SOP LABORATORIA 0,1

SPIS TREŚCI

1. KOMENDY TERMINAL
    - cp
    - gcc
    - SYGNAŁY
    - tar
    - xargs
2. POSIX C
    - atoi()
    - chdir()
    - closedir()
    - environ
    - errno
    - exit()
    - fgets()
    - fprintf()
    - ftw()
    - getcwd()
    - getenv()
    - getopt()
    - lstat()
    - main()
    - nftw()
    - opendir()
    - perror()
    - printf()
    - putenv()
    - readdir()
    - scanf()
    - setenv()
    - stat()
    - strtol()
    - system()
3. INNE ZAGADNIENIA
    - Używanie makefile
4. PLIKI NAGŁÓWKOWE
    - <dirent.h>
    - <stdio.h>
    - <stdlib.h>
    - <sys/stat.h>
    - <unistd.h>
5. MAKRA i PRZYDATNE FUNKCJE
    - __FILE__
    - __LINE__
    - ERR
    - scan_dir() [zliczanie plików w katalogu roboczym po typach]

1. KOMENDY TERMINAL

- **gcc (kompilator)** *[gcc <opcje> <plik>]*
  - -Wall (Warnings: all) → włącza większość ostrzeżeń
  - -o <*filename*> (output) → umieszcza plik wykonywalny w *filename*
  - -Werror → zmienia wszystkie ostrzeżenia w błędy
  - -g → włączenie debuggera
  - -c → kod maszynowy

- **tar (pakowanie plików)** *[tar <opcje> <pliki>]*
  - -c (create) → tworzy nowe archiwum
  - -j (.bzip2) → rozszerzenie .bzip2
  - -z (.bzip) → rozszerzenie .gzip
  - -J (.xz) → rozszerzenie .xz
  - -f (file) <*filename*> → określa nazwę pliku archiwum tar po spacji
  - -t (list) → wyświetla zawartość archiwum
  - -x → wypakuj
  - -v → wypisuj nazwy wszystkich plików

- **cp (copy)** *[cp <source_files> <target>]*
  - -r , -R →kopiowanie rekurencyjne

- **SYGNAŁY** (w bash na labach)
  - EOF →Ctrl + d → End of file, koniec pliku
  - SIGINT →Ctrl + c → pozwala zakończyć program
  - SIGSTOP →Ctrl + z → zwiesza program
  - SIGQUIT →Ctrl + \ → kończy program, generuje zrzut pamięci
  - Zamrożenie terminalu → Ctrl+s
  - Odmrożenie terminalu → Ctrl + q

- **xargs** (wykonuje polecenia z stdin) [**dodatkowe materiały**]
  [xargs <options> <command_line {initial_arguments}>]
  - -0 → kolejne argumenty są pobierane do znaku \0 z wejścia
  - -a *file* → zamiast z stdin pobieramy dane z *file*
  - -d *delimiter* → kolejne argumenty są pobierane do znaku *delimiter* z wejścia
  - -n *max_args* → używa maksymalnie *max_args* na *command_line* (argument xargs)
  - -p → terminal pyta zawsze czy wykonać następne *command_line*

---

2. POSIX C

- **int main (int argc, char ** argv)**
  - Argumenty :
    1. *int argc* → ilość elementów podanych dalej
    2. *char** argv* → tablica (wielkości *argc*) stringów będących kolejnymi argumentami
  - Wartość zwracana : **[#include <stdlib.h>]**
    1. EXIT_SUCCESS → pomyślne wykonanie programu [<*stdlib.h*>]
    2. EXIT_FAILURE → niepomyślne wykonanie programu [<*stdlib.h*>]

- **int printf(const char* restrict format, …) [#include <stdio.h>]**
  - Działanie
    1. Funkcja wypisuje na standardowe wyjście napis ze zmiennymi
  - Argumenty :
    1. *"string z % przed zmiennymi"*
       - d, i → int (dziesiętny)
       - o → int (ósemkowy)
       - u → unsigned (dziesiętny)
       - x → unsigned (szesnastkowy)
       - f, F → double (precyzja 6)
       - e,E → naukowy (z e)
       - s → wskaźnik do char (stringi) (liczba przed s oznacza maksymalną ilość znaków jakie przyjmie scanf)
    2. Zmienne (tyle ile zaznaczonych w stringu)
  - Wartość zwracana :
    1. Jeśli się powiedzie → ilość transmitowanych bitów
    2. Jeśli błąd wyjścia → return errno (wskazuje na error)

