```c
// (1) scandir <dirent.h>            true => nehmen
//                                  filter   compar
struct dirent** namelist;
int n = scandir(".", &namelist, NULL, alphasort);
if(n < 0){
    perror("scandir"); //error
                       //int (*filter)(const struct dirent*)
                       //int (*compar)(const struct dirent**, const struct dirent**)
}else{
    while(n--){
        printf("%s\n", namelist[n]->d_name); //reverse order
        free(namelist[n]);
    }free(namelist);
}
// DIRECTORIES
```

```c
// (2) opendir, readdir <dirent.h><sys/types.h><sys/stat.h>
DIR* dir = opendir("."); //replace "." with your directory
if (dir == NULL) {
    die("opendir"); //error
}
struct dirent* entry;
while(errno = 0, (entry = readdir(dir)) != NULL){
    if(strcmp(entry->d_name, ".") == 0 || !strcmp(entry->d_name, "..")){
        continue; //skip . and ..
    }
    char path[strlen(".") + strlen(entry->d_name) + 2]; //works without malloc,
    sprintf(path, "%s/%s", ".", entry->d_name);
    struct stat info;
    if((stat(path, info)){ // stat() does follow symlink
        continue; //error
    }
    if(!S_ISREG(info.st_mode) || !S_ISDIR(info.st_mode)){
        continue;
    }
    printf("%s: %d bytes\n", path, info.st_size);
}
if(errno){
    die("readdir"); //error
}
if(closedir(dir)){
    die("closedir"); //error
}
// DIRECTORIES
```

```c
// (3) fopen, fgets, ferror <stdio.h>
char buf[100]; //99 + "\0"
FILE* fh = fopen("filename", "r");
if (fh == NULL){
    die("fopen"); //error
} //fread(buf, 1, 100, fh) //Binärdaten
while(fgets(buf, 100, fh) != NULL){
    printf("%s", buf);
}                              //fputs in (5)
if (ferror(fh)){
    die("fgets"); //error
}
if (fclose(fh)){
    die("fclose"); //error
}
// FILES, STREAMS
```

```c
// (4) socket, bind, listen <sys/socket.h>
int sock = socket(AF_INET6, SOCK_STREAM, 0);
if (sock == -1){
    die("socket"); //error
}
struct sockaddr_in6 sin6 = {
    .sin6_family = AF_INET6,
    .sin6_port = htons(LISTEN_PORT),
    .sin6_addr = in6addr_any,
};
if(-1 == bind(sock, (struct sockaddr*) &sin6, sizeof(sin6))){
    die("bind"); //error
}
if (-1 == listen(sock, SOMAXCONN)) {
    die("listen"); //error
}
while (1){
    int c_sock = accept(sock, NULL, NULL);
    if (c_sock == -1){
        perror("accept"); //error
        continue;
    }
    //Code (rx,tx) (5)
    close(c_sock);
}
close(sock);
// SERVER
```

```c
// (5) dup <unistd.h> //rx, tx
//     fdopen <stdio.h>
//     fgets, fputs <stdio.h>
int c_txsock = dup(c_sock); //c_sock aus (4)
if (c_txsock < 0){
    die("dup");
}
FILE* rx = fdopen(c_sock, "r");
if (!rx){
    die("fdopen"); //error
}
FILE* tx = fdopen(c_txsock, "w");
if(!tx){
    die("fdopen"); //error
}
if(fputs(tx, "Anfrage\n") < 0){ //senden
    die("fputs");
}
//empfangen: fgets (3) mit rx

if(fclose(rx)){
    die("fclose"); //error
}
if(fclose(tx)){
    die("fclose"); //error
}
// SEND, RECEIVE SERVER
```

```c
// (6) socket, connect <sys/socket.h><sys/types.h>
struct addrinfo hints = {
    .ai_socktype = SOCK_STREAM,
    .ai_family = AF_UNSPEC,
    .ai_flags = AI_ADDRCONFIG,
};
struct addrinfo* head;
int sock;
struct addrinfo* curr;
int error = getaddrinfo("server", "port", &hints, &head);
    if(error == EAI_SYSTEM){
        die("getaddrinfo"); //error
    }
    if(error){
        fprintf(stderr, "%s\n", gai_strerror(error));
        exit(EXIT_FAILURE); //error
    }
