Swapping

schwebend bereit / schwebend blockiert / bereit / laufend / blockiert / swap in / swap out / Signal / wait / mittel / kurz / nicht mehr als im Speicher passen muss

stopped: kill, überlastet, Verklemmung, vermeidung
→ können bereit, laufende, blockiert, Prozesse

OS-Data
Stack
↓
↑
heap
bss
init. data
text
Eifheader

text: Code
bss: nicht initialisierte globale + static Variablen (alle 0)
data: initialisierte globale & static Variablen
heap: dynamische Erweiterung von bss (malloc)
stack: lokale Variablen, Funktionsparameter
Parameterübergabe: Call by value (Kopie wird übergeben, keine Auswirkungen auf tats. Param beim Aufrufer)
Zeiger: Call by Value übergeben an bilden
call by reference nach → veränderbar

Prozesskontrollblock: Eigentümer, Gruppenzugehörigkeit, Typ, Rechte, Zeitstempel (Zugriff, Änderung), Anzahl der Verweise, Größe in Bytes, Adressen der Daten auf dem Speichermedium. Nicht Name! Nicht Grub Größe!

PAGE-FAULT: MMU löst Trap aus → BS startet Handler
→ sucht freien Arbeitsspeicher → von Platte holen
→ Prozess auf blockiert setzen → Platte signalisiert Interrupt ist fertig → nach Einlagerung Prozess bereit
→ Prozess wird bereit → wiederholen d. gleichen Operation

Unterbrechungen: ... interne Ursache
Interrupt (asynchron, unvorhersehbar, unreproduzierbar) externe Ursache (DMA, I/O)

Speicher/Sortieralgorithmen
best-fit: zunehmende Lochgröße, von vorne kleinstes passendes Loch suchen
worst-fit: abnehmende Lochgröße, von vorne größtes passendes Loch suchen
2-hand... best-fit ... Verschnitt ...
first-fit: aufsteigende Adressen, von vorne
next-fit: wie first fit, aber ab letzter Fundstelle
→ mehr Verschnitt, Verschmelzung großer Löcher, anwendig

Prozessplanung
kooperativ: Prozesse wird CPU nicht entzogen, abh. Prozesse, CPU Abgabe durch Systemaufruf, CPU monopolisierung mgl.
präemptiv: Prozess kann CPU entzogen werden, unabh. Prozesse
Echtzeit: ...
interaktiv: ...
Stapelbetrieb: ...

	FCFS	RR	VRR	SPN (HRRN)	SRTF
kooperativ	x			x	
präemptiv		x	x		x
probabilistisch				x	x
deterministisch	x	x	x		

FCFS: Einplanung nach Ankunftszeit, lange Rechenstöße begünstigt, Konvoieffekt
RR: Zeitscheiben, ...
VRR: ...
SPN (shortest process next): Einplanung nach kürzester ...
HRRN: ...
SRTF (shortest remaining time first): ...
MLQ: ...
MLFQ: ...

Kriterien zur Auswertung einer Einplanungsstrategie:
Antwortzeit, Durchlaufzeit, Termineinhaltung, Vorhersagbarkeit, Durchsatz, Prozessorauslastung, Gerechtigkeit, Dringlichkeit, Lastausgleich

Kontrollfluss zur Fadeneinplanung (Lastfall)

Arten von Synchronisation:
einseitige Synchronisation ...
mehrseitige Synchronisation ...

Lösungen zur Lebendigkeit nichtkooperativer Programme

Mesa	Hoare	Mesa
...	...	...

---

signal / wait / Hoare / MESA signal, continue

Hoare: Ein Signalnehmer muss ... betreten
Mesa: Monitor ... signalisierender Prozess
Wait/broadcast: Hansen
Monitorwarteschlange ...
Signalgeber fortsetzen ...
signal: zeigt Ereignis an, blockiert ...
wait: setzt Prozess bis E aus, gibt Monitor frei

Verklemmungen: Prozesse warten wechselseitig auf das Eintreten von Bedingungen, die nur durch andere Prozesse dieser Gruppe selbst herbeigeführt werden können → Beteiligte bleiben blockiert
Livelock: ...
Deadlock: ...
Voraussetzungen: nicht & nicht-blockierende sync
1. exklusive Belegung von Betriebsmitteln (mutual exclusion)
2. Nachforderung von Betriebsmitteln (hold and wait)
3. unentziehbarkeit der zugeteilten Betriebsmittel (no preemption)
4. zirkuläres Warten (circular wait)
Vorbeugung: indirekt / direkt
Vermeidung: analytische Maßnahme zur Laufzeit

Adressräume
realer Adressraum: der durch einen Prozessor definierte Adressraum ...
logischer Adressraum: ...
virtueller Adressraum: ...

