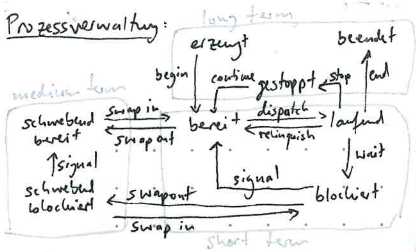
Prozessverwaltung:



erzeugt: wartet ggf. auf Speicherzuteilung
gestoppt: erwartet Fortsetzung
beendet: erwartet Entsorgung
} Lastkontrolle des Systems im Mehrprogrammbetrieb (wenn Ressourcen knapp werden)

schwebend bereit: Ausgelagert, aber bereit
" blockiert: wartet ausgelagert
} Umlagerung v. Programmen

bereit: auf Bereitliste, Scheduling bestimmt Pos.
laufend: CPU-Burst
blockiert: Zuteilung eines Betriebsmittels
} Einlastungsreihenfolge (mehrere Prozesse auf Prozessoren)

Kriterien: benutzer – / systemorientiert
Antwortzeit, Durchlaufzeit, Termineinhaltung, Vorhersagbark. / Durchsatz, Prozessorauslastung, Gerechtigkeit, Dringlichkeiten, Lastausgleich

Klassifikation: Einplanungsalgorithmen
kooperativ: abh. Proz., Abgabe nur Syscall, CPU-Monopolisierung
präemptiv: unabh. Proz., Abgabe CPU-ereig.bed., CPU-Schutz
deterministisch: alles bekannt, genaue Vorhersage, Zeitgarantien
probabilistisch: nicht " , Schätzungen, keine "
vorlaufend/offline: vor Betrieb, vollst. Ablaufplan, EZS
dynamisch/online: während Betrieb, Stapel/interaktive/verteilte S
symmetrisch: Befehlssatzebene, globale Bereitliste, ggseitige Beeinflussung, Lastausgleich
asymmetrisch: Maschinenprogrammebene, pro CPU lok. Bereitliste, ohne ggs. Beeinflussung; obligatorisch in asym., optional in sym. Multiprozessorsystemen

Verfahrensweisen:

	FCFS	RR	VRR	SPN	HRRN	SRTF	MLQ	MLFQ
kooperativ	x			x	x			
verdrängend		x	x			x		
dynamisch			x	x	x	x		x
probabilistisch				x	x	x		
statisch							x	

FCFS first come first served Konvoieffekt (lange Rechenst. bevorzugt)
RR round robin CPU-Schutz, Zeitscheibe, E/A benachteiligt.
VRR virtual RR RR mit Vorzugswarteschlange
SPN shortest process next nach erwarteter Zeit eingeplant
HRRN highest response ratio next ← periodisch umgeplant
SRTF shortest remaining time first (verdrängendes SPN, nach erw. Zeit eingeplant und spontan umgeplant)
ML(F)Q multilevel (feedback) queue
↳ Prozesse nach Typ in separate Liste

Monitore: krit. Abschnitte
- Prozeduraufruf sperrt Monitor, Rückkehr gibt frei
- Monitorwarteschlange: wartende Prozessexemplare für Eintritt
- ~~Ereignis~~ WS: PE, die Aufhebung der Wartebed. erwarten Ereignis

Bedingungsvariablen: blockierend (Vorrang Signalnehmer), nichtblockierend (" Signalgeber)
Hansen: blockierend, alle Signalnehmer auf bereit
Hoare: " , einen Signalnehmer auf bereit
} Signalnehmer darf laufen
Mesa: nichtblockierend, ein o. alle auf bereit } Signalnehmer behält Monitor

Bedingungsvariablen keinen zugänglichen Wert
wait: setzt Prozess bis Anzeige eines Ereignisses aus, gibt Monitor bis zur Wiederaufnahme implizit frei
signal: zeigt Ereignis an, wirkungslos ohne wartenden Prozess genau

Ereignisvariablen: cause/await
cause befreit alle

Syscall:
1. Prozessorstatus sichern
2. Syscall interpretieren
3. Prozessorstatus wiederherstellen &