for(curr = head; curr != NULL; curr = curr->ai_next){ //continue evtl. error: -1
    sock = socket(curr->ai_family, curr->ai_socktype, curr->ai_protocol);
    if(!connect(sock, curr->ai_addr, curr->ai_addrlen)){
        break;
    }
    close(sock) //evtl. erron: -1
}                 //<- lesen/schreiben: (5) / fprintf + fflush
if (curr == NULL){
    die("connect"); //error
}
freeaddrinfo(head);
// CLIENT
```

```c
// (7) pipe <unistd.h> fork, write, read
//     fd[1] -> fd[0]
int fd[2];
char* tosend = "Hallo Welt";
if(pipe(fd)){
    die("pipe"); //error
}
pid_t pid = fork();
if (pid == 0){ //child
    close(fd[0]); //Ausgang nicht benötigt
    write(fd[1], tosend, strlen(tosend)+1);
    exit(EXIT_FAILURE);
}
if (pid == -1) {
    die("fork"); //error
}
//parent
close(fd[1]); //Eingang nicht benötigt
char buf[100] = {0};
read(fd[0], buf, 99 * sizeof(char));
printf("got %s", buf);
// fork: parent -> Child PID
//       Child -> 0 ; getppid()
// PIPES
```

```c
// (8) sigaction, sigemptyset, sigaddset, ... <signal.h>
static void wolfgang(int sig){
    int old = errno;
    int status;
    while(waitpid(-1, &status, WNOHANG) > 0){
        if(WIFEXITED(status)){
            //child terminated normally
        } //collect dead Children
    }
    errno = old;
}
// ..............................................
struct sigaction sa = { //assign Händler to signal
    .sa_handler = wolfgang,
    .sa_flags = SA_RESTART | SA_NOCLDWAIT,
}
if(sigaction(SIGCHLD, &sa, NULL) == -1){
    die("sigaction"); //error
}

sigset_t new, old;
sigemptyset(&new);
sigaddset(&new, SIGCHLD);
sigprocmask(SIG_BLOCK, &new, &old); //Block Signals
    //critical section
    while(waitcond){
        sigsuspend(&old); //Wait unil Signal
    }
sigprocmask(SIG_SETMASK, &old, NULL); //Unblock Signals
//other code
sigprocmask(SIG_BLOCK, &new, &old); //Block Signals
    waitcond++ //or similar
sigprocmask(SIG_SETMASK, &old, NULL); //Unblock Signals

// NOTE! NO BLOCKING STUFF IN SIGNAL-HÄNDLERS. D.H. KEIN PRINTF ETC.
// ..............................................
//Collect zombies with DEFAULT HANDLER / IGNORE SIGNALS
struct sigaction action = {
    .sa_handler = SIG_DFL, //SIG_IGN to ignore
    .sa_flags = SA_NOCLDWAIT | SA_RESTART, //To ignore signal, leave empty
};
sigaction(SIGCHLD, &action, NULL); //error handling above
// SIGNALS
```

```c
// (9) pthread <pthread.h>
for(int i = 0; i < threadcount; i++){ // void* run(void* args){}
    pthread_t thread;
    if (errno = pthread_create(&thread, NULL, run, &runarg)){ //runarg can be NULL
        die("pthread_create"); //error
    }
    pthread_detach(thread);
} //alternatively:
// threads[i] = thread;
// for (int i=0; i < threadcount; i++){
//     pthread_join(threads[i], NULL);
// }
// PTHREADS
//EXIT => pthread_exit(NULL);
```

```c
// (10) SEMAPHORE - SEM / SEM CREATE     malloc <stdlib.h>
typedef struct {
    volatile int count;
    pthread_mutex_t mutex;
    pthread_cond_t condition;
}SEM;
// ..............................................
SEM* semCreate(int val){
    SEM* new = malloc(sizeof(SEM));
    if (new == NULL){
        return NULL; //error
    }
    new->count = val;
    if((errno = pthread_mutex_init(&new->mutex, NULL))){
        free(new);
        return NULL; //error
    }
    if((errno = pthread_cond_init(&new->condition, NULL))){
        pthread_mutex_destroy(&new->mutex);
        free(new);
        return NULL; //error
    }
    return new;
}
// SEMAPHOR CREATE
```

