# Funkcje systemowe w systemie Linux

Konspekt pracy zaliczeniowej

Damian Kuśmierz

Scenariusz przebiegu prezentacji
(Sekcje „tekst do wypowiedzenia” jest to kwestia którą chciałbym powiedzieć przy konkretnym slajdzie.
Nie mają one być czytane słowo w słowo, mają jedynie pomóc w przypadku „zgubienia się” podczas wypowiedzi.
Dla ułatwienia odróżnienia ich od reszty są pisane kursywą.
Tekst ten znajduje się również w notatkach pod poszczególnymi slajdami.):

1. Przedstawienie czym są oraz do czego służą funkcje systemowe, wraz z przykładowym zastosowaniem. Pokazanie różnicy pomiędzy funkcjami systemowymi a funkcjami bibliotecznymi. (slajd 3)

   Tekst do wypowiedzenia:

   a. *Podążając za definicją, możemy się dowiedzieć, że funkcje systemowe, są zaimplementowane w jądrze systemu operacyjnego a także pozwalają między innymi na zarządzanie katalogami, plikami, systemem plików lub procesami.*
   *Oczywiście pozwalają one na znacznie więcej, ale my skupimy się na pierwszych dwóch wykorzystaniach, czyli zarządzanie plikami i folderami.*

      *Mała rada, nieco łatwiej znaleźć informację o tych funkcjach szukając fraz zawierających „syscall”, „system call” bądź „wywołanie systemowe”. Przynajmniej u mnie, frazy zawierające „funkcje systemowe” zwracały bardzo dużo wyników nie będących związanymi z tematem.*

      *Można więc powiedzieć, że te funkcje, wywołania, czy jak wolicie to nazywać, pozwalają na to, co możecie zrobić w Linuxie w terminalu.*

      *Ich używanie też można w pewnym stopniu porównać do wywoływania poleceń w konsoli. Oczywiście są różnice, jedną z głównych jest składnia. Tam gdzie w terminalu polecenie i argumenty oraz same argumenty są oddzielane spacjami, lub innymi poprawnymi białymi znakami, tam w wywołaniach systemowych w C argumenty znajdują się w nawiasie i są oddzielane przecinkami.*

2. Przedstawienie kilku funkcji dostępu do plików wraz z opisem obsługi błędów w tych funkcjach. (Slajdy 4 i 5)
   Opisywane funkcje dostępu do plików: open, openat, creat, write, read, close, lseek.
   Podanie przykładów implementacji w kodzie źródłowym (Przedstawienie kodu źródłowego wykorzystującego daną funkcję oraz uruchomienie programu skompilowanego z tego kodu, przykłady znajdują się w folderze „przykłady”) np.:

   a. open, openat, creat – Otwórz plik.
      Jeśli plik nie istnieje i ustawiono flagę O_CREAT, to plik zostanie stworzony.
      Zwraca deskryptor pliku lub -1 jeśli wystąpił błąd otwarcia pliku, ustawia wtedy też zmienną errno.