Synchronisation

unterdrückend: verhindert Prozessauflösung anderer Prozesse
blockierend: sperrt Betriebsmittelvergabe an Prozesse
nichtblockierend: unterbindet Zustandsverstetigung durch Prozesse
wechselseitiger Ausschluss: vor Aktion Betriebsmittel sperren, nach Aktion " entsperren

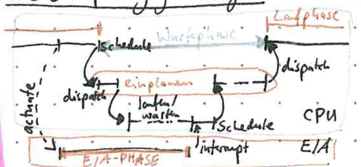
einseitige Synchronisation: Auswirkung nur auf einen Prozess (konsumierb. Betr.mittel, log. Synchr.)
Unterbrechungssteuerung, Verdrängungssteuerung (Bedingungssynchronisation)
mehrseitige Synchronisation: alle durchlaufen dasselbe Vergabeprotokoll (wiederverwendb. Betr.mittel)
↳ blockierend/pessimistisch
↳ nichtblockierend/optimistisch
} Synchr. Umlaufsperre
↳ lost wakeup: zw. Feststellung der Wartebed. eines Prozesses und der Blockierung wird Bed. aufgehoben

Fortschrittsgarantien:
obstruction-free: einzelner Prozess wird Aktion in begrenzter Anzahl v. Schritten beenden
lock-free: jeder Schritt trägt zum Fortschritt bei, starvation!
wait-free: garantiert systemweiten Fortschritt, keine starvation

Nachteile blockierender Synchronisation:
performance: Kreisch- & Busbreite
robustness: krit. Abschnitt abstürzen
interference: Prioritätsverletzung mit Scheduler
liveness: starvation / deadlock

Voraussetzungen für Verklemmungen
1. wechselseitiger Ausschluss
2. Nachforderung eines / mehrer. Betriebsmittel
3. Unentziehbarkeit der zugeteilten "
4. zirkuläres Warten

Prozesseinplanungsgrundlagen:



actuate: laufender Prozess stößt E/A-Vorgang an
schedule: wartet passiv auf Beendigung von E/A
dispatch: lastet eingeplanten Faden ein
interrupt: Ende E/A
schedule: wartender Faden wird eingeplant
dispatch: Eingelastet sobald dann

Mittlere Fadenverzögerung: $\frac{1}{n}\sum_{i=1}^{n}(i-1)\cdot k = \frac{n-1}{2}\cdot k$

exception: Sonderfall, der Unterbrechung/Abbruch der Ausführung bedeutet
→ Wiederaufnahme: nach Behandlung
→ Termination: Schutz/Zugriffsverletzung

Ausführung:
1. Ausführung eines Maschinenprogramms durch CPU
2. Ausnahme detektiert
3. Teilinterpretation, Ausnahmebehandlung
4. Beendigung der ↗, Wiederaufnahme

Betriebssystemlatenz: Zeit zw. Abfangzeitpunkt und Wiederaufnahme
Unterbrechungslatenz: Zeit zw. Wahrnehmung durch CPU und Annahme der Unterbr.
trap: synchron, vorhersagbar, reproduzierbar
interrupt: asynchron, unvorhersagbar, nicht reproduzierbar

Repräsentation
verankert: innerhalb: Betriebssystem/Kernel, Prozessinkarnation als Wurzel
oberhalb: Laufzeit/Anwendungssystem, " als Blatt o. innerer Knoten

Gewichtsklasse
featherweight: im selben Adressraum, ebenda fortsetzend
lightweight: im Kernadressraum, denselben Anwendungsadr-raum teilend
heavyweight: im " , im anderen Anwendungsadressraum landend

Adressräume

real: lückenhafter, wirklicher Hauptspeicher
logischer: konzeptionell lückenfreier, " , kann nicht swappen
virtueller: lückenloser, scheinbarer " , "

Paging: PT mit Seitendeskriptoren:
- Kachel-/Seitenrahmennummer
- Schreibschutz-/Präsenz-, sticky-, Modifikationsbit

Segmentierung: ST mit Segmentdeskriptoren:
- Basis, Limit
- Typ (Text, Daten, Stapel), Zugriffsrechte