⑩ II SEMAPHORE P() V() semDestroy()

P() - Lock

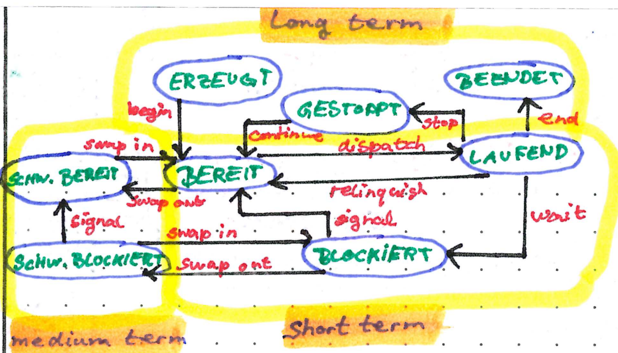
```
void P(SEM* sem) {
    pthread_mutex_lock(&sem->mutex)
    while(sem->count < 1){
        pthread_cond_wait(&sem->condition, &sem->mutex);
        //ähnlich sigsuspend => kein lost wakeup
    }
    sem->count--;
    ptread_mutex_unlock(&sem->mutex);
}
```

V() - UnLock

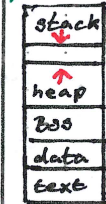
```
void V(SEM* sem){
    pthread_mutex_lock(&sem->mutex);
    sem->count++;
    if(sem->count==1){
        pthread_cond_broadcast(&sem->condition);
    }
    pthread_mutex_unlock(&sem->mutex);
}
```

SEM-DESTROY

```
void semDestroy(SEM* sem){
    if(sem==NULL){return;} //error
    errno=pthread_cond_destroy(&sem->condition);
    if(errno){return;} //error
    errno=pthread_mutex_destroy(&sem->mutex);
    if(errno){return;} //error
    free(sem);
}
```



Adressraum



Einplanungsverfahren

- FCFS | Verarbeitung nach Ankunftszeit
- RR | regelmäßige umplanung
- VRR | RR mit variablen Zeitscheiben nach Vorzugsqueue
- SPN | Einplanung nach erwarteter Bedienzeit
- HRRN | SPN mit höhere Priorität für ältere Prozesse
- SRTF | SPN mit spontanen Umplanungen
- MLQ | Mischbetrieb, verschiedene Strategien für Prozes
- FB | regelmäßige Umplanung mit Prioritätsebenen und größeren Zeitscheiben

⑪ BNDBUF / CAS

```
typedef struct {
    int* values;
    size_t size;
    SEM* frei;
    SEM* belegt;
    volatile int first, last;
} BNDBUF;
```

bbPut()

```
void bbPut(BNDBUF* bb, int val){
    P(bb->frei);
    bb->values[bb->last]=val;
    bb->last=(bb->last+1)%bb->size;
    V(bb->belegt)
}
```

bbGet()

```
int bbGet(BNDBUF* bb) {
    int curr, next, val;
    P(bb->belegt);
    do{
        curr=bb->first;
        val=bb->values[curr];
        if(curr < bb->size-1){
            next=curr+1;
        }else{
            next=0;
        }
    } while(__sync_bool_compare_and_swap(
                    &bb->first,curr,next)==0);
    V(bb->frei);
    return val;
}
```

	FCFS	RR	VRR	SPN	HRRN	SRTF	FB
kooperativ	✓			(✓)	(✓)		
verdrängend		✓	✓			✓	✓
probabilistisch				✓	✓		
deterministisch	–	–	–	–	–	–	–
Konvoi	×	×					
Verhungern	×			×			

Speicherverwaltung

- **best fit**
  - kleinstes, passendes Loch
  - Wenig Verschnitt
  - Langsam (Suchaufwand)
- **worst fit**
  - größtes, passendes Loch
  - konstanter Suchaufwand
  - hinterlässt große Löcher
- **first fit**
  - erstes, passendes Loch
  - schnell
  - kleine Löcher vorne, große hinten
  - Verschwendung, steigender Suchaufwand
- **next fit**
  - first fit ab letztem gefüllten Loch
  - eher gleich große Löcher
  - im Mittel abnehmender Suchaufwand
- **Buddy**
  - halbierung

2^n - Zweierpotenzen

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576
21	2097152
22	4194304
23	8388608
24	16777216
25	33554432
26	67108864
27	134217728
28	268435456
29	536870912
30	1073741824

- RAID 0: Speicher ~~spiegeln~~ aufteilen
- RAID 1: Speicher spiegeln
- RAID 4: ab 3 Platten: Platte mit XOR-Parität
- RAID 5: Wie RAID 4 mit verteiltem Paritätsblock
- RAID 6: RAID 5 mit 2 Paritätsblöcken
  - ↳ Ausfall von 2 Platten möglich

Verdrängungsstrategien

MFU, LFU: unsinnig

LRU
- clock: aging Schieberegister
- second chance: PRES-BIT
- third chance: PRES-BIT + DIRTY-BIT

} NACH TLB-Kopie

Freiseitenpuffer vor Auslagerung

Planungsstrategie:

kooperativ ⟷ präemptiv
deterministisch ⟷ probabilistisch
offline ⟷ online

- kibi $2^{10} = 1024^1$
- mebi $2^{20} = 1024^2$
- gebi $2^{30} = 1024^3$
- tebi $2^{40} = 1024^4$