- **int scanf(const char* format, …) [#include <stdio.h>]**
  - Działanie :
    1. Funkcja pobiera dane ze standardowego wejścia i zapisuje je w zmiennych
  - Argument :
    1. *"string z typami zmiennych (jak w **printf**)"*
    2. Adresy zmiennych (& lub nazwa tablicy)
  - Wartość zwracana
    1. Jeśli się powiedzie →liczba dopasowanych elementów
    2. Jeśli błąd →EOF (znacznik końca plików)

- **void perror (const char *s) [#include <stdio.h>]**
  - Działanie :
    1. Funkcja wypisuje błąd na standardowy strumień błędów.
       - Jeśli (s != null) to najpierw wypisuje po dwukropku s
       - Potem wypisuje wiadomość o błędzie po nowej linii
         - Zawartość wiadomości == strerror(errno)
  - Argument :
    1. Napis s → może być null

- **int fprintf (FILE* stream, const char* format , …) [#include<stdio.h>]**
  - Działa jak **printf,** ale na początku można podać strumień do jakiego wysyłamy dane
    1. Np. frpintf(stderr,"Błąd nr %d" , 5);

- **char* fgets(char* restrict s,int n, FILE* restrict stream) [#include<stdio.h>]**
  - Pobiera bajty ze strumienia *stream* i wpisuje do *s* aż do n-1 znaku lub znaku nowej linii (kończy również po napotkaniu EOF) i dostawia \0 na końcu s
  - Argumenty :
    1. Char* s → string (tablica char) – miejsce na zapisanie wczytanego tekstu
    2. Int n → liczba znaków, które zostaną wstawione do s
       - uwzględnia bajt zerowy, więc ze strumienia można wczytać maksymalnie n-1 znaków)
    3. File* stream → strumień wejściowy [np. stdin]
  - Wartość zwracana
    1. Jeśli wszystko się powiedzie → zwraca string s (const char* s)
    2. Jeśli FILE* jest na końcu (EOF) → zwraca null pointer
       - EOF indicator w strumieniu FILE
    3. W przypadku błędu w odczycie ze strumienia → zwraca null pointer
       - Error indicator w strumieniu FILE
       - Ustawia zmienną errno na odpowiednią wartość błędu

- **void exit (int status) [#include<stdlib.h>]**
  - Działanie
    1. Wypisywana jest zawartość strumieni wyjściowych i zamykane są wszelkie zasoby z jakich korzystał program (uchwyty, pamięć, gniazda)
    2. Kończy działanie programu i zwraca kod wyjścia
  - Argumenty
    1. Int status
       - Pozytywne zakończenie programu → 0 lub EXIT_SUCCESS
       - Negatywne zakończenie programu →EXIT_FAILURE lub dowolna liczba
       - Liczy się 8 najmniej znaczących bitów argumentu

- **int atoi (const char* str) [#include <stdlib.h>]**
  - Działanie :
    1. Konwertuje string na int jeśli to możliwe
    2. Jeśli konwersja niemożliwa to zwraca 0

- **long strtol(const char* str, char** end, int base) [#include <stdlib.h>]**
  - Działanie
    1. Konwertuje wartość zapisaną w łańcuchu znaków w dowolnym systemie liczbowym do postaci liczby typu całkowitego (long)
  - Argumenty :
    1. *const char* start* → łańcuch znaków, który ma zostać przekonwertowany
    2. *char ** end* → argument wyjściowy
       - Zostanie tu zapisany adres pierwszego znaku, którego nie udało się przekonwertować
       - Funkcja zignoruje znaczenie argumentu w przypadku, gdy end=NULL
    3. *int base* →System liczbowy, w którym zapisana jest wartość
       - Wartość musi być całkowita z przedziału [2,36]
       - Jeśli base==0 to funkcja wykrywa rodzaj systemu liczbowego
  - Zwracana wartość
    1. W przypadku sukcesu → zwraca przekonwertowany string
    2. Gdy wartość w *start* spowoduje przepełnienie → zwraca LONG_MAX || LONG_MIN
       - errno zostaje ustawione na ERANGE
    3. Gdy nie jest możliwa konwersja → 0

