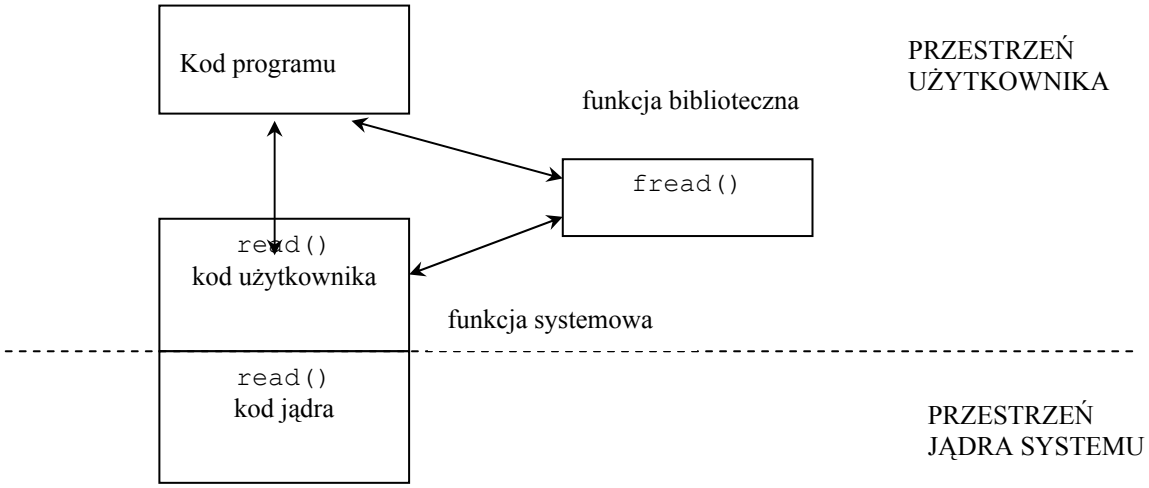
Funkcje systemowe

- *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.
- Funkcje systemowe są wykorzystywane do zarządzania procesami, obsługą sygnałów, usługami systemu plików, zarządzania pamięcią ...
- 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:



Użyteczne polecenia

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

Obsługa błędów funkcji systemowych

- Po wywołaniu funkcji interesuje nas:
  - Czy wywołanie zakończyło się powodzeniem?
  - Jeśli nie, to co było przyczyną błędu?
- W większości funkcji systemowych obowiązuje konwencja, że wartość zwracana oznacza powodzenie lub niepowodzenie wywołania funkcji.
  Dwie główne kategorie zwracanych wartości:
  - liczba typu int – niepowodzenie oznaczane jest w *większości* wypadków jako wartość –1
  - wskaźnik - niepowodzenie oznaczane jest w *większości* wypadków jako `NULL`

Ustalenie przyczyny błędu

Po każdym niepowodzeniu większość funkcji systemowych wstawia do zmiennej `errno` kod błędu. Korzystanie ze zmiennej `errno` wymaga dołączenia pliku nagłówkowego `errno.h`. Wartość `errno` jest aktualizowana tylko po wystąpieniu błędu.

- Przykłady kodów błędów

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`

Wyprowadzenie informacji o błędach

- Wyprowadzenie kodu błędu ze zmiennej `errno`
- Wykorzystanie tablicy komunikatów `sys_errlist[]`
- Wygenerowanie komunikatu na podstawie zmiennej `errno`:

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

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

- 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,"Nie udalo sie otworzyc pliku: %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("Nie udalo sie otworzyc pliku");
    exit(1);
  }
  /* działania na pliku */
  exit(0);
}
```

W obydwu przypadkach, jeśli pliku nie będzie otrzymamy komunikat:

```
Nie udalo sie otworzyc pliku: No such file or directory
```

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
                             nie dostępny */
      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();
  }
}
```