Kompilacja biblioteki statycznej i wspoldzielonej, razem z dołączeniem

```
static:
 $(CC) -c library.c
 ar rcs library.a library.o
 $(CC) $(MAIN_SRC).c -I. library.a -o $(MAIN_SRC)_static

shared:
 $(CC) -c -fPIC library.c -o library.o
 $(CC) -shared -fPIC -o library.so library.o
 $(CC) -L. library.so -o $(MAIN_SRC)_shared $(MAIN_SRC).c
```

Pomiar czasu

```
void printTime(clock_t rStartTime, struct tms tmsStartTime, clock_t rEndTime, struct tms tmsEndTime){
    printf("Real:   %.2lf s   ", timeDiff(rStartTime, rEndTime));
    printf("User:   %.2lf s   ", timeDiff(tmsStartTime.tms_utime, tmsEndTime.tms_utime));
    printf("System: %.2lf s\n", timeDiff(tmsStartTime.tms_stime, tmsEndTime.tms_stime));
}

  clock_t rTime[6] = {0, 0, 0, 0, 0, 0};
  struct tms* tmsTime[6];

  rTime[currentTime] = times(tmsTime[currentTime]);
  currentTime++;

/// calculations

  rTime[currentTime] = times(tmsTime[currentTime]);
  currentTime++;
```

Odczytywanie statystyk pliku

```
  struct stat file_stat;   // file stats struct

  stat(abs_path, &file_stat);

  sprawdzanie czy jest folderem : S_ISDIR(stat.st_mode);
```

Uzywanie ntfw

nftw(r_path, print_nftw, 10, FTW_PHYS);

r_path - sciezka do pliku
print_nftw - int print_nftw(const char *fpath, const struct stat *file_stat, int flag, struct FTW *ftw)
FTW_PHYS - perform a physical walk and shall not follow symbolic links,

Funkcje rodziny exec

int execl(char const *path, char const *arg0, ...) //funkcja jako pierwszy argument przyjmuje ścieżkę do pliku, następne są argumenty wywołania funkcji, gdzie arg0 jest nazwą programu

int execle(char const *path, char const *arg0, ..., char const * const *envp) podobnie jak execl, ale pozwala na podanie w ostatnim argumencie tablicy ze zmiennymi środowiskowymi

int execlp(char const *file, char const *arg0, ...) - również przyjmuje listę argumentów ale, nie podajemy tutaj ścieżki do pliku, lecz samą jego nazwę, zmienna środowiskowa PATH zostanie przeszukana w celu zlokalizowania pliku

int execv(char const *path, char const * const * argv) - analogicznie do execl, ale argumenty podawane są w tablicy

int execve(char const *path, char const * const *argv, char const * const *envp) - analogicznie do execle, również argumenty przekazujemy tutaj w tablicy tablic znakowych

int execvp(char const *file, char const * const *argv) - analogicznie do execlp, argumenty w tablicy

Flagi waitpid(pid, &status, flaga)

WNOHANG
WIFEXIT

makro:
WEXITSTATUS()

Sygnaly

a) Zbiory sygnalow

int sigemptyset (sigset_t* signal_set); - Inicjalizacja pustego zbioru sygnałów.
int sigfillset (sigset_t* signal_set); - Inicializacja wszytskich sygnalow
int sigdelset (sigset_t* signal_set, int sig_no); - usuniecie sygnalu ze zbioru

b) struct sigaction {

void (*sa_handler)(int);
void (*sa_sigaction)(int, siginfo_t *, void *);
sigset_t sa_mask; // blokowane sygnaly
int sa_flags; // jezeli SA_SIGINFO to definiujemy sa_sigaction i mozemy odczytac kto wyslal, wszystko jest w siginfo
void (*sa_restorer)(void); }

c) Aby ustawic blokowanie sygnalow

sigaction(int sig_no, const struct sigaction *act, struct sigaction *old_act); // old_act ustawic na NULL

signal(handler, int sig_no); - void handler();

d) wysylanie sygnalow

int raise(int signal);
Wysyła sygnał do bieżącego procesu; do tego celu w rzeczywistości wykorzystuje funkcję kill(). O ile nie zostanie przechwycenie lub zignorowanie sygnału, proces zostanie zakończony. Wywołanie raise() jest równoważne z wywołaniem kill(getpid(), sig);

int sigqueue(pid_t pid, int sig, const union sigval value)
Funkcja ta wysyła sygnał sig do procesu o danym pid. Jeśli przekazany pid jest równy 0 sygnał nie zostanie wysłany, natomiast nastąpi sprawdzenie ewentualnych błędów, które mogłyby nastąpić przy wysyłaniu. Argument sigval może zawierać dodatkową wartość wysłaną wraz z sygnałem. Typ sigval zdefiniowany jest następująco:

```
union sigval {
 int sival_int;
 void *sival_ptr;
}
```

Potoki

```
int pipe(int fd[2]);
```
read(skad, do_jakiej_zmiennej, ile);
write(dokad, z_jakiej_zmiennej, ile);

```
FILE* popen(const char* command, const char* type);
int pclose(FILE* stream);
```
Funkcja `popen` tworzy potok, nowy proces, ustawia jego wejście lub wyjście standardowe na stosowną końcówkę potoku (zależnie od wartości argumentu `type` - `"r"` oznacza, że chcemy odczytać wyjście procesu, `"w"` że chcemy pisać na jego wejście) oraz uruchamia w procesie potomnym shell (`/bin/sh`) podając mu wartość `command` jako polecenie do zinterpretowania. Jeśli operacja ta się powiedzie, `popen` zwraca obiekt `FILE*`, którego można używać z funkcjami wejścia/wyjścia biblioteki standardowej C.

Potoki nienazwane(FIFO)

```
int mkfifo(const char *pathname, mode_t mode);
```
// oczytywanie i zapisywanie dokladnie tak samo jak w przypadku zwyklych plikow, mode dziala jak w przypadku fopen()