SPIS TREŚCI

- **int getopt (int argc, char* const* argv, const char *optstring) [#include<unistd.h>]** (materiały)
  - Działanie :
    1. Każde wywołanie *getopt()* obsługuje jedną opcję. Funkcja zwraca kod rozpoznanej opcji, po zakończeniu zwraca -1.
  - Argumenty
    1. *int argc* → liczba elementów do analizy (elementów tablicy *argv*)
    2. *char* const* argv* → tablica podanych argumentów do analizy
    3. *const char* optstring* → jakie opcje są obsługiwane
       - przykład: ‘’t:n:” dla opcji –t oraz –n
       - jeśli opcja wymaga argumentu, po nazwie należy umieścić dwukropek „ : ”
       - jeśli opcja ma opcjonalny argument, po nazwie należy umieścić „ :: ”
  - Zmienne :
    1. *int opterr* → jeśli (opterr != 0) getopt wyświetla *error message* na stderr gdy napotka nieznany znacznik opcji lub opcję z brakującym argumentem (domyślne zachowanie)
       → jeśli ustawi się (opterr = 0) funkcja *getopt()* nie wydrukuje żadnych komunikatów, ale nadal będzie zwracać „?” żeby wskazać błąd
    2. *int optopt* → gdy *getopt()* spotyka nieznaną opcję lub opcję z brakującym argumentem, *optopt* przechowuje znak tej opcji
    3. *int optind* → *getopt()* ustawia *optind* na indeks następnego elementu do wykonania w *argv*
       → gdy *getopt()* skończy znajdowanie opcji, można użyć tej zmiennej do określenia, gdzie zaczynają się argumenty nie będące opcjami
       → początkowa wartość *optind = 1*
    4. *char* optarg* → wskazuje argument związany z opcją, o ile opcja go wymaga
  - Zwracana wartość
    1. *opcja* → zwraca znak następnej opcji
    2. *:* → jeśli do następnej opcji brakuje argumentu
    3. *?* → jeśli znajdzie opcję niezaznaczoną w *optstring* lub brak opcji dla jakiegoś argumentu
    4. *-1* → jeśli skończą się opcje (kończy działanie)

- **extern char** environ**
  - Tablica napisów reprezentujących zmienne środowiskowe (na końcu NULL)
    1. USER → Nazwa zalogowanego użytkownika
    2. LOGNAME → Nazwa zalogowanego użytkownika
    3. HOME → Katalog domowy użytkownika
    4. LANG → Ustawienia narodowe
    5. PATH → Ścieżka do plików wykonywalnych
    6. PWD → Katalog roboczy
    7. SHELL → Ścieżka do shella
    8. TERM → Typ terminala z jakiego korzystamy
    9. PAGER → Ustawienia wyświetlania plików tekstowych
    10. EDITOR/VISUAL → Ustawienia edytowania plików tekstowych

---

SPIS TREŚCI

- **char* getenv(const char* name) [#include<stdlib.h>]**
  - Działanie :
    1. Funkcja pobiera wartość zmiennej środowiskowej
    2. *Przykład :*

       const char name = ''HOME";*
       char value;*
       value=getenv(name);
  - Argumenty :
    1. *const char* name* → nazwa zmiennej środowiskowej
  - Wartość zwracana :
    1. wartość zmiennej *name* jako napis (const char*) → jeśli *name* istnieje
    2. null pointer → jeśli *name* nie istnieje

- **int putenv (char* string) [#include<stdlib.h>]**
  - Działanie :
    1. Funkcja zmienia lub dodaje zmienną do środowiska
  - Argumenty :
    1. *char* string* → format : ''name=value" , nowa zmienna
  - Wartość zwracana :
    1. 0 →poprawne wykonane
    2. !=0 →niepoprawne wykonanie, ustawia *errno*
       - ENOMEM - brak dostępnej pamięci

