① scandir <dirent.h>

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

filter, compare

```
// int (*filter)(const struct dirent *)
// int (*compar)(const struct dirent**, const struct dirent**)
```

DIRECTORIES

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,".") && strcmp(entry->d_name,"..")))
    continue;
  char path[strlen(".")+strlen(entry->d_name)+2];
  sprintf(path, "%s/%s", ".", entry->d_name);
  struct stat info;
  if(lstat(path, &info)){ // lstat() does follow symlink
    continue;
  }
  if(!(S_ISREG(info.st_mode) && S_ISDIR(info.st_mode))){
    continue;
  }
  printf("%s: %ld bytes\n", path, info.st_size);
}
if(errno) die("readdir");
if(closedir(dir)) die("closedir");
```

DIRECTORIES

③ fopen, fgets, ferror <stdio.h>

```
char buf[100];
FILE *fh = fopen("filename", "r");
if(fh==NULL) die("fopen");
while(fgets(buf,100,fh)!=NULL){...}
if(ferror(fh)) die("fgets");
if(fclose(fh)) die("fclose");
```

FILE STREAM

④ socket, bind, listen, accept <sys/socket.h>

```
int sock = socket(AF_INET6, SOCK_STREAM, 0);
if(sock==-1) die("socket");
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");
if(-1==listen(sock, SOMAXCONN)) die("listen");
while(1){
  int c_sock = accept(sock, NULL, NULL);
  if(c_sock==-1) perror("accept"); continue;
  // Code (rx, tx) ⑤
  close(c_sock);}
close(sock);
```

SERVER

⑤ dup <unistd.h> fdopen <stdio.h> fgets fputs <stdio.h>

```
int c_txsock = dup(c_sock);
if(c_txsock<0) die("dup");
FILE *rx = fdopen(c_sock, "r");
FILE *tx = fdopen(c_txsock, "w");
if(!tx || !rx) die("fopen");
if(fputs(tx, "Antwort\n")<0) die("fputs");
// empfangen: fgets ③ mit rx
```

```
if(fclose(rx)){
  die("fclose");
}
if(fclose(tx)){
  die("fclose");
}
```

END RECEIVE SERVER

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, *curr;
int sock;
int error = getaddrinfo("server", "port", &hints, &head);
if(error == EAI_SYSTEM) die("getaddrinfo");
if(error) fprintf(stderr, "%s\n", gai_strerror(error)); exit(EXIT_FAILURE);
for(curr=head; curr!=NULL; curr=curr->ai_next){
  sock = socket(curr->ai_family, curr->ai_socktype, curr->ai_protocol);
  if(!connect(sock, curr->ai_addr, curr->ai_addrlen)) break;
  close(sock); // Verb. Error-Handling: -1
}
if(curr==NULL) die("connect");
freeaddrinfo(head);
```

Lesen/Schreiben: ⑤ / fprintf + fflush

CLIENT

SIGNALS ⑧ sigaction, sigemptyset, sigaddset, ... <signal.h>

```
static void handler(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 = {
  .sa_handler = handler | SIG_DFL | SIG_IGN,
  .sa_flags = SA_RESTART | SA_NOCLDWAIT,
};
if(sigaction(SIGCHLD, &sa, NULL)==-1) die("sigaction")
sigset_t new, old;
sigemptyset(&new); o.sigfillset(&new);
sigaddset(&new, SIGCHLD); o.sigdelset(&new, SIGCHLD);
sigprocmask(SIG_BLOCK, &new, &old); // Block signals
  // critical section
  while(wait cond) sigsuspend(&old);
sigprocmask(SIG_SETMASK, &old, NULL); // Unblock signals
// other code...
```

NOTE! No blocking stuff in signal-handler e.g.: printf etc.

Warten mit: if(-1==sigsuspend(&new))

⑦ pthread <pthread.h>

```
for(int i=0; i<threadcount; i++){ // void *run(void *args){...}
  pthread_t thread;
  if(errno = pthread_create(&thread, NULL, run, &numarg)){ // numarg kann NULL sein
    die("pthread_create");}}
// pthread_detach(thread) o. pthread_join(thread, NULL) danach
// EXIT => pthread_exit(NULL)
```