Paging: Unterteilung d. Adressraums in gleichgroße Einheiten und deren lineare Aufteilung (page → A, Av; page-frame | Kachel → Ar)
Seitendeskriptor: Seitenrahmennummer, Schreibschutzbit, Präsenzbit, Privilegstufe, Seite (rahmen) größe, Zugriffsrechte, Addr. der Seite im Hintergrundspeicher, Informationen ...

Segmentierung: Unterteilung des Adressraums in Einheiten von möglicherweise verschiedener Größe ...
(S|A): S: Segmentnummer, A: Adresse

Segmentdeskriptor: Basis ..., Limit ..., Attribute ...

TLB: Cache der MMU für das Übersetzungsergebnis, d. eine Ober/Untermenge von Deskriptoren

Speicherzuteilung / Platzierungsstrategie:
Bitmap ...
Halde/Liste ...
Buddy-Verfahren ...
interne Fragmentierung ...
externe Fragmentierung ...
Verschmelzung ...

Einlagerungsstrategie (on demand / anticipatory paging)

Abbildung / Seitenfehler

Ersetzungsstrategie:
MIN/OPT: ...
FIFO: ...
LRU: ...
LFU: ...
Second Chance (Clock) ...

---

gültig → 1 Bit wird bei Schreibzugriff gesetzt ...
Seitenflattern, Thrashing = kritisches Systemverhalten ...
Freiseitenpuffer (page buffering) ...

Arbeitsmenge (Working Set): ...

Speichern v. Dateien
kontinuierliche Speicherung ...
verkettete Speicherung ...
FAT ...
indizierte Speicherung ...
Inodes ...

RAID
Raid 0: ...
Raid 1: ...
Raid 4/5: ...

DATEISYSTEM
Verzeichnis / Inode / Datei
Symbol Name
Verzeichnis / Inode / Daten
Namensverzeichnis ...
Hardlink / Symlink ...

BETRIEBSARTEN
1. Stapelverarbeitung ...
2. Mehrprogrammbetrieb ...
3. Dialogbetrieb ...
4. Echtzeitbetrieb ...

II Abgesetzte Ein-/Ausgabe ...
III Adressraumschutz ...
IV Überlagerungstechnik ...

Zeitverzug an Lost, Terminverletzung ist tolerierbar
fest: Ergebnis ist wertlos, wird verworfen
hart: ...
Behandlung: fest → BS, hart → Anwendungsprogramm

Dualmodus / Einplanungsebenen
1) langfristig (s-min): Lastkontrolle ...
2) mittelfristig: Planung ...
3) kurzfristig: Planung (µs-ms) ...

Gebietsklassen von Prozessen ...

Einplanung = Vorgang d. Reihenfolgebildung von Aufträgen
Einlastung = Moment d. Zuteilung von Betriebsmitteln

Hard Link vs Symbolic Link:
Hard Link: Paar aus Dateiname + Dateikopfnummer
Symbolic Link: ...

Hardlink ↛ Datei weg, Symbolic Link ↛ Datei weg

Signale und Forkexec:
fork: Kindprozess erbt Signalbehandlung und Masken
exec: Signalmaske wird beibehalten; Signalbehandlung für SIG_DFL und SIG_IGN bleiben, benutzerdefinierte Behandlung wird auf SIG_DFL zurückgesetzt

ABA-Problem (wenn CAS)
... full / free
Anfangszustand
Thread 1: bbGet(), Value = 4, ...
Thread 2: bbGet() = 4
Thread 2: bbPut(9)
Thread 2: bbGet() = 2
Jetzt T1 wieder aufgeweckt, sieht dass der gleiche Position ist und gibt 2 zurück, obwohl die jetzt ...

Lösung: separate Zähl Variable (Double word CAS) ...

Betriebsmittel: teilbar (präemptiv) / unteilbar; wiederverwendbar / konsumierbar ...

Kern: ...
Signal (Meldung) / Nachricht (Datum)

Einseitige / Mehrseitige Synchronisation ...
Statische / dynamische Bindung ...