- **int setenv (const char* envname, const char* envval, int overwrite) [#include<stdlib.h>]**
  - Działanie :
    1. Funkcja zmienia lub dodaje zmienną środowiskową
  - Argumenty :
    1. *const char* envname* → nazwa zmiennej do dodania lub zmiany
    2. *const char* envval* → wartość zmiennej dodawanej lub zmienianej
    3. *int overwrite* → opcja nadpisania
  - Wartość zwracana
    1. 0 → operacja zakończona pomyślnie
    2. -1 → operacja zakończona niepomyślnie, ustawia *errno*
       - ENIVAL - *envname* jest pusty albo zawiera ''="
       - ENOMEM - brak dostępnej pamięci

- **int system (const char* command) [#include<stdlib.h>]**
  - Działanie :
    1. Wykonuje komendę shell
  - Argumenty :
    1. *command* → Komenda która jest wykonana przez powłokę jako proces potomny
  - Wartość zwracana :
    1. 0 → jeśli *command* jest NULL i żadna powłoka nie jest dostępna errno = ENOENT
    2. -1 → jeśli proces potomny nie może zostać stworzony
       → jeśli status procesu potomnego nie może zostać pobrany
       - E2BIG - lista argumentów zbyt długa
       - ENOENT - interpreter poleceń nie został znaleziony
       - ENOEXEC - plik interpretera poleceń posiada niepoprawny format i nie jest plikiem wykonywalnym
       - ENOMEM - brak dostępnej pamięci
    3. ! = 0 → jeśli *command* jest NULL i istnieje dostępna powłoka
    4. wartość zwracana przez powłokę z komendy *command* w innym wypadku
  - KOMENTARZ :
    1. Uzycie funkcji system jest równoważne z uruchomieniem powłoki jako procesu dziecka – komenda jest podana. Funkcja system zwróci to co normalnie zwróciłby program, to co można w bash'u sprawdzić wywołaniem $ - czyli status zakończenia ostatniego polecenia/programu

---

- **DIR* opendir (const char* dirname) [#include <dirent.h>]**
  - Opis działania :
    1. Funkcja otwiera strumień katalogu określony przez nazwę katalogu
    2. DIR = directory stream - strumień katalogu [#include <dirent.h>]
  - Argumenty :
    1. *const char* dirname* → nazwa katalogu do otwarcia
  - Wartość zwracana :
    1. *DIR* directory* → w przypadku powodzenia
    2. *Null pointer* → w przypadku niepowodzenia, ustawia errno na :
       - EACCES - brak uprawnień
       - ELOOP - *dirname* zawiera dowiązanie cykliczne (takie, że jego rozwinięcie nadal używa tego dowiązania)
       - ENAMETOOLONG - nazwa *dirname* za długa
       - ENOENT - brak katalogu o wskazanej nazwie lub *dirname* pusty
       - ENOTDIR - podana nazwa nie jest związana z katalogiem
       - EMFILE - zbyt wiele deskryptorów jest w użyciu przez wywołujący proces
       - ENFILE - zbyt wiele plików jest aktualnie otwartych w systemie

---

- **int closedir (DIR* dirp) [#include < dirent.h>]**
  - Działanie :
    1. Zamyka strumień katalogu *dirp*
    2. DIR = directory stream - strumień katalogu [#include <dirent.h>]
  - Argumenty :
    1. *DIR* dirp* → strumień katalogu do zamknięcia
  - Wartość zwracana
    1. 0 → w przypadku powodzenia
    2. -1 → w przypadku niepowodzenia, ustawia errno :
       - EBADF - *dirp* nie wskazuje na otwarty strumień katalogu
       - EINTR - funkcja *closedir()* została przerwana przez sygnał