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

int main(void) {
	int fdesc = open("testowyPlik.txt", O_WRONLY | O_APPEND);
	if (fdesc == -1) {
		puts("Błąd odczytu pliku");
		return 1;
	}
	puts("Pomyślnie otwarto plik");
	return 0;
}
```

   b. Tekst do wypowiedzenia:

      *(Slajd 4)*
      *Przed przystąpieniem do opisywania tych funkcji mam kilka dodatkowych informacji.*

      *Deskryptor - Zmienna zawierająca liczbowy identyfikator*

*otwartego w programie pliku, nie wiem czy w innych systemach jest to zaimplementowane inaczej.*
*Deskryptory otwarte przez użytkownika zaczynają się zwykle od 3, ponieważ deskryptory 0, 1 i 2 są otwierane automatycznie przez proces i określają odpowiednio: Standardowe wejście, Standardowe wyjście i Standardowe wyjście błędów.*

*Demony lub usługi nie zawsze mają zaimplementowane te trzy pierwsze deskryptory gdyż procesy pracujące jako demony lub usługi działają jako procesy drugo planowe, bez konieczności interakcji z użytkownikiem.*

*Errno jest to zmienna do której funkcje systemowe w przypadku niepowodzenia zapisują kod błędu. Trzeba jednak pamiętać, że nie ma pewności, że w przypadku powodzenia syscall zapisze w errno wartość 0, oznaczającą brak błędu, więc dla bezpieczeństwa polecam tak jakby zresetować wartość errno przed użyciem jakiejkolwiek funkcji systemowej, gdyż w przeciwnym wypadku możecie odczytywać błędy pozostawione w errno przez syscall, który był wywołany wcześniej.*

*Informacją, że wystąpił błąd a errno zawiera jego kod zwykle jest zwrócenie przez funkcję wartości -1 lub NULL w zależności czy funkcja zwraca typ int czy wskaźnik.*
*Mając to za sobą przejdźmy do konkretów.*

*Z góry zaznaczę, że każda prezentowana funkcja ustawia wartość zmiennej errno w przypadku niepowodzenia.*

*Jak możecie zauważyć, pierwsze wywołanie ma aż trzy formy, open, openat oraz creat.*
*Wszystkie trzy otwierają plik, ale nieznacznie się różnią.:*
*- w open podajecie jako argumenty ścieżkę do tworzonego pliku, może być to ścieżka absolutna lub relatywna oraz flagi, flagi łączymy pojedynczą pionową kreską | w taki sam sposób jak w konsoli używacie pipe czyli FLAGA1 | Flaga2.*

*Na koniec możecie dodać trzeci argument, tak zwane „mode_t", który określa uprawnienia przyznane tworzonemu plikowi.*

*- openat jest podobny, ale jako pierwszy argument przyjmuje deskryptor katalogu, lub jeśli ktoś chce się czepiać szczegółów „desktyptor pliku katalogu", od którego zaczyna się ścieżka podawana w drugim argumencie, flagi i opcjonalny mode_t to w tym przypadku odpowiednio trzeci i czwarty argument*

*- creat jest odpowiednikiem open wywoływanego z flagami „Stwórz jeśli nie istnieje" (**O_CREAT**), „Do zapisu" (**O_WRONLY**) oraz „Wyczyść plik jeśli istnieje" (**O_TRUNC**).*

*creat jako argumenty przyjmuje wyłącznie ścieżkę do pliku i mode_t*

*Jeśli wszystko przebiegło pomyślnie, a mamy nadzieję, że tak się właśnie stało, to zostanie zwrócony deskryptor pliku, który możemy użyć w kolejnych funkcjach.*

*Przechodząc dalej, natrafiamy na funkcję write, która jak sama nazwa wskazuje pozwala na zapisanie jakiegoś ciągu znaków do pliku.*

*Chciałbym tutaj przypomnieć że każdy ciąg znaków w języku C powinien być zakończony symbolem NULL oznaczonym \0. Jeśli na końcu ciągu znaków nie będzie tego symbolu to będą działy się bardzo złe rzeczy, na przykład czytanie śmieciowych danych będących poza zadeklarowanym ciąggiem, albo choćby Segmentation Fault, który można porównać do okienka „Program przestał działać" w Windowsie, z tym że występującym gdy próbujemy uzyskać dostęp do danych które są chronione w pamięci komputera takie jak na przykład dane innego programu.*