Verzeichnis: ...
Prozess / Threads ...
Signale & Threads ...

Offene Dateien & fork/exec
werden bei fork vererbt & bleiben auch bei exec offen → close on exec flag

Ressourcen: ...

Inode-Aufgabe
...
Folder / file
Wenn dahinter kein Pfeil bei Inhalt → ??!

Dass noch Pfeil immer in Datei, das vor Pfeil im Verzeichnis

Namensraum: ...

Buddy-Aufgabe: in Tabelle stehen, links Größe, in Tabelle ...

log. Adressraum: Schutz ...

```
struct addrinfo hints = {
  .ai_socktype = SOCK_STREAM,
  .ai_family   = AF_UNSPEC,
  .ai_flags    = AI_ADDRCONFIG,
};
struct addrinfo *head;
int addrinfo = getaddrinfo(hostpart, &hints, &head);
if (addrinfo != 0) {
  if (err == EAI_SYSTEM) { perror "getaddrinfo"; }
  else fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(addrinfo));
}

int sock;
for (curr = head; curr != NULL; curr = curr->ai_next) {
  sock = socket(curr->ai_family, curr->ai_socktype, curr->ai_protocol);
  if (sock == -1) continue;
  if (connect(sock, curr->ai_addr, curr->ai_addrlen) == 0) break;
  close(sock);
}
if (curr == NULL) { fprintf(stderr, "no socket found"); exit(EXIT_F); }
```

free!

KOMMUNIKATION:

```
FILE *rx = fdopen(sock, "r");
if (rx == NULL) fail("fdopen");
int sockdup = dup(sock);
if (sockdup == -1) fail("dup");
FILE *tx = fdopen(sockdup, "a");
if (tx == NULL) fail("fdopen");
```

Nach Senden an Server (fprintf(tx,"msg")) immer if(fflush(tx)!=0){ fail(fflush); }; !

eher close und continue!

SERVER: SOCKET ERSTELLEN, VERBINDUNGSANNAHME VORBEREITEN UND VERBINDUNGSANNAHME:

```
int listensock = socket(AF_INET6, SOCK_STREAM, 0);
if (listensock == -1) fail("socket");
struct sockaddr_in6 name = {
  .sin6_family = AF_INET6,        muss übergeben werden
  .sin6_port   = htons(port);
  .sin6_addr   = in6addr_any,
};
if (bind(listensock, (struct sockaddr*)&name, sizeof(name)) == -1) { fail("bind"); }
if (listen(listensock, SOMAXCONN) == -1) { fail("listen"); }
for (;;) {                                  verbindung
  int clientsock = accept(listensock, NULL, NULL);
  if (clientsock == -1) { perror("accept"); continue; }
  // do something else handle connection(...);
  close(clientsock);   ← nur evtl!
}
```

SIGNALBEHANDLUNG //bsp Term. zombies sofort sammeln

```
sigset_t mask;
sigemptyset(&mask);
struct sigaction s = {
  .sa_handler = SIG_IGN   handle bei eigener Behandlung
  .sa_mask    = mask,
  .sa_flags   = SA_RESTART,   → nicht bei DFL und IGN
};
if (sigaction(SIGCHLD, &s, NULL) == -1) { fail("sigact"); }
```

PROZESS ERZEUGEN

```
pid_t pid = fork();
if (pid == -1) { perror("fork"); }
else if (pid == 0) { //child process
  hier ggf. listen sock close!
} else { //parent process ..}
```

INHALT EINES GEÖFFNETEN FILES AN CLIENT SENDEN:

```
int c;
while (1) {
  c = getc(file);
  if (c == EOF) {
    if (ferror(file)) {
      perror("file");
    }
    break;
  }
  putc(c, tx);
}
```

int fileno(FILE *stream);
returnd filedeskriptor d. stream
int chdir(path) : -1, sett errno

aus thread_handler returnen = pthread_exit

SIGNALE UND FORK/EXEC:

fork: Kindprozess erbt Signal & maske vom Vaterprozess
exec: Signalmaske und Behalten, Signalbehandlung wird nur behalten bei SIG_DFL oder SIG_IGN