- **struct dirent* readdir (DIR* dirp) [#include <dirent.h>]**
  - Działanie :
    1. Funkcja zwraca wskaźnik do struktury reprezentującej plik/folder w *dirp* i ustawiająca wskaźnik na następny element katalogu *dirp*
    2. DIR = directory stream - strumień katalogu [#include <dirent.h>]
  - Argumenty :
    1. *DIR* dirp* → strumień katalogu który czytamy
  - Wartość zwracana :
    1. *dirent* a* → wskaźnik reprezentujący aktualnie wskazywany plik/folder w strumieniu *dirp*
    2. *null pointer* → po osiągnięciu końca strumienia
    3. *null pointer + errno* → przy błędach
       - EOVERFLOW - jedna z wartości w strukturze nie może być zwrócona poprawnie
       - EBADF - *dirp* nie wskazuje na otwarty strumień katalogu
       - ENOENT - obezna pozycja strumienia jest nieprawidłowa

---

- **int stat (const char* path , struct stat* buf) [#include < sys/stat.h>]**
  - Działanie :
    1. Funkcja uzyskuje informacje o pliku *path* i zapisuje je w *buf*
    2. Nie są wymagane uprawnienia do odczytu, zapisu i uruchamiania katalogu *path*
    3. Jeśli *path* jest dowiązaniem symbolicznym, *stat()* szuka w zawartości tego dowiązania i zwraca wartość jeśli plik istnieje
  - Argumenty :
    1. *const char* path* → ścieżka do katalogu
    2. *struct stat* buf* → bufor pamięci na pozyskiwane informacje
       - *dev_t* *st_dev* → numer urzadzenia zawierajacy dany plik
       - *int_t* *st_ino* → numer i-węzła
       - *mode_t* *st_mode* → 16-bitowy typ pliku
       - *nlink_t* *st_nlink* → licznik dowiązań
       - *uid_t* *st_uid* → ID właściciela pliku
       - *gid_t* *st_gid* → ID grupy
       - *dev_t* *st_rdev* → numer urządzenia związany z plikiem specjalnym
       - *off_t* *st_size* → aktualna wielkość pliku
       - *struct timespec* *st_atim* → czas dostępu do pliku
       - *struct timespec* *st_mtim* → czas modyfikacji pliku
       - *struct timespec* *st_ctim* → czas zmiany stanu pliku
  - Makra na sprawdzanie typu plików : jak w **lstat()**
  - Wartość zwracana :
    1. 0 → operacja zakończona powodzeniem
    2. -1 → operacja zakończona niepowodzeniam, ustawia errno na :
       - EACCESS → brak uprawnień do któregoś fragmentu *path*
       - EIO → błąd podczas czytania z systemu plików
       - ELOOP → za dużo poziomów dowiązań symbolicznych *path*
       - ENAMETOOLONG → nazwa *path* za długa
       - ENOENT → brak katalogu o nazwie *path* lub *path* jest pusty
       - ENOTDIR → *path* nie jest katalogiem
       - EOVERFLOW → rozmiar pliku w bajtach, liczba bloków lub numer i-węzła nie może być poprawnie zapisana w *buf*

- **int lstat (const char* path, struct stat* buf) [#include<sys/stat.h>]**
  - Działanie :
    1. Działa jak **stat()** , ale w przypadku dowiązań symbolicznych zwróci informację o dowiązaniu, nie o pliku na który wskazuje
  - Makra na sprawdzanie typu plików
    1. S_ISBLK(st_mode) → czy jest urządzeniem blokowym?
    2. S_ISCHR(st_mode) → czy jest urządzeniem znakowym?
    3. S_ISDIR(st_mode) → czy jest katalogiem ?
    4. S_ISIFO(st_mode) → czy jest FIFO?
    5. S_ISLNK(st_mode) → czy jest dowiązaniem symbolicznym? (symbolic link)
    6. S_ISREG(st_mode) → czy jest regularnym plikiem?
    7. S_ISSOCK(st_mode) → czy jest gniazdem ?
  - Stałe typu plików
    1. Maska : → S_IFMT
    2. Użycie maski : → if (st_mode & S_IFMT)
    3. Stałe :
       - S_IFBLK → urządzenie blokowe
       - S_IFCHR → urządzenie znakowe
       - S_IFDIR → katalog
       - S_IFIFO → FIFO/pipe
       - S_IFLNK → dowiązanie symbolincze (symlink)
       - S_IFREG → regularny plik
       - S_IFSOCK → gniazdo

