struct sigaction:
sa_handler
sa_mask
sa_flags

sigaction(SIG, &sa, NULL) -> -1 + e
ERRNO im signal handler sichern!

SIGCHLD
SIGINT
SIGPIPE

calloc(arrsize, sizeof(type))
-> NULL + e

sigset_t mask
sigemptyset(&mask)
sigaddset(&mask, SIG)
sigprocmask(SIG_BLOCK / SIG_SETMASK, &mask, &oldmask)
sigsuspend(&mask)
--> -1 + e

fork()
fcntl(fd, F_SETFD, FD_CLOEXEC)
fcntl(fd, F_DUPFD_CLOEXEC)
execl (path, path, arg1, ...)
pid_t = waitpid(-1 / pid, &event, 0 / WNOHANG)
-> -1 + e / 0 = no zombie / pid of zombie

WEXITSTATUS(event) -> 1/0
WIFEXITED(event) -> 1/0
kill(pid, SIG)
--> -1 + e

pthread_cond_broadcast
pthread_cond_signal
pthread_cond_wait(
&cond, &mutex)
--> != 0 = err-num

pthread_mutex_lock
pthread_mutex_unlock
--> != 0 = err-num

pthread_create (pthread_t*, NULL, &worker, &arg)
pthread_detach(pthread_t)
pthread_join(pthread_t, NULL)
--> != 0 = err-num

fprintf
sprinf
fn/snprintf
--> <0 + e / bytes

fflush -> EOF + e
oder auch \n

write(fd, buf, bytes)
-> -1 + e

fputs(s, tx) -> EOF + e / bytes
fputc(c, tx) -> EOF + e
+flush(tx) -> EOF + e

fgets(buf, sizeof(buf), rx)
-> NULL + ferror + e
fgetc(rx) -> EOF + ferror + e
if(feof) / if(ferror)

fdopen(fd, mode)
fopen(path, mode) w für create
--> NULL + e

open(path, O_CREAT / O_RDWR / O_RDONLY / O_WRONLY) -> -1 + e

fclose -> EOF + e
close -> -1 + e

socket(AF_INET6, SOCK_STREAM, 0)

sockaddr_in6 addr = {
.sin6_family = AF_INET6
.sin6_addr = in6addr_any
.sin6_port = htons(port) }

bind(sock, &addr, NULL)
listen(sock, SOMAXCONN)
accept(sock, NULL, NULL)

connect(sock,
struct sockaddr * addr,
socklen_t addr_len)

dup(fd)
dup2(fd_both, fd_old)
fileno(FILE*)
--> -1 + e

DIR* opendir (path)
-> NULL + e

dirent* readdir(DIR*)
-> NULL + e vergleichen

closedir -> -1 + e

stat/lstat(path, stat*)
-> -1 + e
IS_REG(st.st_mode)
IS_DIR(st.st_mode)

unlink(path)-> -1 + e

strtok(s, "sep") /
strtok_r(... , char**)
strchr(s, c)
strrchr(s, c)
strcmp(s, s)
strncmp(s, s, n)
strcat(s, s)
strncat(s, s, n)
strcpy(s,s)

```
scandir -> -1 + e
struct dirent **namelist;
int n = scandir(".", &namelist, 0, alphasort);
if (n < 0) perror("scandir");
else {
  for (i = 0; i < n; i++) {
    printf("%s\n", namelist[i]->d_name);
    free(namelist[i]); } }
free(namelist);
```

```
do{ Berechnung } while(!CAS(...))
```

```
_Atomic <type>
_Atomic C x = ATOMIC_VAR_INIT(C value)
C atomic_load(volatile A *object)
atomic_store(volatile A *object, C desired)
bool atomic_compare_exchange_strong(
volatile A* obj, C* expected, C desired);
```