Policies: placement: Wo ist Platz?
fetch: Wann muss Datum im HS sein?
replacement: Welches Datum ist ersetzbar?
Speicherzuteilung: Maschinenprogr. + BS (Halde)
Speichervirtualisierung: BS

best-fit: aufsteigende Lochgrößen, kleinstes passendes Loch } beste Zuteilung, min. Verschnitt, kleine Löcher, steigender Suchaufwand
worst fit: absteigende Lochgrößen, größtes passendes Loch } schnell, begünstigt Zerstückelung, konstanter Suchaufwand
first fit: (Adresse sortiert, Lochliste) schnell, verschwendung, kleine Löcher vorne, große hinten, steigender Suchaufwand
next fit: reihum Variante von first fit, gleich große Löcher, Suche beginnt bei zuletzt zugeteilten Loch, eher abnehmender Suchaufwand
buddy: sucht kleinstes passendes Loch buddy; der Größe 2^i, $2^{i-1} < size < 2^i$

Speicherverschnitt
intern: Paging, Buddy
extern: Segmentierung, "
Verschmelzung:
buddy: mit Buddy verschm.
first/next: beim Einsortieren Nachbarschaft prüfen
best/worst: beim Einsortieren → Liste durchlaufen + Nachbarn finden
Umlagern: Gebrauchsstücke + Kopiervorgänge + Relokation

intern: Speicherverwaltung gibt größere Blöcke als benötigt. Diff. ist interne Frag. keine GG-Maßnahmen
extern: Speicherbereiche sind nicht verschmelzbar und lassen sich nicht nutzen
Maßn.: Verschmelzen, Umlagen

Verklemmung:
zykl. Wartesituation zw. mehreren Prozessen

Liveloch:
Zwischen zwei Zust. hin und her, können nicht ausbrechen

Reihenfolge o. freiwillige Abgabe

Wiederaufnahmemodell
Fortsetzung des unterbrochenen Programms, immer bei Interrupts, kein bei Traps
Beendigungsmodell
Abbruch, Traps hingegen so

Einplanung (scheduling) / Umplanung (rescheduling)
begin: nach Erzeugung des Prozesses
relinquish: wenn Prozess freiwillig CPU abgibt
signal: Von Prozess erwartetes Ereignis ist eingetreten
continue: Prozess kann wieder aufgenommen werden
Scheduling latency: Zeitdauer Ein/Umplanung
dispatching latency: Zeitspanne Einplanung - Prozessorzuteilung
Ereignis → Signalisierung → Einplanung ↓ Reaktion ← Einlastung

Ersetzungsstrategien:
FIFO, LFU (least freq. used)
LRU (least recently used):
Second-chance: aging register, regelmäßiger Check, ob use-Bit gesetzt

OPT/MIN: Seite, die zukünftig nicht mehr ref. wird.
→ Seitenersetzungsstrategie, die mit Hilfe eines Referenzbits eine einfachere Impl. an LRU realisiert

dauerhafte Betriebsmittel: CPU, GPU, RAM, I/O, krit. Abschn. → belegt, benutzt, freigegeben
konsumierbar: Signal (trap), Interrupt Request, Meldung → produziert, empfangen, benutzt, freigegeben

Namen
Kontext: Namensraum flacher Struktur
derselbe Name in mehreren Kontexten
inode: user ID, group ID, Rechte, Zeitstempel, Anzahl Verweise, Typ (Verzeichnis, symlink, reg. Dat...)
inode table: array von Indexknoten
directory: Abbildungstabelle
file: abgeschl. Einheit zsh. Daten
Symboltabelle: alle def. Symbole mit Wert
Relokationstabelle: alle undefinierten Stellen

Terminvorgaben: nicht termingerecht??
weich/soft: weiterhin von Nutzen, verliert aber
fest/firm: Erg. verworfen, Prozess neugestartet, tolerierbar
hard/strikt: Katastrophe! Verletzung nicht tolerierbar.