- **int errno [#include <errno.h>]**
  - Zmienna używana przez wiele funkcji do wskazywania na błędy
  - typ int
  - **Przykłady wartości**

- **char* getcwd (char* buf, size_t size) [#include<unistd.h>]**
  - Działanie :
    1. Funkcja pozyskuje ścieżkę aktualnego roboczego katalogu
  - Argumenty :
    1. *char* buf* → w tym stringu umieszczana jest ścieżka
    2. *size_t size* → rozmiar bufora *buf*
  - Wartość zwracana :
    1. *buf* (ścieżka) → operacja zakończona powodzeniem
    2. null pointer → operacja zakończona niepowodzeniem, ustawia errno:
       - EINVAL - *size* jest 0
       - ERANGE - *size* jest większy od 0 ale mniejszy od długości ścieżki + 1
       - EACCES - brak dostępu do któregokolwiek katalogu z szukanej ścieżki
       - ENOMEM - niewystarczająca ilość pamięci do wykonania operacji

---

SPIS TREŚCI

- **int chdir (const char* path) [#include<unistd.h>]**
  - Działanie :
    1. Zmienia aktualny katalog roboczy
  - Argumenty :
    1. *const char* path* → ścieżka do nowego katalogu roboczego
  - Wartość zwracana :
    1. 0 → operacja zakończona powodzeniem
    2. -1 → operacja zakończona niepowodzeniem, ustawia errno:
       - EACCES - brak dostępu do któregokolwiek katalogu z *path*
       - ELOOP - za dużo poziomów dowiązań symbolicznych w *path*
       - ENAMETOOLONG - nazwa *path* dłuższa niż {NAME_MAX}
       - ENOENT - któryś z katalogów w *path* nie istnieje lub *path* pusty
       - ENOTDIR - część *path* odnosi się do czegoś innego niż katalog lub dowiązanie symboliczne katalogu

---

- **int ftw (const char* path,**
  int (*fn)(const char* file , const struct stat *ptr, int flag),
  int ndirs) [#include <ftw.h>]
  - Działanie :
    1. Funkcja przechodzi przez drzewo katalogów (rekurencyjnie) startując z *path* . Dla każdej znalezionej pozycji w drzewie wywołuje funkcję *fn*
    2. Najpierw sprawdza katalog, potem dopiero jego podkatalogi
    3. Przeszukiwanie trwa dopóki $ftw()$ nie przejdzie przez całe drzewo, fn nie zwróci niezerowej wartości lub nie wystąpi jakiś błąd (inny niż EACCES) odkryty przez $ftw()$
  - Argumenty :
    1. *const char* path* → ścieżka do katalogu, od którego zaczynamy przeglądać drzewo katalogów
    2. *int (*fn)* → funkcja wykonywana na każdym elemencie
       - *concst char* file* - nazwa aktualnie przeglądanego pliku
       - *const struct stat * ptr* - status aktualnie przeglądanego pliku zwrócony przez **stat()** lub **lstat()**
       - *int flag* - flaga o wartości :
         - FTW_D → dla katalogu
         - FTW_DNR → dla katalogu bez możliwości odczytu
         - FTW_F → dla plików niebędących katalogami
         - FTW_SL → dla dowiązań symbolicznych
         - FT_NS → operacja stat nie powiodła się na pozycji, która nie jest linkiem symbolicznym
    3. *int ndirs* → maksymalna ilość strumieni i **deskryptorów pliku** które mogą być otwarte przez $ftw()$, wartość z przedziału $[\ 1, \{OPEN_{MAX}\}\ ]$
  - Wartość zwracana :
    1. 0 → operacja zakończona powodzeniem, funkcja $ftw()$ przeszła przez wszystkie możliwe katalogi i wykonała fn
    2. -1 lub inna niezerowa wartość zwrócona przez fn → operacja zakończona niepowodzeniem, napotkano błąd , ustawiane jest errno :
       - EACCESS - brak dostępu do któregokolwiek składnika *path*
       - ELOOP - za dużo dowiązań symbolicznych podczas przeglądania *path*
       - ENAMETOOLONG - *path* ma za długą nazwę
       - ENOENT - *path* odwołuje się do nieistniejącego katalogu lub *path* jest puste
       - ENOTDIR - jakiś element *path* nie jest katalogiem lub dowiązaniem symbolicznym do katalogu
       - EOVERFLOW - jakieś pole *struct stat* ptr* nie może być poprawnie reprezentowane
       - EINVAL - wartość *ndirs* jest niewłaściwa

