Bauhaus-Universität Weimar

Fakultät Medien

Vernetzte Medien

Dr.-Ing. Günther Schatter

Weimar, April 2000

Lehrgebiet Betriebssysteme/Systemprogrammierung

Unterlagen für die Systemprogrammierung in C unter UNIX

Inhalt

Anmerkung:
Die Unterlagen wurden anhand von Literatur und Firmenunterlagen nach bestem Wissen zusammengestellt.
Abweichungen sind durch technische Weiterentwicklungen bedingt und nicht auszuschließen.
Hinweise erbeten an guenther.`schatter@medien.uni-weimar.de`

call, function	*Bedeutung*	*Zuordnung*			
		file	IPC	proc	sys
`access()`	Zugriffsrechte File prüfen	•			
`alarm()`	Weckeralarm einstellen			•	
`chdir()`	Wechsel Directory	•			
`chmod()`	Zugriffsrechte File ändern	•			
`chown()`	Besitzrechte File ändern	•			
`close()`	File schließen	•			
`closedir()`	Directory schließen	•			
`creat()`	File definieren	•			
`ctime()`	Konvertiert Zeit/ Datum in String				
`dup()`	File einen weiteren Filedeskriptor zuordnen	•			
`exec..()`	Programmüberlagerung			•	
`exit()`	Prozeß beenden			•	
`fclose()`	File schließen, mit Pointeradressierung	•			
`fcntl()`	File-Kontrolloperationen	•			
`fopen()`	File öffnen, mit Pointeradressierung	•			
`fork()`	Prozeß erzeugen			•	
`fread()`	File lesen, mit Pointeradressierung	•			
`fseek()`	Filepointer positionieren, Pointeradressierung	•			
`fwrite()`	File schreiben, mit Pointeradressierung	•			
`getc()`	Zeichen aus File lesen	•			
`getchar()`	Zeichen von stdin lesen	•			
`getenv()`	Umgebungsvariable holen				•
`getpid()`	Prozeß-ID bestimmen			•	
`getuid()`	User-ID bestimmen			•	
`ioctl()`	Terminal-Kontrolloperationen	•			
`kill()`	Signal senden		•		
`link()`	Link definieren	•			
`lseek()`	Filepointer positionieren	•			
`malloc()`	Speicher allokation			•	
`mkdir()`	Directory erzeugen	•			
`mknod()`	Directory oder File erzeugen	•			
`msgctl`	Messagequeue Kontrolloperationen		•		
`msgget()`	Messagequeue definieren		•		
`msgrcv()`	Message aus Queue lesen		•		
`msgsnd()`	Message an Queue senden		•		
`open()`	File öffnen	•			
`pause()`	auf Signal warten			•	
`perror()`	Fehlerausgabe				•
`pipe()`	Pipe definieren		•		
`poll()`	Input/ Output multiplexen	•			
`printf()`	formatierte Ausgabe auf stdout	•			
`putc()`	String auf stdout ausgeben	•			
`putchar()`	Zeichen auf stdout ausgeben	•			
`read()`	File lesen	•			
`rmdir()`	Directory löschen	•			
`scanf()`	formatierte Eingabe von stdin	•			
`semctl()`	Semaphor steuern		•		
`semget()`	Semaphor definieren		•		
`semop()`	Semaphor-Operationen		•		
`setenv()`	Umgebungsvariable definieren				•
`setuid()`	tatsächliche und effektive user-ID setzen			•	
`shmat()`	Shared memory einbinden		•		
`shmctl()`	Shared memory Kontrolloperationen		•		
`shmdt()`	Shared memory lösen		•		
`shmget()`	Shared memory definieren		•		
`signal()`	Signalreaktion definieren		•		
`sleep()`	Verzögerung			•	
`stat()`	Prüfen ob File existiert und Status holen	•			
`sync()`	Fileverwaltung und Datenblöcke aktualisieren	•			
`sysconf()`	Systemkonfigurationsdaten lesen				•
`time()`	Systemzeit holen				•
`times()`	Prozeßzeiten lesen			•	
`umask()`	Defaultwert Filezugriff definieren	•			
`unlink()`	Directoryeintrag löschen, File löschen	•			
`utime()`	Filezeiten setzen	•			
`wait()`	Warten auf Prozeßende bzw. Signal			•	
`write()`	File schreiben	•			