RAID 0: über mehrere Platten, keine Sicherheit
RAID 1: mehrfache Platten, Platte kann ausfallen, schnelleres Lesen
RAID 4: über mehrere Platten, eine Parität, Platte kann ausfallen, schnelles Lesen
RAID 5: Parität über alle Platten
RAID 6: 2x Parität

benutzerorientiert:
- gute Reaktion
- häufige Prozesswechsel
systemorientiert
- Optimierung der Rechenzeit
- optimale Ressourcenausnutzung
Echtzeit: Termineinhaltung, Vorhersagbarkeit, Dringlichkeit

Userthreads: "federgewichtig" nicht aufgeteilt
- 1:many, Anwendungsebene
- Kernel sieht nur einen Kontrollfluss
- Erzeugung + Umschaltung sehr billig
- Programmierer kann über Scheduling Strategie entscheiden
- im lokalen Benutzeradressraum

Kernel-Threads: "leichtgewichtig"
- 1:1
- jeder Thread ist dem BS bekannt
- Erzeugung + Umschaltung > UT
- gemeinsamer Adressraum

Kernel-Threads: "schwergewichtig"
- 1:1
- Prozessinkarnation in eigenem Adressraum
- teurer als leichtgew. Prozesse

Aufteilung auf mehrere Prozessoren, parallel ausführbar

Pagefault Schritte:
1. MMU löst Trap aus
2. BS löst PageFault-Handler aus
3. Freier Speicher gesucht, evtl. ausgelagert
4. Einlagerung wird angestoßen ↖ 5. Zustand blockiert
6. Interrupt an CPU
7. Present Bit 1
8. → Bereit, 9. → laufend
10. Bei Einlastung Befehl wiederholen

Compare-Fkt: (compare)

```c
static int compare (const void* x, const void* y){
    const char *a = *(const char **)x;
    return strcmp(a,b); }
```

Sortieren: (Sort)

```c
# define eva ;;
int platz=1
int zaehler=0
char** list= malloc (1* sizeof(char*));
if(list== NULL) { die("malloc");}
for (eva){  if (platz== zaehler) {platz *=2;      Fehlerbeh.
    list = realloc (list, platz * sizeof(char*));
char input[LENGTH+1];
    if (fgets(input, LENGTH+1,stdin) == NULL) {
        if (feof(stdin)!= 0){break;}else{die("fgets");}
    if (input [strlen (input) -1] != '\n'){ int c= fgetc(stdin);
        while (! (c=='\n' || c=='EOF')) { c= fgetc(stdin);}
        continue; }
    list [zaehler] = malloc (102* sizeof(char*)); FEHLER
    strcpy (list[zaehler], input); zaehler++;
}
qsort(list, zaehler, sizeof(char*), compare);
free(list);
```

```c
while ((w= waitpid (-1, &wstatus, WNOHANG)) >0) ...
pid_t w; int wstatus;   if WIFEXITED(wstatus)
                        printf("...", WEXITSTATUS(wstatus))
```

Fork

```c
pid_t p;   p= fork();
if (p== -1) {die(fork)}
else if (p==0) {Kind;} else {Eltern}
```

Scanf

```c
while (! feof(stdin)){
    int a,b,c,d,e,f; char tmp[2];
    int scan = scanf("... %1[\n]", &a, ..., tmp);
    if (scan < 7) //bis hier
```

Threads

```c
pthread_t thread;
int result= pthread_create(&thread, NULL, thread_start, arg);
if (result != 0) die;
result = pthread_detach (pthread_self());
if (result != 0) die
static void *thread_start (void *arg){
    (void) arg;
    return NULL;}
```

Verzeichnisse

```c
DIR *dir= opendir ("path"); struct dirent *entry;
if (dir == NULL) die(..);
while (errno=0, (entry= readdir(dir)) != NULL) {
    if (strcmp (entry->d_name, "."/"..") == 0 ||..) {
        continue;
    char path [strlen(path) + strlen(entry->d_name)+2];
    sprintf (path, "%s/%s", path, entry->d_name);
    struct stat info;
    if (lstat (path, &info) != 0){ continue; // Fehler}
    if (!S_ISREG(info.st_mode) || !S_ISDIR(info.st_mode)){continue;}
    printf ("%s: %d bytes\n", path, info.st_size);
}
if (errno != 0) die(...);
if (closedir (dir) != 0) die(...);
```

```c
errno= 0; char *endptr;
long portasint = strtol (port, &endptr, 10);
if ((errno== ERANGE && (portasint == LONG_MIN || MAX))
    || (errno != 0 && portasint == 0) || (*port == '\0')
    || (*endptr != '\0')) { //wrong port }
if (portasint > INT_MAX || < 0)   // zu groß/klein
```