*Write jako swoje argumenty przyjmuje deskryptor pliku do którego mają zostać zapisane dane, bufor (tablicę lub wskaźnik o typie char) oraz wielkość zapisywanego buforu w bajtach.*

*Jak wszystko przebiegnie pomyślnie, to dostaniemy od tej funkcji 0.*

*Kolejną omawianą funkcją będzie read, Tutaj również nie odkryjemy ameryki mówiąc, że służy ona do odczytania zawartości pliku.*
*Jako argumenty przyjmuje ona deskryptor pliku, bufor do którego mają być wczytane dane oraz ilość bajtów jaką chcemy odczytać.*
*Jeśli ilość odczytanych bajtów jest mniejsza niż długość pliku, to można ponownie odczytać dane z pliku, od miejsca gdzie poprzednie odczytywanie się zakończyło.*
*Tutaj również jeśli wszystko przebiegnie popranie to dostaniemy 0, a jak nie, to -1.*

*Następne będzie lseek, które zwraca pozycję na której znajdujemy się w pliku, może to na przykład posłużyć do ustawienia pozycji w pliku, sprawdzenia ile danych zostało jeszcze do odczytania, albo do sprawdzenia jak duży jest cały plik, żeby móc zaalokować bufor o odpowiedniej wielkości.*
*Argumentami tej funkcji są kolejno: deskryptor pliku, offset, który nie zawsze jest przydatny, oraz nasze ulubione, flagi, które w tym przypadku pozwalają na np.: ustawienie pozycji w pliku na taką którą ustawiliśmy w drugim argumencie albo rozpoczęcie od obecnej pozycji w pliku.*

*Ostatnią funkcją obsługi plików, którą omówimy, będzie close, bo zwykle jest to ostatnia funkcja której używa się do jakiegoś działania na jakimś pliku. Jeśli spróbujecie zapisać coś do deskryptora który został zamknięty, to otwieracie wasz kod na sporo błędów, mających potencjał na dość poważne konsekwencje, więc po prostu na to uważajcie.*
*Ta funkcja jest chyba najprostszą z przedstawionych, ponieważ przyjmuje tylko jeden argument, będący deskryptorem pliku który chcemy zamknąć.*
*Jeśli uda się zamknąć plik, to zwrócona zostanie wartość 0.*

*(Slajd 5)*

*Przydałoby się chyba pokazać te funkcje w akcji, ale zanim przejdę do kodu, dam wam troszkę czasu jeśli chcecie otworzyć te strony.*

*## NASTĘPNIE POKAZUJE KOD ŹRÓDŁOWY fileAccess.c I DZIAŁANIE PROGRAMU*

3. Przedstawienie kilku funkcji zarządzania katalogami wraz z opisem obsługi błędów (Slajdy 6 i 7).
   Opisywane funkcje dostępu do plików: mkdir, mkdirat, opendir, fdopendir, closedir, readdir, chdir, fchdir, getcwd, getwd, get_current_dir_name.
   Podanie przykładów implementacji, podobnym jak ten powyżej, na przykład:
   a. mkdir, mkdirat – Utwórz katalog. W przypadku pomyślnego utworzenia katalogu, zwraca 0, w przeciwnym wypadku zwraca -1 i ustawia errno.
   b. Tekst do wypowiedzenia:

      *(Slajd 6)*
      *Tutaj również szybko przypomnę o resetowaniu wartości errno przed użyciem tych funkcji, gdyż można odczytywać śmieciowe błędy.*

      *Zaczynamy od tworzenia katalogów, mając do tego celu 2 formy polecenia mkdir i mkdirat.*
      *Działają one w dokładnie taki sam sposób poza jedną drobną różnicą,*
      *Mkdir przyjmuje 2 argumenty, ścieżkę do katalogu i dobrze nam już znane mode_t, natomiast mkdirat przyjmuje 3 argumenty.*

*Pierwszym jest deskryptor katalogu, drugim jest ścieżka do katalogu a trzeci to mode_t.*
*Różnica w działaniu tych 2 funkcji pojawia się przy użyciu relatywnych ścieżek.*

*Używając takiej ścieżki w mkdir będzie ona zinterpretowana jako ścieżka relatywna do katalogu w którym pracuje program,*
*Natomiast w mkdirat będzie ona zinterpretowana jako ścieżka relatywna do katalogu do którego deskryptor podamy w pierwszym argumencie.*
*Jeśli w mkdirat użyjemy absolutnej ścieżki, to deskryptor katalogu zostanie zignorowany.*