Übersicht über Prototypen von Systemcalls und Systemfunktionen

Prototypen von ausgewählten Systemcalls und Systemfunktionen

void	**_exit**(int)*status;*	<unistd.h> This function never returns
void	**access**(const char **pathname*, int *mode*);	<unistd.h> mode: R_OK, W_OK, X_OK, F_OK Returns: 0 if OK -1 on error
unsigned int	**alarm** (unsigned int *seconds);*	<*unistd.h*> Returns: 0 or #seconds until previously set alarm
int	**chdir**(const char **pathname*);	<unistd.h> Returns: 0 if OK, -1 on error
int	**chmod**(const char **pathname*, mode_t *mode*);	<sys/types.h> <sys/stat.h> *mode*: S_IS[UG]ID, S_ISVTX, S_I[RWX] (USR|GRP|OTH) Returns: 0 if OK, -1 on error Flags: siehe Tabelle
int	**chown**(const char **pathname, uid_t owner, gid_t group*);	<sys/types.h> <unistd.h> Returns; 0 if OK, -1 on error
int	**close**(int *filedes);*	<unistd.h> Returns; 0 if OK, -1 on error
int	**closedir**(DIR **dp);*	<sys/types.h> <dirent.h> Returns: 0 if OK, -1 on error
int	**creat**(const char **pathname*, mode_t *mode);*	<sys/types.h> <sys/stat.h> <fcntl.h> *mode*: S_IS[UG]ID, S_ISVTX, S_I[RWX] (USR|GRP|OTH) Returns: file descriptor opened for write-only if OK, -1 on error Flags: siehe Tabelle
char	***ctime**(const time_t **calptr);*	<time.h> Returns: pointer to null terminated string

```
int         dup(int filedes);
                                <unistd.h>
                                Returns: new file descriptor if OK, -1 on error
```

```
int         execl(const char *pathname, const char *arg0, ... /* (char *) 0 */ );
                                <unistd.h>
                                Returns: -1 on error, no return on success
```

```
int         execle(const char *pathname, const char *arg0, ... /* (char *) 0,
                    char *const envp[] */ );
                                <unistd.h>
                                Returns: -1 on error, no return on success
```

```
int         execlp(const char *filename, const char *arg0, ... /* (char *) 0 */ );
                                <unistd.h>
                                Returns: -1 on error, no return on success
```

```
int         execv(const char *pathname, char *const argv[]);
                                <unistd.h>
                                Returns: -1 on error, no return on success
```

```
int         execve(const char *pathname, char *const argv[], char *const envp[]);
                                <unistd.h>
                                Returns: -1 on error, no return on success
```

```
int         execvp(const char *filename, char *const argv[]);
                                <unistd.h>
                                Returns: -1 on error, no return on success
```

```
void        exit(int status);
                                <stdlib.h>
                                This function never returns
```

```
int         fclose(FILE *fp);
                                <stdio.h>
                                Returns: 0 if OK, EOF on error
```

```
int         fcntl(int filedes, int cmd, ... /* int arg */ );
                                <sys/types.h>
                                <unistd.h>
                                <fcntl.h>
                                cmd: F_DUPFD, F_GETED, F_SETFD, F_GETFL, F_SETFL
                                Returns: depends on cmd if OK, -1 on error
                                Flags: siehe Tabelle
```

```
FILE        *fopen(const char *pathname, const char *type);
                                <stdio.h>
                                type: "r", "w", "a", "r+", "w+", "a+",
                                Returns: file pointer if OK, NULL on error
```

```
pid_t       fork(void);
                                <sys/types.h>
                                <unistd.h>
                                Returns: 0 in child, process ID of child in parent, -1 on error
```

```
size_t      fread(void *ptr, size_t size, size_t nobj, FILE *fp);
                                <stdio.h>
                                Returns: number of objects read
```

```
int         fseek(FILE *fp, long offset, int whence);
                                <stdio.h>
                                whence: SEEK_SET, SEEK_CUR, SEEK_END
                                Returns: 0 if OK, nonzero on error
```

```
size_t      fwrite(const void *ptr, size_t size, size_t nobj, FILE *fp);
                                <stdio.h>
                                Returns: number of objects written
```

```
int         getc(FILE *fp);
                                <stdio.h>
                                Returns: next character if OK, EOF on end of file or error
```

```
int         getchar(void);
                                <stdio.h>
                                Returns: next character if OK, EOF on end of file or error
```

```
char        *getenv(const char *name);
                                <stdlib.h>
