Semaphore: P: `{ S{M * sem, unsigned n) { pthread_mutex_lock(&sem->mutex); while (sem->value < n) { pthread_cond_wait(&sem->cond, &sem->mutex); } sem->value -= n; pthread_mutex_unlock(&sem->mutex); } V(S{M * sem, unsigned n) { pthread_mutex_lock(&sem->mutex); sem->value += n; pthread_cond_signal(&sem->cond); pthread_mutex_unlock(&sem->mutex);`

Parameter verarbeiten: `for (int i = 1; i < argc; i++) { myfunction(argv[i]); }` **Server Socket (wartet auf eingehende Nachricht):** `struct sigaction action = { .sa_handler = SIG_IGN, }; sigemptyset(&action.sa_mask); sigaction(SIGPIPE, &action, NULL); int server_socket = socket(AF_INET6, SOCK_STREAM, 0); if (server_sock == -1) { die("socket"); } struct sockaddr_in6 name = { .sin6_family = AF_INET6, .sin6_port = htons(portNo), .sin6_addr = in6addr_any }; if (bind(server_sock, (struct sockaddr*) &name, sizeof(name)) == -1) { die("bind"); } if (listen(server_sock, SOMAXCONN) == -1) { die("listen"); } // Ab hier wird while(1) { int client_sock = accept(server_sock, NULL, NULL); if (client_sock == -1) { perror("accept"); } else { myfunction(); }}`

Thread Pool aufsetzen: `pthread_t workers[THREADS], for (int i = 0; i < THREADS; i++) { int tmp_errno = pthread_create(&workers[i], NULL, &worker, &i); // Aufgabe z.B. &params); if (tmp_errno != 0) { errno = tmp_errno; die("pthread_create"); }} worker function:`

Signalbehandlung: `struct sigaction = { .sa_handler = &[Name] oder SIG_IGN, .sa_flags = SA_RESTART, }; sigemptyset(&action.sa_mask); sigaction(SIGINT, &action, NULL);` returns NULL

Prozess warten auf SIGINT: `sigset_t mask; sigfillset(&mask); sigdelset(&mask, SIGINT); sigsuspend(&mask)` **Auf Threads warten:** `for (int i = 0; i < THREADS; i++) { int tmp_errno = pthread_join(threads[i]); if (tmp_errno != 0) { errno = tmp_errno; die("pthread_join"); }` **Listen Socket:** `int listen_sock = socket(AF_in6, SOCK_STREAM, 0); if (listen_sock < 0) die("socket"); struct sockaddr_in6 name = { .sin6_family = AF_INET6, .sin6_port = htons(port), .sin6_addr = in6addr_any }; if (bind(listen_sock, (struct sockaddr*) &name, sizeof(name)) < 0) die("bind"); if (listen(listen_sock, SOMAXCON) < 0) die("listen"); return listen_sock;` **File exzergen:** `r = read w = write FILE *rx = fdopen(socket, "r") if (rx == NULL) { die("fdopen"); } // Falls sowohl r als auch w notwendig, int socket2 = dup(socket) if (socket2 < 0) die();`