PROZESSWEITE SIGNALMASKE BLOCKIEREN/FREIGEBEN/BLOCKIEREN

```
sigset_t set;
sigset_t oldset;
if (sigemptyset(&set) == -1) { die("sigemptyset"); }
if (sigaddset(&set, SIGCHLD) == -1) { die "sigaddset"; }
if (sigprocmask(SIG_BLOCK, &set, &oldset) == -1) { die ..}
```

how: SIG_BLOCK: alle maske u set
SIG_UNBLOCK: setzt alle maske n set
SIG_SETMASK: set als neue maske ← am besten

PASSIVES WARTEN AUF EIN SIGNAL - SIGSUSPEND

warten auf vordergrund-prozesse!

```
sigset_t set, oldset;
sigemptyset(&set); sigaddset(&set, sigprocmask(BLOCK))
while (!condition) {
  sigsuspend(&oldmask);
}
sigprocmask(SIG_SETMASK, &oldset, NULL)
```

waitpid-Aufruf nur im Handler

DATEIEN VON DATEIEN

```
if ((fdin = open(cmd->infile, O_RDONLY)) == -1) { die("open"); }
if (dup2(fdin, STDIN_FILENO) == -1) { die ("dup2"); }
if (close(fdin) == -1) { die("close"); }
```

fd1 → offset → Datei A
fd2 → offset → Datei 2
dup2(fd2, fd1)

BEISPIEL handler FUNKTION

```
int status, pid
int olderrno = errno
while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
  if (child_f_pid == pid) {
    child_f_pid = 0
  }
  pl_ist = delete_terminated(pid, pl_ist);
}
errno = olderno;
```

→ if (WIFEXITED(status)) printf(WEXITSTATUS(status));

NICHT BLOCKIERENDE SYNCHRONISATION (CAS):

```
do {
  // lokale kopie der kritischen Variable
  // Berechne Wert neuen wert
} while (CAS(krit Variable, wert alt, wert neu) == false);
```

THREADS ERSTELLEN

```
for (int i = 0; i < nThreads; i++) {
  pthread_t *t;
  if ((errno = pthread_create(&t, NULL, handler fkt, NULL)) != 0) {
    //Sachen freigeben und sterben
  }
  am Ende d. handlerfkt fdsocket(tx) fclose(tx)
}
```

args → cast v. arg in handler möglich: cast = struct triangle* args;

args als void* übergeben

```
volatile int counter;
pthread_mutex_t mutex;
pthread_cond_t cond;
};

SEM *semCreate(int initval) {
  SEM *s = malloc(sizeof(SEM));
  if (s == NULL) { return NULL; }
  s->counter = initval;
  if ((errno = pthread_mutex_init(&s->mutex, NULL)) != 0) {
    free(s); return NULL;
  }
  if ((errno = pthread_cond_init(&s->cond, NULL)) != 0) {
    pthread_mutex_destroy(&s->mutex);
    free(s); return NULL;
  }
  return s;
}
void P(SEM *sem) {
  if ((errno = pthread_mutex_lock(&sem->mutex)) != 0) {
    die ("pthread_mutex_lock");
  }
  while (sem->counter < 1) {
    if ((errno = pthread_cond_wait(&sem->cond, &sem->mutex)) != 0) { die ("pthread_cond_wait"); }
  }
  sem->counter -= 1;
  if ((errno = pthread_mutex_unlock(&sem->mutex)) != 0) {
    die ("pthread_mutex_unlock");
  }
}
void V(SEM *sem) {
  if ((errno = pthread_mutex_lock(&sem->mutex)) != 0) {
    die ("pthread_mutex_lock");
  }
  sem->counter += 1;
  if ((errno = pthread_cond_signal(&sem->cond)) != 0) {
    die ("pthread_cond_signal");
  }
  if ((errno = pthread_mutex_unlock(&sem->mutex)) != 0) {
    die ("pthread_mutex_unlock");
  }
}
void SemDestroy(SEM *sem) {
  if (SEM != NULL) {
    pthread_mutex_destroy(&sem->mutex);
    pthread_cond_destroy(&sem->cond);
    free(sem);
  }
}
```

VERZEICHNIS DURCHLAUFEN

```
struct dirent *entry;
DIR *curDir = opendir(path);
if (!curDir) {
  die("opendir");
}
while ((errno = 0, (entry = readdir(curDir)) != NULL)) {
  //if (!strcmp(entry->d_name, ".") || !strcmp(entry->d_name, "..")) { continue; }
  // mach Sachen
}
if (errno && entry == NULL) {
  perror("readdir");
}
closedir(dir); ← außerhalb Whileschleife
```

wie if(!curDir) && (errno != EACCES) die(opendir)
→ keine Zugriffsrechte

PRÜFEN OB REGULÄRE DATEI ODER DIRECTORY - STAT

```
struct stat s;
if (lstat(path, &s) == -1) { perror("lstat"); }
if (S_ISREG(s.st_mode)) { //...
if (S_ISDIR(s.st_mode)) { //...
```