Umleitung

```c
int fd_in; if ((fd_in = open ("file", O_RDONLY)) ==-1) die(..);
if (dup2 (fd_in, STDIN_FILENO) == -1) die(..);
if ((fd_out = creat ("file", S_IRUSR | S_IWUSR |
    S_IRGRP, S_IROTH)) == -1) die
if (dup2 (fd_out, STDOUT_FILENO) == -1) die...
```

Connect to server

```c
struct addrinfo hints= {
    .ai_socktype = SOCK_STREAM,
    .ai_family = AF_UNSPEC,
    .ai_flags = AI_ADDRCONFIG,
};
struct addrinfo *head;
int addrinf = getaddrinfo ("host", port, &hints, &head);
if (addrinf != 0) fprintf("%s\n", gai_strerror(addrinf));
int socke;
struct addrinfo *curr;
for (curr=head; curr != NULL; curr = curr->ai_next){
    socke= socket (curr->ai_family, curr->ai_socktype,
        curr->ai_protocol);
    if (socke == -1) continue;
    if (connect (socke, curr->ai_addr, curr->ai_addrlen)==0)
        break;
    close (socke);
}
freeaddrinfo (head);
if (curr == NULL) {die ("connect");}
```

~~Fdopen~~ File

```c
FILE *rx = fdopen (socke, "r");
if (rx == NULL) die ("fdopen"); close (socke); cont.
int secondSocke = dup (socke);
if (secondSocke == -1) die ("dup"); fclose (rx); close (socke); cont.
FILE *tx = fdopen (secondSocke, "w");
if (!tx) fclose(rx); close (fd2); close(fd); cont.
```

Client Server

```c
int listenSocke = socket (AF_INET6, SOCK_STREAM, 0);
if (listenSocke == -1) die ..
struct sockaddr_in6 name = {
    .sin6_family = AF_INET6,
    .sin6_port = htons (portasint),
    .sin6_addr = in6addr_any, };
if (bind (listenSocke, (struct sockaddr *) &name, sizeof (name)) == -1)
    {die ("bind")};
if (listen (listenSocke, SOMAXCONN) == -1) {die ..}
for (eva){
    int clientSocke = accept (listenSocke, NULL, NULL);
    if (clientSocke == -1) continue;
    // do sth. bbPut (clientSocke)
    close (clientSocke); }
```

Sig Handling

```c
                   static void signalhandler (int sign)
                   int e = errno; ... errno=e;
struct sigaction action {
    .sa_handler = SIG_IGN, // Signalbeh. Fkt. etc.
    .sa_flags = SA_RESTART, };
sigemptyset (&action, sa_mask);
if (sigaction (SIGPIPE, &action, NULL) == -1) die ("sigaction");
   ↑ abzufangendes Signal
```

Blockierung

```c
sigset_t set, oldset;
sigemptyset (&set);
sigaddset (&set, SIGCHLD);
sigprocmask (SIG_(UN)BLOCK, &set, &oldset);
// do stuff
sigprocmask (SIG_SETMASK, &oldset, NULL);
while (wolfgang == 0) { //warte auf signal
    sigsuspend (&oldset); //nimmt alte Maske bis Signal
}
wolfgang = 0;
```

```c
int flags = fcntl (fileno (..), F_GETFD, 0);
if (flags == -1) die ..
if (fcntl (fileno(..), F_SETFD, flags | FD_CLOEXEC) == -1)
    die ("fcntl");
int fd = open ("path", O_RDONLY | O_CLOEXEC);
if (fd == -1) die ("open");
```

```c
struct dirent **eintraege;
int anzahl = scandir (fullpath, &einträge, filter, alphasort);
if (anzahl == -1) { // Fehlerbeh. }
dirListingBegin (rtx, path);
for (int i=0; i< anzahl; i++) {
    dirListingPrintEntry (tx, path, eintraege [i]->d_name);
    free (eintraege [i]);
}
dirListingEnd (write);
free (eintraege);
static int filter (const struct dirent *meow) {
    if (meow->d_name[0] == '.') { return 0;
    return 1;
```