*Po pomyślnym utworzeniu katalogu otrzymacie 0, w przypadku błędu -1 i kod błędu w errno.*

*Przechodząc dalej widzimy dwie wersję funkcji otwarcia katalogu, różnią się one przyjmowanym argumentem, opendir przyjmuje ścieżkę do pliku, fdopendir przyjmuję deskryptor katalogu.*
*Obie zwracają wskaźnik do strumienia katalogu z którego można odczytać jego zawartość. W przypadku błędu zostanie zwrócony NULL.*

*Kolejną funkcją będzie readdir, służąca do odczytania zawartości katalogu, każde wywołanie zwraca wskaźnik o strukturze „dirent" składającej się z numeru „inode", offsetu, którego nie powinno się używać wraz ze współczesnymi systemami plików, wielkość wpisu w bajtach typ wpisu i nazwy wpisu.*
*Funkcja ta zwróci wartość NULL w dwóch przypadkach, gdy wystąpi jakiś błąd oraz gdy dotrzemy do końca katalogu, należy wtedy sprawdzić czy errno zawiera kod błędu, jeśli jest tam 0, to wszystko przebiegło dobrze.*
*Po odczytaniu strumienia katalogu można na przykład zamknąć go.*

*Do tego celu służy funkcja closedir, która jako jedyny argument przyjmuje wskaźnik do strumienia katalogu.*
*W przypadku sukcesu zostanie zwrócone 0, w przeciwnym wypadku -1 i zostanie ustawione errno.*

*Kolejną funkcję, mającą aż 3 różne wersje, zwraca wskaźnik do ścieżki katalogu w którym uruchomiliśmy program.*
*Getcwd i getwd mogą ponadto przyjąć jako argument bufor do którego mogą zapisać ścieżkę do pliku.*
*Getcwd może dodatkowo przyjąć jako 2 argument wielkość bufora w bajtach.*
*W przypadku niepowodzenia zostanie zwrócone NULL.*

*Ostatnią funkcją jakiej się przyjrzymy będzie funkcja zmiany katalogu w którym pracuje program.*
*Ma ona dwie wersje, pierwsza, chdir, przyjmuje jako argument ścieżkę do pliku, natomiast druga, fchdi,r przyjmuje deskryptor pliku.*
*Jeśli katalog pracy zostanie zmieniony pomyślnie zostanie zwrócone 0, w przypadku błędu -1.*
*Przed użyciem readdir zaleca się ustawić wartość errno na 0*

*(Slajd 7)*

*Te linki już widzieliście, więc po prostu przejdę do pokazania przykładu z przedstawionymi przed chwilą funkcjami*
*## NASTĘPNIE POKAZUJE KOD ŹRÓDŁOWY directoryAccess.c I DZIAŁANIE PROGRAMU*

4. Przedstawienia ćwiczenia do wykonania przez kolegów z zajęć (Slajd 8)
    a. Tekst do wypowiedzenia:

*Skoro przebrnęliśmy przez przedstawienie funkcji i przykładów to chyba nadszedł czas na małe ćwiczenie, które, miejmy nadzieję, nie zajmie zbyt długo czasu.*

*Jeśli macie problem z odczytaniem go, to znajduje się ono również na stronie*
*https://epat.xyz/so/pomoce/zadanie*

*dostaniecie się tam też z poziomu strony z pomocami do prezentacji*

*Dodatkowe informacje na temat funkcji macie pod linkiem widocznym na ekranie.*
*W razie problemów, śmiało pytajcie, postaram się wam jakoś pomóc.*

Źródła wykorzystane przy tworzeniu pracy:


https://en.wikipedia.org/wiki/System_call
https://en.wikipedia.org/wiki/Linux_kernel_interfaces
http://man7.org/linux/man-pages/man2/syscalls.2.html
http://man7.org/linux/man-pages/man2/open.2.html
http://codewiki.wikidot.com/c:system-calls:open
http://man7.org/linux/man-pages/man3/errno.3.html
http://man7.org/linux/man-pages/man2/write.2.html
https://www.man7.org/linux/man-pages/man2/read.2.html
https://www.man7.org/linux/man-pages/man2/close.2.html
https://www.man7.org/linux/man-pages/man2/lseek.2.html
https://www.man7.org/linux/man-pages/man2/mkdir.2.html
https://www.man7.org/linux/man-pages/man3/opendir.3.html
https://www.man7.org/linux/man-pages/man3/readdir.3.html
https://www.man7.org/linux/man-pages/man2/chdir.2.html
https://man7.org/linux/man-pages/man3/getcwd.3.html
http://codewiki.wikidot.com/c:system-calls:write
https://www.man7.org/linux/man-pages/man3/closedir.3.html