PTHREADS

⑨ SEMAPHORE

```
typedef struct{
  int value;
  pthread_mutex_t mutex;
  pthread_cond_t condition;
} SEM;
```

```
SEM *semCreate(int val){
  SEM *new = malloc(sizeof(SEM));
  if(new==NULL) return NULL;
  if(errno = pthread_mutex_init(&new->mutex, NULL)){
    free(new);
    return NULL;
  }
  if(errno = pthread_cond_init(&new->condition, NULL)){
    free(new);
    pthread_mutex_destroy(&new->mutex);
    return NULL;}
  new->value = val;
  return new;
}
```

```
void P(SEM *sem){
  pthread_mutex_lock(&sem->mutex);
  while(sem->value <= 0) pthread_cond_wait(&sem->condition, &sem->mutex);
  sem->value--;
  pthread_mutex_unlock(&sem->mutex);
}
void V(SEM *sem){
  pthread_mutex_lock(&sem->mutex);
  sem->value++;
  pthread_cond_broadcast(&sem->condition);
  pthread_mutex_unlock(&sem->mutex);
}
```

```
void semDestroy(SEM *sem){
  errno = pthread_mutex_destroy(&sem->mutex);
  errno = pthread_cond_destroy(&sem->condition);
  free(sem);
} // Fehler behandlung nicht wirklich pos.
```

SEMAPHORE

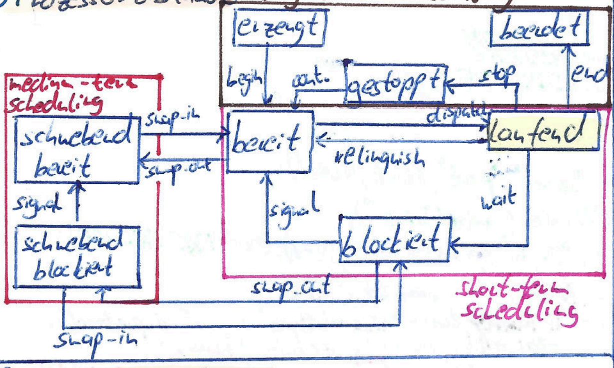
⑩ BNDBUF / CAS

```
typedef struct{
  int *values;
  SEM *writeSem;
  SEM *readSem;
  size_t size;
  size_t w;
  _Atomic size_t r;
} BNDBUF;
```

```
int bbget(BNDBUF *bb){
  if(!bb) return 0;
  P(bb->readSem);
  int value = 0;
  size_t old_r = bb->r;
  size_t new_r;
  do{
    value = bb->values[old_r];
    new_r = (old_r+1) % bb->size;
  } while(!CAS(&bb->r, &old_r, new_r));
  V(bb->writeSem);
  return value;
```

BNDBUF/CAS

① Prozesszustände



② Adressraum + Einplanungsverfahren

stack
*
heap
BSS
data
text
ELF-H.

FCFS	Prozesse mit {langen / kurzen} bursts werden {begünstigt / benachteiligt}
RR	FCFS mit Zeitscheiben => Umplanung
VRR	RR mit variablen Zeitscheiben => Bevorzugung ↑ nach E/A-burst
SPN	kürzere Prozesse zuerst => nur abschätzend mgl. (Mittelwert)
HRRN	SPN mit priorität für ältere Prozesse => hungerfreies SPN
SRTF	Verdrängendes SPN => spontane Umplanung
MLQ	Mischbetrieb aus versch. Verfahren
MLFQ	kurze Prozess werden begünstigt durch Bestrafung und Alterung von langen Prozessen => Zeit muss nicht bekannt sein wie bei SPN

Einordnung:

	FCFS	RR	VRR	SPN	HRRN	SRTF
kooperativ	✓			(✓)	(✓)	
verdrängend		✓	✓			✓
probabilistisch				✓	✓	✓
deterministisch	–	–	–	–	–	–

④ Fragmentierung

- intern: zu viel Speicher (false pos.) für Block
- extern: zu viele kleine Blöcke => genug Speicher, aber fragmentiert

③ Speicherverwaltung

best fit	- kleinstes, passendes Loch - wenig Verschnitt - langsam (Suchaufwand)
worst fit	- größtes, passendes Loch - hoher Verschnitt - schnell; konst. Suchaufwand
first fit	- erstes, passendes Loch - schnell - kleine Löcher vorne, große hinten - Verschiebung: steigender Suchaufwand
next fit	- first fit ab letztem gef. Loch - eher gleich große Löcher - im Mittel abnehmender Suchaufwand
buddy	Halbierung in Zweierpotenzen

⑤ Verdrängungsstrategien

- Pager des BS: 1. Referenzierung 2. Seitenfehler 3. Einlagerung 3. Verbuchung 5. Wiederanlauf

lok. Verfahren:
- Freiseitenpuffer
- Arbeitsmenge (komb. mit lok. LRU)

globale Verfahren:

FIFO	zuerst eingelagerte Seite wird ersetzt
LFU/MFU	seltenste/meist referenzierte wird ersetzt
LRU (bestes)	am längsten nicht mehr ref. Seite wird ersetzt - aging Schieberegister (clock) - second chance: PRES-BIT - third chance: PRES-BIT + DIRTY-BIT

=> Problem thrashing

⑩ Echtzeitbetriebe

Arten:
- weich
- hart
- fest

⑪ Gewichtsklassen (in Nebenläufigkeit)

- feather: in selber Ad.raum
- light: in Kernel, selber Addr.raum, Teilend
- heavy: in Kernel, eig. Ad.raum

⑥ Dateisysteme (mehrere Platten)

RAID 0	Speicherung über mehrere Platten
RAID 1	Daten auf zwei Platten gesp. gesp.
RAID 4	RAID 0 + Paritätsplatte
RAID 5	Paritätsblöcke über alle Platten vert.
RAID 6	RAID 5 mit zwei Paritätsblöcken

⑦ Arten Dateisysteme

FAT	- Ein Block enthält Verkettung der Daten + mehrfache Speicherung der FAT mgl. - Verkettung führt zu langen Verz. (Suche)
Inode	- Indizierung des Speichers + Große Dateien durch mehrstufige Indizierung adressierbar - mehrere Blöcke müssen für Ind. gel. werden
NTFS	- File reference mit Sequenznummer + Dateinummer -> Rückgrat ist glob. MFT + Journaling-File-System changes werden prot. (Redo/Undo) - Nachteile: etwas ineffizienter

⑧ Ausnahmen

Modelle: Wiederaufnahme (z.B. Seitenfehler), Termination (z.B. ung. Befehl)

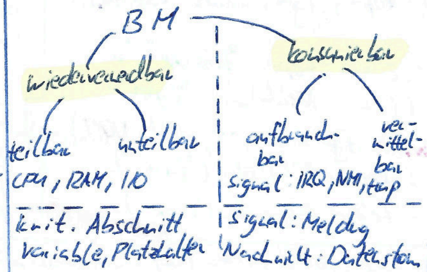
Arten:

Trap	Abfangen für Ausnahmen von internen Ursachen (synchron, vorhersehbar)
Interrupt	Unterbrechung durch Ausnahmen von ext. Ursachen (asynchron, unvorhersehbar)

Sicherungsarten:
- CPU: totale o. partielle Zustandssicherung
  - minimal SR + PC saved
  - maximal alle Register gespeichert
- BS: partielle Zustandssicherung
  - alle {dann noch unges. / flüchtigen / im weit. Verlauf verw.} CPU-Register

=> umgesetzt durch Mantelprozedur

⑨ Betriebsmittel



⑪ Gütemerkmale bei Prozesseinplanung

benutzerorientierte Kriterien:

Antwortzeit	Durchlaufzeit
Termineinhaltung	Vorhersagbarkeit

systemorientierte Kriterien:

Durchsatz	Prozessorauslastung
Gerechtigkeit	Dringlichkeit
Lastausgleich	

Betriebsarten:
- allg.: Gerechtigkeit, Lastausgleich
- Stapelbetrieb: Durchsatz, Durchlaufzeit, Prozessorauslastung
- Dialogbetrieb: Antwortzeit
- Echtzeitbetrieb: Dringlichkeit, Termineinhaltung, Vorhersagbarkeit

⑫ Nichtsequentialität

Ansätze: optimistisch, pessimistisch

Einplanung: off-line <=> on-line
- off-line: statisch, -> impl. Sync.
- on-line: dynamisch, -> expl. Sync., => cause of race-cond.

Verfahrensweisen:

unterdrückend	neue Proz. werden verhindert
blockierend	neuer Proz. wird suspendiert
nicht-block.	neuer Proz. wird zurückgerollt (CAS)

Monitore:

Hansen	- block. Bedingungsvariable -> SG verlässt Monitor <=> alle Neben. bereit
Hoare	- block. BV. -> SG verlässt Monitor <=> 1 Neben. fährt fort (atomar)
Mesa	- nicht block. BV -> SG verlässt Monitor <=> 1 oder alle Neben. auf "bereit"

⑬ Adressbildung

TLB: Cache für dereferenzierte virtuelle Adressen
-> lookup miss führt zu Tabellenwanderung (hardware- oder software-geführt)

⑭ Verklemmung

Verklemmung bez. man einen Zustand, in der bet. Proz. wechselseitig auf den Eintritt von Bed. warten, die nur durch andere Prozesse in dieser Gruppe hergestellt werden können

Notwendige Bed.:
- Ausschließlichkeit
- Nachforderung
- Unentziehbarkeit

Hinreichende Bed.:
- zirkuläres Warten