Input

```c
char request [8192+1];
if (fgets (request, 8192+1, rx) == NULL) {
    if (ferror (rx)) { fprintf (stderr, "fgets") }
    if (feof (rx)) {...}
}
char *line = NULL;
size_t blub = 0;
if (getline (&line, &blub, rx) == -1) {
    free (line); ... }
for (eva) {
    int akt = getc (stdin);
    if (akt == EOF) {
        if (ferror (stdin) != 0) {
            fprintf (stderr, "getc");
            exit (EXIT_FAILURE);
        }
        if (feof (stdin) != 0) {
            break;
    } ... do stuff ...
    fputc (akt, tx);
}
```

```c
static void toilette (FILE* kacka) {
    int klopapier = fflush (kacka);
    if (klopapier != 0) { die ("fflush");
}
```

```c
static void die (char* msg) {
    perror (msg);
    exit (EXIT_FAILURE);
}
```

fclose nicht vergessen
free "   "

static das
Schwein + Fktdef.
fflusch + Fehler
fflausch das
Schaf   mutexe destroy
close/fclose
freeaddrinfo   semDestroy

```c
static sigset_t set, oldset;   static volatile
                               unsigned int proccounter;
int main (..) {
    struct sigaction sa = {
        .sa_flags = SA_RESTART,
        .sa_handler = handler,
    };
    sigemptyset (&sa.sa_mask);
    sigaddset (&set, SIGCHLD);
    ...
    sigprocmask (SIG_BLOCK, &set, &oldset);
    while (proccounter >= MAX_PROCS)
        sigsuspend (&oldset);
    :
    sigprocmask (SIG_SETMASK, &oldset, NULL);
```

getaddrinfo

MEOW!

Makefile

```make
CC = gcc
CCFLAGS = -std=c99 -Wpedantic
    -Wall -Werror -Wconversion
    -Wextra -Wshadow -pthread
    -D_XOPENSOURCE=700 -g
AR = ar
ARFLAGS = -rcs
all: jbuffer-test jbuffer-test-dynamic
clean:
    rm -rf sem.o semdyn.o jbuffer.o
    jbufferdyn.o libjbuffer.a jbuffer-test
    libjbuffer-dynamic.so jbuffer-test-dynamic
jbuffer-test-dynamic: jbuffer-test.o libjbuffer-dyn.so
    $(CC) -pthread -o $@ jbuf-test.o
        -ljbuffer-dynamic -L.
libjbuffer-dynamic.so: semdyn.o jbufferdyn.o
    $(CC) $(CCFLAGS) -shared -pthread -o $@ $^
semdyn.o: sem.c sem.h
    $(CC) $(CCFLAGS) -pthread -fPIC -o $@ -c $<
jbufferdyn.o: jbuffer.c jbuffer.h sem.h
    $(CC) $(CCFLAGS) -pthread -fPIC -o $@ -c $<
jbuffer-test: jbuffer-test.o libjbuffer.a
    $(CC) -pthread -o $@ jbuffer-test.o -ljbuffer -L.
libjbuffer.a: jbuffer.o sem.o
    $(AR) $(ARFLAGS) -o $@ $^
sem.o: sem.c sem.h
    $(CC) $(CCFLAGS) -c -pthread $<
jbuffer.o: jbuffer.c jbuffer.h sem.h
    $(CC) $(CCFLAGS) -c -pthread $<
EASY:
%.o: %.c
    $(CC) $(CCFLAGS) -c $<
sister.o: sister.c connection.h cmdline.h
snail: snail.o
    $(CC) -o snail snail.o
snail.o: snail.c
    $(CC) $(CCFLAGS) -c -o snail.o snail.c
```

