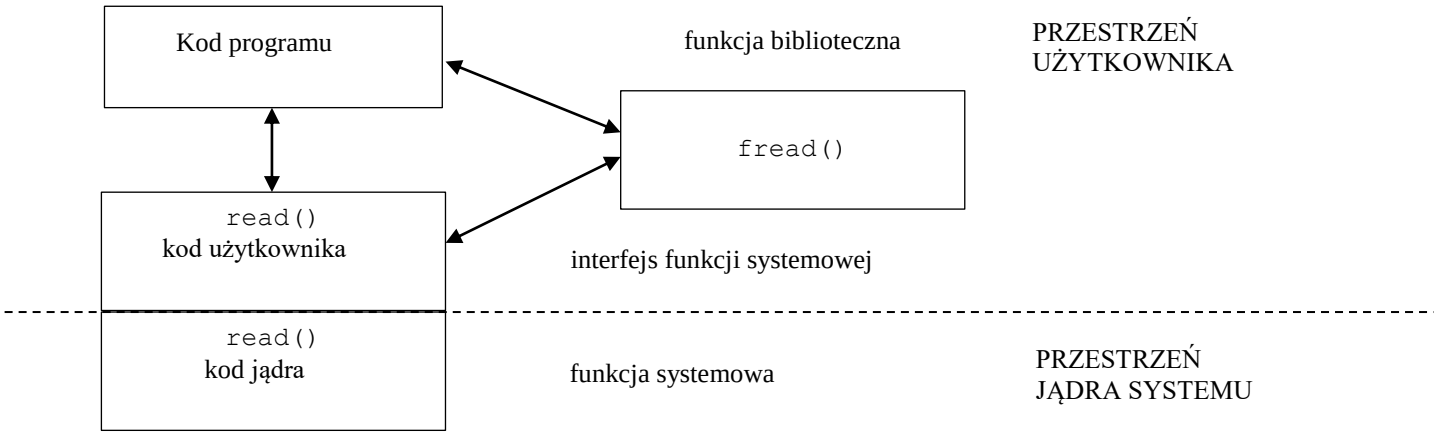
1 Funkcje systemowe

- *Funkcja biblioteczna*: zwykła funkcja znajdujące się w bibliotece, nie należącej do programu; wywołanie działa tak jak wywołanie zwykłej funkcji
- *Funkcja systemowa* (ang. *system call*) to funkcja, za pomocą której aktywny proces może uzyskać usługę ze strony jądra. Jest zaimplementowana w jądrze systemu. Wywołanie systemowe oznacza przekazanie sterowania jądru, któremu przekazywane są argumenty funkcji.
- *Wywołanie systemowe* (ang. *system call*) to interfejs do funkcji systemowej, za pomocą którego użytkownik może zasygnalizować, że chce wywołać jakąś funkcję systemową i przekazać do niej argumenty.
- Potocznie używając określenia funkcja systemowa mamy na myśli wywołanie systemowe.
- Przykład:

```
// czytanie z pliku za pomocą funkcji bibliotecznej C
nObj=fread(buforWe,rozmiarObj,ileObj,plikWsk);

// czytanie z pliku za pomocą funkcji systemowej
nBajtow=read(plikDeskryptor,buforWe,ileBajtow);
```

Zależność między funkcją biblioteczną i funkcją systemową:



1.1 Obsługa błędów funkcji systemowych

Numer	Nazwa	Opis
1	EPERM	Operation not permitted
2	ENOENT	No such file or directory
3	ESRCH	No such process
4	EINTR	Interrupted system call
5	EIO	I/O error

- Jeśli wystąpi błąd, to funkcja systemowa ustawia odpowiednią wartość w zmiennej `errno` i w większości wypadków zwraca wartość –1 do procesu wywołującego, sygnalizując powstanie błędu.
- Korzystanie ze zmiennej `errno` wymaga dołączenia pliku nagłówkowego errno.h.
- Zmienna `errno` nie jest zerowana podczas ponownego wywołania funkcji systemowej, w razie wystąpienia błędu jest po prostu zastępowana nową wartością, jeśli wystąpi błąd.
- **Funkcje biblioteczne korzystające z `errno`:**

```
#include <stdio.h>
void perror(const char *s);
```

Przesyła na `stderr` opis ostatniego błędu poprzedzony napisem `s` (o ile jest niepusty) i dwukropkiem.

```
#include <string.h>
char *strerror(int errnum);
```

Zwraca opis błędu o wskazanym numerze.