File Lesen: `fclose(rx)` **Zeilen von File einlesen:** `char buf[n]; while (fgets(buf, sizeof(buf), stream) != NULL) { size_t len = strlen(buf); if (len > sizeof(buf)-2 && buf[len-1] != '\n') { int c; while ((c = fgetc(stream)) != EOF) { if (c == '\n') { break; } } if (ferror(stream)) { die("fgets"); }} if (buf[len-1] == '\n') buf[len-1] = '\0'; } // Zeile in x verarbeiten` `(x = char *x)` **Client Socket DNS Anfrage + verbinden mit URL:** `char *url, char *port struct addrinfo hints = { .ai_socktype = SOCK_STREAM, ai.family = AF_UNSPEC, ai.flags = AI_ADDRCONFIG, }; struct addrinfo *head; int ret_addrinfo = getaddrinfo(url, port, &hints, &head); if (ret_addrinfo == EAI_SYSTEM) { die("getaddrinfo"); } else if (ret_addrinfo != 0) { fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(ret_addrinfo)); exit(EXIT_FAILURE); } int sock; struct addrinfo *curr; for (curr = head; curr != NULL; curr = curr->ai_next) { sock = socket(curr->ai_family, curr->ai_socktype, curr->ai_protocol); if (sock == -1) { perror("socket"); continue; } int ret_con = connect(sock, curr->ai_addr, curr->ai_addrlen); if (ret_con == -1) { perror("connect"); } else if (ret_con == 0) { break; } else if (close(sock) == -1) { perror("close"); }} if (curr == NULL) { fprintf("keine Adressen gefunden\n"); exit(EXIT_FAILURE); }` **Jobprozess SIGKILL schicken** `void *timeout(void *job) Jobinfo *jobinfo = (Jobinfo*) job; sleep(10); if (kill(job->info->pid, SIGKILL) == -1) { perror("kill"); pthread_exit(NULL); }` **Ganzes Directory / Verzeichnis durchgehen:** `char *path = "." DIR *dir = opendir(path); if (dir == NULL) { die("opendir"); } struct dirent *dirent; while (errno = 0, (dirent = readdir(dir)) != NULL) { if (strcmp(dirent->d_name, ".") == 0 || strcmp(dirent->d_name, "..") == 0) { continue; } char npath[strlen(path) + strlen(dirent->d_name) + 2]; if (sprintf(npath, "%s/%s", path, dirent->d_name) < 0) { perror("sprintf"); continue; } struct stat info; if (lstat(npath, &info) == -1) { perror("lstat"); continue; } if (S_ISREG(info.st_mode)) { // regular file } else if (S_ISDIR(info.st_mode)) { // directory } } if (errno) { die("readdir"); } if (closedir(dir) == -1) { die("closedir"); }`

CAS: `atomic_int a = ATOMIC_VAR_INIT(10); int old, local; do { old = atomic_load(&a); local = old-1; } while (!atomic_compare_exchange_strong(&a, &old, &local));` **Warten auf bestimmten Child:** `pid_t pid = fork(); ... int wstatus; while (waitpid(pid, &wstatus, 0) != -1) { if (WIFEXITED(wstatus) || WIFSIGNALED(wstatus)) { return; }} perror("waitpid");` **Warten auf alle Childs:** `int status; pid_t child = waitpid(-1, &status, WNOHANG); if (child == -1) { perror("waitpid"); } else if (child == 0) { // Childs nicht terminiert } else { ... // Childs done } // extra Status ausgeben: if (!WIFEXITED(status)) { if printf(stderr, "%s\n", WEXITSTATUS(status)) < 0) { perror("fprintf"); }}}` **FORK:** `pid_t p = fork(); if (p == -1) { die("fork"); } else if (p == 0) { // Kind } else { // Elternprozess }` **Pattern Matching:** `int fn_res = fnmatch("*.txt", basename(path), FNM_PERIOD); if (!fn_Res) { // path endet mit .txt } else if (fn_res == FN_NOMATCH) { // no match } else { die(); }` **Input mit Trennzeichen trennen:** `char input[] char delim[] = ":" // Trennzeichen hier - char *argv[strlen(input)/2+1]; argv[0] = strtok(input, delim); int index = 1; while ((argv[index++] = strtok(NULL, delim)) != NULL);` **Pointer zum 1. Char eines Strings:** `if (strchr(myText, '\n') != null) {...}` **A an B anhängen (Strings):** `char full_path[strlen(path) + strlen(address) + 1]; strcpy(full_path, path); strcat(full_path, address);` **Makefile:** `CC = gcc CFLAGS = ... Phony: all clean all: patriots clean: rm -f patriots patriots.o %.o: %.c $(CC) $(CFLAGS) -c -o $@ $<`

`%: %.o $(CC) $(CFLAGS) -o $@ $^ $<`

`patriots.o: patriots.c triangle.h bluffer.h`

`patriots: patriots.o triangle.o bluffer.o sem.o`