SIGNAL MASKE
thread-lokal →
pthread_sigmask

SEM

```c
#include <pthread.h>, <errno.h>, <stdlib.h>
#include "sem.h"
struct SEM {
    volatile int val;
    pthread_mutex_t m;
    pthread_cond_t c;
}
SEM *semCreate (int initVal) {
    int errno = errno;
    SEM *meow = malloc (sizeof (struct SEM));
    if (meow == NULL) return NULL;
    errno = pthread_mutex_init (&meow->m, NULL);
    if (errno != 0) { free (meow);
        errno = errno; return NULL; }
    errno = pthread_cond_init (&meow->c, NULL);
    if (errno != 0) {
        pthread_mutex_destroy (&meow->m);
        free (meow); errno = errno;
        return NULL; }
    meow->val = initVal;
    return meow;
}
void semDestroy (SEM *sem) {
    if (sem == NULL) return
    pthread_cond_destroy (&sem->c);
    pthread_mutex_destroy (&sem->m);
    free (sem);
}
void P (SEM *sem) {
    pthread_mutex_lock (&sem->m);
    while (sem->val <= 0) {
        pthread_cond_wait (&sem->c,
            &sem->m); }
    sem->val = sem->val - 1;
    pthread_mutex_unlock (&sem->m);
}
void V (SEM *sem) {
    pthread_mutex_lock (&sem->m);
    sem->val = sem->val + 1;
    if (sem->val == 1) {
        pthread_cond_broadcast (&sem->c);
    }
    pthread_mutex_unlock (&sem->m);
}
```

```c
static void signal (int signal) {
    int e = errno;
    while (waitpid (-1, NULL, WNOHANG) > 0)
        // do sth
        // processes --;
    errno = e;
}
```

```c
int fd;
FILE *f = fdopen (fd, "w");
if (!f) ...
fclose (f)
close (fd)
```

JBUF

```c
#include <stdlib.h>, <limits.h>, <stdbool.h>
                     <errno.h>
#include "jbuffer.h", "sem.h"
struct BNDBUF {
    size_t size;
    volatile size_t schreib, les;
    SEM *voll;
    SEM *frei;
    int mudas_array []; };
BNDBUF *bbCreate (size_t size) {
    if (size > INT_MAX || size <= 0) {
        errno = EINVAL; return NULL; }
    BNDBUF *bluff = malloc (sizeof (struct
        BNDBUF) + size * sizeof (int));
    if (bluff == NULL) return NULL;
    bluff->size = size;
    bluff->schreib = 0; bluff->les = 0;
    bluff->voll = semCreate (0);
    if (bluff->voll == NULL) {free(bluff); return NULL}
    bluff->frei = semCreate ((int) size);
    if (bluff->frei == NULL) { semDestroy (bluff->voll)
        free (bluff); return NULL; }
    return bluff;
}
void bbDestroy (BNDBUF *bb) {
    if (bb == NULL) return;
    semDestroy (bb->voll); semDestroy (bb->frei)
    free (bb);
}
void bbPut (BNDBUF *bb, int value) {
    P (bb->frei);
    bb->mudas_array [bb->schreib] = value;
    bb->schreib = (bb->schreib +1) % bb->size;
    V (bb->voll);
}
int bbGet (BNDBUF *bb) {
    int readint;
    size_t pos, nextpos;
    P (bb->voll);
    do { pos = bb->les;
        nextpos = (bb->les+1) % bb->size;
        readint = bb->mudas Array [pos];
    } while (CAS (&bb->les, pos, nextpos) == false)
    V (bb->frei);
    return readint;
}
```

CAS = __sync_bool_compare_and_swap

strtok_r

```c
char *vorname, *nachname, *rest, *tmp;
if ((vorname = strtok_r (input, " \t\n", &tmp)) == NULL
    { continue; }
if ((nachname = strtok_r (NULL, " \t\n", &tmp)) == NULL)
    { continue; }
if ((rest = strtok_r (NULL, " \t\n", &tmp)) == NULL)
    { continue; }
strcpy (list [zaehler].first, vorname); ...
```

```c
int namecmp (const void *p1, const void *p2)
    const struct name *a = (const struct name*) p1
       "     "       *b = (const struct name*) p2
    if (strcmp (a->last, b->last) == 0) {
        return (strcmp (a->first, b->first)); }
    return strcmp (a->last, b->last);
}
```

Buffer für Filepointer:

```c
FILE *tx = fdopen (clientSock, "w");
if (!tx) { perror ("fdopen");
    close (clientSock);
    continue; }
```

```c
FILE *dev = fopen (device, "a");
                            ↑ append
while ((c = fgetc (rx)) != EOF) {
    if (fputc (c, dev) == EOF) {
        perror ("fputc");
        continue; break;
    }
}
```