                                Returns: pionter to value associated with name, NULL if not found
```

```
int         getmsg(int filedes, struct strbuf *ctlptr, struct strbuf *dataptr, int *flagptr);
                                <stropts.h>
                                *flagptr: 0, RS_HIPRI
                                Returns: nonegative value if OK, -1 on error
```

```
pid         getpid(void);
                                <sys/types.h>
                                <unistd.h>
                                Returns: process ID of calling process
```

```
uid_t       getuid(void);
                                <sys/types.h>
                                <unistd.h>
                                Returns: real user ID of calling process
```

```
int         ioctl(int filedes, int request, ...);
                                <unistd.h>     /* SVR4 */
                                <sys/ioctl.h>  /* 4.3+BSD */
                                Returns: -1 on error, something else if OK
```

```
int         kill(pid_t pid, int signo);
                                <sys/types.h>
                                <signal.h>
                                Returns: 0 if OK, -1 on error
```

```
int         link(const char *existingpath, const char *newpath);
                                <unistd.h>
                                Returns: 0 if OK, -1 on error
```

```
off_t       lseek(int filedes, off_t offset, int whence);
                                <sys/types.h>
                                <unistd.h>
                                whence: SEEK_SET, SEEK_CUR, SEEK_END
                                Returns: new file offset if OK, -1 on error
```

```
void        *malloc(size_t size);
                                <stdlib.h>
                                Returns: nonnull pointer if OK, NULL on error
```

```
int         mkdir(const char *pathname, mode_t mode);
                                <sys/types.h>
                                <sys/stat.h>
                                mode: S_IS[UG]ID, S_ISVTX, S_I[RWX] (USR|GRP|OTH)
                                Returns: 0 if OK, -1 on error
```

```
int         mknod(const char *pathname, mode_t mode, dev_t dev);
                                <sys/types.h>
                                <sys/stat.h>
                                mode: S_IFMT: S_IFIFO, S_IFCHR, S_IFDIR, S_IFBLK, S_IFREG
                                Returns: 0 if OK, -1 on error
```

```
int         msgctl(int msqid, int cmd, struct msqid_ds *buf);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/msg.h>
                                cmd: IPC_STAT, IPC_SET, IPC_RMID
                                Returns: 0 if OK, -1 on error
```

```
int         msgget(key_t key, int flag);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/msg.h>
                                flag: 0, IPC_CREAT, IPC_exel
                                Returns: message queue ID if OK, -1 on error
```

```
int         msgrcv(int msqid, void *ptr, size_t nbytes, long type, int flag);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/msg.h>
                                flag: 0, IPC_NOWAIT, MSG_NOERROR
                                Returns: size of data portion of message if OK, -1 on error
```

```
int         msgsnd(int msqid, const void *ptr,  size_t  nbytes,  int  flag );
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/msg.h>
                                flag:  0, IPC_NOWAIT
                                Returns: 0 if OK, -1 on error
```

```
int         open(const char  *pathname, int oflag,  .../*,  mode_t  mode */ );
                                <sys/types.h>
                                <sys/stat.h>
                                <fcntl.h>
                                oflag:  O_RDONLY, O_WRONLY, O_RDWR;
                                     O_APPEND, O_CREAT, O_EXCL, O_TRUNC,
                                     O_NOCTTY, O_NONBLOCK, O_SYNC
                                mode: S_IS[UG]ID, S_ISVTX, S_I[RWX] (USR|GRP|OTH)
                                Returns: file descriptor if OK, -1 on error
                                Flags: siehe Tabelle
```

```
int         pause(void);
                                <unistd.h>
                                Returns: -1 with  errno  set to  EINTR
```

```
void        perror(const  char  *msg);
                                <stdio.h>
```

```
int         pipe(int  fildes[2] );
                                <unistd.h>
                                Returns: 0 if OK, -1 on error
```

```
int         poll(struct  pollfd  fdarray[ ],  unsigned  long  nfds, int timeout);
                                <stropts.h>
                                <poll.h>
                                Returns: count of ready descriptors, 0 on timeout, -1 on error
```

```
int         printf(const  char  *format,  ...);
                                <stdio.h>