PUFFER IMPLEMENTIERUNG

```
BNDBUF *bbCreate(size_t size) {
  if (size > (MAX)) {
    errno = EINVAL;
    return NULL;
  }
  BNDBUF *bb = malloc(sizeof(BNDBUF) + size * sizeof(*bb->data));
  if (bb == NULL) { return NULL; }
  bb->readIndex = 0;
  bb->writeIndex = 0;
  bb->size = size;
  bb->writeSem = semCreate((int)size)
  bb->readSem = semCreate(0);
  if (bb->writeSem == NULL || bb->readSem == NULL) {
    bbDestroy(bb);
    return NULL;
  }
  return bb;
}
void bbDestroy(BNDBUF **bb) {
  if (bb == NULL) return;
  semDestroy(bb->writeSem);
  semDestroy(bb->readSem);
}
void bbPut(BNDBUF *bb, int value) {
  P(bb->writeSem);
  bb->data[bb->writeIndex] = value
  bb->writeIndex = (bb->writeIndex+1) % bb->size;
  V(bb->readSem);
}
int bbGet(BNDBUF *bb) {
  size_t pos;
  int value;
  size_t nextpos;
  P(bb->readSem);
  do {
    pos = bb->readIndex;
    size_t index = pos % bb->size;
    value = bb->data[index];
    if (pos < SIZE_MAX) { //no overflow
      nextpos = pos + 1;
    } else { nextpos = index + 1; }
  } while (!atomic_compare_exchange_strong(&bb->readIndex, pos, nextpos));
  V(bb->writeSem);
  return value;
}
```

struct BNDBUF{
size_t size;
volatile size_t readIndex;
volatile size_t writeIndex;
SEM *readSem
SEM *writeSem;
int data[]
}

ohne CAS:
int bbGet(bb){
P(bb->readsem)
int idx;
idx = bb->out;
out = (out+1)%size;
V(bb->out)
return v;
}

SORT UND STRCMP:

```
static int compare(const void *a, const void *b) {
  const char *const *ia = (const char *const *) a;
  const char *const *ib = (const char *const *) b;
  return strcmp(*ia, *ib);
}
//dann aufrufen in:
qsort(eingabe, counter, sizeof(char*), compare);
//counter = Anzahl elemente mit größe
//eingabe = Start des zu vergleichenden Arrays
```

NÜTZLICHE STRING FUNKTIONEN:

int sprintf(char *s, char *format, ...) : schreibt string formatiert in string s (bei error (return) <0)
int sscanf(char *input, char *format, ...) : scannt input und speichert werte in nach format in werte abgelegt
Bsp: int matches = sscanf(line, "%d:%d", &x, &y)
if (matches != 2) //

ESSEN VON STRINGS

```
char *strtok(char *s, char *delim) : teilt string s beim Trennzeichen delim ab, erster aufruf mit string, alle weiteren mit NULL. Setzt \0. Amende liefert es NULL
char *strtok_r(char *s, char *delim, char **save) wie strtok, reentrant version
Bsp:
char *saveptr;
strtok_r("Hi du", " ", &saveptr)
char *strcpy(char *dest, char *src) : kopiert src in dest, mit /0 !buffer overflow möglich
gibt pointer zu dest zurück
char *strcat(char *dest, char *src) : hängt src ans Ende von dest
char *strstr(char *haystack, char *needle) : findet das erste vorkommen von string needle in str haystack oder NULL wenn nicht vorhanden
char *strncpy(char *dest, char *src, size_t n) : kopiert bis n chars von src in dest, wie strcpy sonst
int strcmp(char *a, char *b) : vergleicht 2 Strings, 0 wenn gleich, <0 wenn a<b aber >0, wenn a>b.
```

BEISPIEL GEBRAUCH MIT REALLOC:

```
name = eingabe = malloc(sizeof(name));
int counter = 0;
//Befüllen
if (!realloc(eingabe, ((++counter)+1) * sizeof(name)))
  //Fehlerbehandlung
qsort aufrufen
```