- **int nftw (const char* path,**
  int (*fn)(const char* file, const struct stat* ptr, int flag, struct FTW* f),
  int fd_limit, int flags) [#include<ftw.h>]
  - Działanie :
    1. Funkcja działa jak **ftw()** ale przyjmuje dodatkowy argument *flags*, który jest sumą logiczną przyjmowanych flag
    2. Koniecznie trzeba w programie poprzedzić włączanie bibliotek deklaracją :
       - #define _XOPEN_SOURCE 500
         - jest ona niezbędna, inaczej LINUX nie widzi *nftw()*
    3. **UWAGA !**
       - W przypadku tej funkcji można i trzeba używać zmiennych globalnych
  - Argumenty :
    1. *const char* path* → ścieżka do katalogu, od którego zaczynamy przeglądać drzewo katalogów
    2. *int (*fn)* → funkcja wykonywana na każdym elemencie
       - *const char* file* - nazwa aktualnie przeglądanego pliku
       - *const struct stat* ptr* - status aktualnie przeglądanego pliku zwrócony przez **stat()** lub **lstat()**
       - *int flag* - flaga o wartości :
         - FTW_D → dla katalogu
         - FTW_DNR → dla katalogu bez możliwości odczytu
         - FTW_F → dla plików niebędących katalogami
         - FTW_SL → dla dowiązań symbolicznych
         - FT_NS → operacja stat nie powiodła się na pozycji, która nie jest linkiem symbolicznym
       - *struct* FTW f*
         - *int base* - offset nazwy pliku (pathname) elementu w ścieżce *file*
         - *int level* - zagłębienie elementu względem korzenia *path*, którego głębokość wynosi 0
    3. *int fd_limit* → maksymalna ilość strumieni i **deskryptorów pliku** które mogą być otwarte przez $nftw()$
    4. *int flags* → suma logiczna następujących flag
       - FTW_CHDIR
         - ustawiona : funkcja wywołuje **chdir()** dla każdego katalogu zanim zacznie przeglądać jego zawartość
         - nieustawiona: *nftw()* nie zmienia katalogu roboczego
       - FTW_DEPTH
         - ustawiona: *nftw()* wykonuje przeszukiwanie najpierw wgłąb, najpierw podkatalogi i zawartość katalogu, potem sam katalog
         - nieustawiona : *nftw()* wykonuje się najpierw dla katalogu, a potem dla jego zawartości i podkatalogów
       - FTW_MOUNT
         - ustawiona : *nftw()* pozostaje w obrębie systemu plików *path*
         - nieustawiona: powinna zgłosić wszystkie napotkane pliki
       - FTW_PHYS

- ustawiona : *nftw()* nie podąża za dowiązaniami symbolicznymi
- nieustawiona : linki symboliczne są rozwijane, ale pliki nie są raportowane wielokrotnie

- Wartość zwracana :
  1. 0 → po przeszukaniu całego drzewa
  2. -1 lub inna niezerowa wartość zwrócona przez fn → operacja zakończona niepowodzeniem, napotkano błąd , ustawiane jest errno :
     - EACCESS - brak dostępu do któregokolwiek składnika *path*
     - ELOOP - za dużo dowiązań symbolicznych podczas przeglądania *path*
     - ENAMETOOLONG - *path* ma za długą nazwę
     - ENOENT - *path* odwołuje się do nieistniejącego katalogu lub *path* jest puste
     - ENOTDIR - jakiś element *path* nie jest katalogiem lub dowiązaniem symbolicznym do katalogu
     - EOVERFLOW - jakieś pole *struct stat* ptr* nie może być poprawnie reprezentowane
     - EMFILE - wszystkie **deskryptory** dostępne dla procesu są aktualnie otwarte
     - ENFILE - zbyt wiele plików jest aktualnie otwartych w systemie