- Przykłady:

```
#include <string.h> // dla strerror()
#include <stdio.h> // dla printf()
#include <fcntl.h> // dla open()
#include <errno.h> // dla errno

int main() {
      int fd;
      fd=open("plik.txt",O_RDONLY);
      if (fd == -1) {
            fprintf(stderr,"open: %s\n", strerror(errno));
            exit(1);
      }
      /* działania na pliku */
      exit(0);
}
```

```
#include <stdio.h> // dla printf()
#include <fcntl.h> // dla open()
#include <errno.h> // dla errno

int main() {
      int fd;
      fd=open("plik.txt",O_RDONLY);
      if (fd == -1) {
            perror("open");
            exit(1);
      }
      /* działania na pliku */
      exit(0);
}
```

W obu przypadkach, jeśli pliku nie będzie otrzymamy komunikat:

```
open: No such file or directory
```

1.2 Dwa przykłady obsługi błędów

Mitchell, Oldham, Samuel: Linux Programowanie dla zaawansowanych: str. 31

A.

```
rval = chown(path, user_id, -1);
assert(rval==0)
```

B.

```
rval = chown(path, user_id, -1);
if (rval != 0) {
      int error_code = errno;
      assert (rval == -1);
      switch (error_code) {
            case EPERM:       /* Brak uprawnień */
            case EROFS:       /* Nazwa path w systemie plików
                                 tylko do odczytu */
            case ENAMETOOLONG: /* Nazwa path jest za długa */
            case ENOENT:      /* Plik path nie istnieje */
            case ENOTDIR:     /* Składnik nazwy path nie jest
                                 katalogiem */
            case EACCES:      /* Składnik nazwy path jest
                                 niedostępny */
            fprintf(stderr,
                  "błąd przy zmianie właściciela na %s: %s\n",
                  path, strerror(error_code));
            /*
            nie kończę programu,
            daje szansę użytkownikowi podać inną nazwę pliku
            */
                  break;
            case EFAULT:      /* Nazwa path odnosi się do
                              nieprawidłowego adresu pamięci.
                              Jest to prawdopodobnie
                              błąd w programie
                              */
                  abort();
            case ENOMEM:      /* Brak pamięci jądra systemu */
                  fprintf(stderr, "%s\n", strerror(error_code));
                  exit(1);
            default: /* Inny niespodziewany kod błędu */
                  abort();
      }
}
```

1.3 Użyteczne polecenia

- Polecenie `strace`: śledzi wykonanie innego programu i wypisuje wszystkie wykonane przez niego wywoływania i otrzymane sygnały.

Przykład:

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

#define MAX_BUF 128

int main() {
      int fd;
      char buf[MAX_BUF+1];
      fd = open("dane.txt",O_RDONLY);
      read(fd, buf, MAX_BUF);
      printf("przeczytano: %s\n",buf);
      close(fd);
      return 0;
}
```

Kompilacja:
```
gcc p1.c -Wall -o p1
```

Wykonanie:
```
./p1
przeczytano:8I▒\▒
```

Gdzie jest błąd?

```
strace ./p1
execve("./p1", ["./p1"], [/* 35 vars */]) = 0
… // pominięto ładowanie bibliotek
open("dane.txt", O_RDONLY) = -1     ENOENT (No such file or directory)
read(-1, 0xbffda00b, 128)  = -1     EBADF (Bad file descriptor)
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7790000
write(1, "przeczytano: \10\250\240\375\277\\zr\n", 22przeczytano:▒▒\zr
) = 22
close(-1)                       = -1              EBADF (Bad file descriptor)
exit_group(0)                   = ?
```

Gdzie jest błąd?