ZEILENWEISES EINLESEN VOM CLIENT:

```
char buffer[MAX_LINE + 1];
alle Zeilen:
while (fgets(buffer, sizeof(buffer), rx)) {
  //do sth
} if (ferror(rx)) perror("fgets") Fehlerbeh. nach Schleife!
fgets gibt buffer bei Erfolg zurück, und NULL bei error oder EOF
```

fgets liest bis EOF oder \n, \n wird in Puffer gespeichert!

```
else Bsp:
if (fgets(buffer, sizeof(buffer), rx) == NULL) {
  if (ferror(rx) != 0)
    perror fgets;
  else
    //EOF-handlen
```

DATEI ANLEGEN

```
FILE *f = fopen(char *name, char *mode)
mit mode: r=read, w=write, a=append, r+ = read+write
Fehlerfall: NULL → setzt errno, perror!
Schließen mit fclose(f)
schreiben und lesen wie beim server (fputs, fgets..)
```

MAKEFILE

```
CC = gcc
CFLAGS = -std=c11 -Wall -Werror -pedantic -D_XOPEN_SOURCE=700 -pthread
.PHONY: all clean
all: Name d. ausführb. Datei
clean:
  rm -f ausführb. und o Dateien
snail: snail.o → daraus passierts .o Dateien
  $(CC) -o snail snail.o
snail.o: snail.c + alle .h Dateien die included sind a
  $(CC) -c $(CFLAGS) -o snail.o snail.c
```

STATIC LIBRARY

```
libjbuffer.a : jbuffer.o semo
  ar rcs libjbuffer.a jbuffer.o sem.o
```

DYNAMIC LIBRARY

```
sem-dym.o : sem.c sem.h
  gcc -c -fpic $(CFLAG) -o $@ $<
jbuffer-dym.o : jbuffer.c jbuffer.h sem.h
  gcc -c -fpic $(CFLAG) -o $@ $<
libjbuffer-dynamic.so : sem-dym.o jbuffer-dym.o
  gcc -shared $(CFLAG) -o libjbuffer-dynamic.so
  sem-dym.o jbuffer-dym.o
//binden dynamisch bei Anwendung:
jbuffer-test-dynamic : jbuffer-test.o libjbuffer-dynamic.so
  gcc $(CFLAG) -o $@ $< -L. -ljbuffer-dynamic
```

FORK - ANZAHL PROZESSE LIMITIERT - SIGNALE

passives warten mit Signalen

handler für SIGCHLD

```
void handler(int signal)
int old = errno;
while (waitpid(-1, NULL, WNOHANG) > 0) {
  curproc--;
}
errno = old
```

mit curprocs global + static volatile int curprocs = 0;

```
vor fork:
sigset_t set, oldset;
sigemptyset(&set);
sigemptyset(&oldset);
sigaddset(&set, SIGCHLD);
sigprocmask(SIG_BLOCK, &set, NULL);
while (curprocs >= MAX_PROCS) {
  sigsuspend(&oldset);
}
Curprocs++;
sigprocmask(SIG_UNBLOCK, &set, NULL);
//dann normal forken, im parent oft mix
```

CLOSE ON EXEC - FCNTL

```
1. int fd = open("index.html", O_CLOEXEC)
2. int flags = fcntl(fd, F_GETFD, 0);
   fcntl(fd, F_SETFD, flags | FD_CLOEXEC);
(dup und dup2 setzen close on exec zurück, opendir setzt closeonexec automatisch)
```

EXEC

```
execlp(char *path, *arg1, *arg2, ..NULL)
execvp(char *path, char *argv[])
Name der Programmdatei muss doppelt übergeben werden als path und argv[0]
exec kehrt nur im Fehlerfall zurück! return -1, errno gesetzt
```

KILL

```
int kill(pid_t, int sig): Sendet signal zu Prozess(gruppe)
SIGTERM → beenden, kindprozesse
error: errno set, -1 returned
```