3. INNE ZAGADNIENIA

- Działanie **make**

- *all : <nazwa_docelowa>*
  <nazwa_docelowa>: <składnik1> <składnik2> …
  [tab]<po znaku tabulacji instrukcja jak zebrac te skladniki>
  <składnik1> :
  [tab]<po znaku tabulacji instrukcja jak zbudować składnik 1>
  <składnik2> :
  [tab] < po znaku tabulacji instrukcja jak zbudować składnik 2>
  …
  .PHONY: clean → dzięki tej linii, jeśli mamy plik *clean* w katalogu roboczym, to nie będzie problemu po wpisaniu *make clean* – zostanie wykonana instrukcja clean
  clean :
  *[tab] rm –rf *.o* → ta linia usunie wszystkie pliki z kodem maszynowym (.o)

- myślnik (-) dodany przed instrukcją oznacza ignorowanie błędów

- *Przykład :*

```
main:  main.o  hello.o
        gcc -o main main.o hello.o
main.o: main.c hello.h
        gcc -o main.o -c main.c
hello.o: hello.c hello.h
        gcc -o hello.o -c hello.c
.PHONY: clean
clean:
        -rm -f main.o hello.o main
```

4. PLIKI NAGŁÓWKOWE

- *<stdio.h>*
  - $fgets()$
  - $fprintf()$
  - $printf()$
  - $scanf()$

- *<stdlib.h>*
  - $atoi()$
  - $exit()$
  - $EXIT\_FAILURE$
  - $EXIT\_SUCCESS$
  - $strtol()$
  - $getenv()$
  - $putenv()$
  - $setenv()$
  - $system()$

- *<unistd.h>*
  - $getopt()$
  - $getcwd()$
  - $chdir()$

- *<dirent.h>*
  - $opendir()$
  - $closedir()$
  - $readdir()$

- *<sys/stat.h>*
  - $stat()$
  - $lstat()$

- *<ftw.h>*
  - $ftw()$
  - $nftw()$

5. MAKRA I PRZYDATNE FUNKCJE

- __FILE__ → pokazuje nazwę pliku źródłowego (char*)

- __LINE__ → pokazuje linię w której zostanie wywołane (int)

- ERR

```
#define ERR (source)   ( perror ( source ) \
                       fprintf ( sdterr , ‘’%s:%d\n” , __FILE__ , __LINE__) , \
                       exit (EXIT_FAILURE) )
```

- *np. ERR(‘’Name too long!”);*

- void scan_dir()
  - funkcja zlicza pliki, linki, katalogi i inne obiekty w katalogu roboczym (bez podkatalogów)

```
void scan_dir()
 {
        DIR* dirp;
        struct dirent *dp;
        struct stat filestat;
        int dirs=0, iles=0, links=0, other=0;
        if ( NULL == ( dirp = opendir ( ‘’ . ” ) != NULL ) ) )
        do
        {
                errno = 0;
                if  ( (dp = readdir ( dirp ) != NULL)
                {
                        if ( lstat (dp -> d_name, &filestat ) )
                                ERR( ‘’ lstat ‘’ );
                        if ( S_ISDIR ( filestat.st_mode ) )
                                dirs++;
                        else if ( S_ISREG ( filestat.st_mode ) )
                                files++;
                        else if (S_ISLNK ( filestat.st_mode) )
                                links++;
                        else
                                other++;
                }
        } while ( dp != NULL )
        if ( errno != 0 )
                ERR ( ‘’ readdir ‘’ );
        if ( closedir ( drip ) )
                ERR ( ‘’ closedir ‘’ );
        printf ( ‘’ Files : %d, Dirs : %d , Links: %d, Other : %d\n ‘’, files, dirs, links, other );
 }
```