SNAIL

```
// Get own FQDN & username
long hostname_max = sysconf(_SC_HOST_NAME_MAX);
if (hostname_max == -1 && errno) die("sysconf");
char fqdn[hostname_max+1];
if (gethostname(fqdn, sizeof(fqdn)) == -1) die("gethost");
// DNS Lookup
struct addrinfo hints = {
  .ai_flags = AI_CANONNAME;
};
struct addrinfo *result;
int addrinfo = getaddrinfo(fqdn, NULL, &hints, &result);
if (addrinfo != 0) {
  fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(addrinfo));
  exit(EXIT_FAILURE);
}
char *fqdn = head->ai_canonname;
```

```
errno = 0
struct passwd *pws = getpwuid(userid)
if (pws == NULL) {
  if (errno == 0) {
    fprintf(stderr, "...");
    exit(EXIT_failure);
  } else { fail("getpwuid");
char senderaddress[sizeof(pws->pw_name) + sizeof(fqdn) + 2];
strcpy(senderaddress, pws->pw_name);
strcat(strcat(senderaddress, "@"), fqdn);
```

CHECKLISTE

- Ohne im Handler sichern
- strtok bei globalen Variablen und Funktionen
- free und close auch im Fehlerfall
- Fehlerbehandlungen!
- in Threads kein exit → fpt bei: Buffer?
- im Kind kein return → return NULL in Thread-fkt
- Mutex exit, Signale blockieren, generell Signale richtig
- return nicht vergessen (auch am Ende von Thread-fkt)

Aufgaben

- Wiederaufnahmemodell: erfolgreiche Behandlung der Ausnahmesituation führt zur Fortsetzung / Ausführung des unterbrochenen Programms. Trap kann interpretiert werden als bedingtes Unterbrechungsmodell, dann ist die Ausnahmesituation nicht schwerwiegend, wenn sie auf bewusstes Eingreifen der zum Abbruch führenden Programms steht. Trap kann, linker fast daraus nie so behandelt werden. Trap: falsche hardwarearguments (Rechnerfehler), Adressverletzung, unbekannter Befehl
- Terminierung: Signalisierung externer Ereignisse, Beendigung Ein

redirect stdout

```
int fdout;
if (cmd->outfile != NULL) {
  if ((fdout = open(cmd->outfile, O_WRONLY | O_TRUNC | O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP)) == -1) {
    perror; return -1;
  }
  if (dup2(fdout, STDOUT_FILENO) == -1) { perror; return; }
  if (close(fdout) == -1) { perror; return; }
```

int scandir(char *dirp, dirent ***namelist, int (*filter)(const dirent *))

->scannt Directory dirp, nicht filter() auf jeden Eintrag, der die Filter != 0 wurden in alphabetische Reihenfolge sortiert und im array namelist gespeichert. filter = NULL → alle Einträge ausgewählt

Zeichenketten alphabetisch vergleichen:

```
static int filterHiddenEntry(const struct dirent *entry) {
  return (entry->d_name[0] != '.');
}
if ((int erg = scandir(path, &entries, filter, alphasort)) {
  // Fehlerbehandlung, setzt errno
}
mit struct dirent **entries;
```

Wichtig bei mehrere Synchro

Vorteil: etwas parallelität, verklemmungsfreiheit, Nachteile: CAS schwierig -> komplette Transaktion muss wiederholt werden, Verhungern möglich, ggf. ineffizient, Algorithmen, ABA Problem möglich

```
const char *a : daten const, nur bei Aufruf mit char *a : const a: pointer const, daten not
```

Essen von synchro machen

unfaire Sperre: verwaltet prozess auf Warteschlange -> aktueller prozess wird nicht verzögert

blockierend: sperrt CPU-Verbrauch

nicht-blockierend: unterbricht nicht

- federgewichtige P profitieren nicht von mehreren Prozessoren, weil es keine Kontrolle von ihnen hat
- leicht & schwergewichtige können echt parallel laufen auf mehreren Kernen

Unterschied Monitor/pthread:

pthread - Mutexe, Joins, sync, cond-var
Monitor: Kapselung, Datenabstraktion, Komplexität von Prozessen gemeinsam genutzte Daten zu (Un)recht zusammenschaltet, Programmierer muss sich nicht explizit um Art der Synchronisation synchro-mechanismen

- mehrseitig: Schlüsselwörter, semaphor -> monitor gerahmen unter wechselseitigen Ausschluss gestellt
- einfach: Bedingungsvariablen

Wait: blockiert Prozess bis zum Eintreten eines Ereignis & implizite Monitorfreigabe
Signal: eintreten eines Ereignis und stellt

1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

verschnellern bei first-fit einfacher als wo

durch Virtualisierung ist der Eintrag von physikalischen RAM möglich, aber abwechseln auf logischer Ebene unmöglich ist.