                                Returns: # characters output if OK, negative value if output error
```

```
int         putc(int  c, FILE *fp);

                                <stdio.h>
                                Returns: c if OK, EOF on error
```

```
int         putchar(int  c);
                                <stdio.h>
                                Returns: c if OK, EOF  on error
```

```
int         putmsg(int fildes, const struct strbuf *ctlptr, const struct strbuf *dataptr,
                int flag);
                                <stropts.h>
                                flag: 0, RS_HIPRI
                                Returns: 0 if OK, -1 on error
```

```
int         putpmsg(int fildes, const struct strbuf *ctlptr, const struct strbuf
                    *dataptr, int band, int flag);
                                <stropts.h>
                                flag: 0, MSG_HIPRI, MSG_BAND
                                Returns: 0 if OK, -1 on error
```

```
ssize_t     read(int fildes, void *buff, size_t nbytes);
                                <unistd.h>
                                Returns: #bytes read if OK, 0 if end of file, -1 on error
```

```
int         rmdir(const char *pathname);
                                <unistd.h>
                                Returns: 0 if OK, -1 on error
```

```
int         scanf(const char *format, ...);
                                <stdio.h>
                                Returns: #input items assigned, EOF if input error or EOF before any
                                conversion
```

```
int         semctl(int semid, int semnum, int cmd, union semum arg);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/sem.h>
                                cmd: IPC_STAT, IPC_SET, IPC_RMID, GETPID,
                                     GETNCNT, GETZCNT, GETVAL, SETVAL, GETALL,
                                     SETALL
                                Returns: (depends on command)
```

```
int         semget(key_t key, int nsems, int flag);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/sem.h>
                                flag: 0, IPC_CREAT, IPC_EXCL
                                Returns: semaphore ID if OK, -1 on error
```

```
int         semop(int semid, struct sembuf semoparray[], size_t nops);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/sem.h>
                                Returns: 0 if OK, -1 on error
```

```
int         setenv(const char *name, const char *value, int rewrite);
                                <stdlib.h>
                                Returns: 0 if OK, nonzero on error
```

```
int         setuid(uid_t uid);
                                <sys/types.h>
                                <unistd.h>
                                Returns: 0 if OK, -1 on error
```

```
void        *shmat(int shmid, void *addr, int flag);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/shm.h>
                                flag:  0,  SHM_RND, SHM_RDONLY
                                Returns: pointer to shared memory segment if OK, -1 on error
```

```
int         shmctl(int  shmid, int cmd, struct shmid_ds  *buf);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/shm.h>
                                cmd:  IPC_STAT, IPC_SET, IPC_RMID, SHM_LOCK,
                                      SHM_UNLOCK
                                Returns: 0 if OK, -1 on error
```

```
int         shmdt(void *addr);
                                <sys/types.h
                                <sys/ipc.h>
                                <sys/shm.h>
                                Returns: 0 if OK, -1 on error
```

```
int         shmget(key_t key, int size, int flag);
                                <sys/types.h>
                                <sys/ipc.h>
                                <sys/shm.h>
                                flag:  0,  IPC_CREAT, IPC_EXCL
                                Returns: shared memory ID if OK, -1 on error
```

```
void        (*signal(int  signo, void  (*func) (int))) (int);
                                <signal.h>
                                Returns: previous disposition of signal, SIG_ERR on error
```

```
unsigned
int         sleep(unsigned int seconds);
                                <unistd.h>
                                Returns: 0 or number of unslept seconds
```

```
int         stat(const  char  *pathname, struct  stat  *buf);
                                <sys/types.h>
                                <sys/stat.h>
                                Returns: 0 if OK, -1 on error
```

```
int         symlink(const  char  *actualpath, const char  *sympath);
                                <unistd.h>
                                Returns: 0 if OK, -1 on error
```

```
int         sync(void);
                                <unistd.h>
                                Returns: 0 if OK, -1 on error
```

`long`	**`sysconf`**`(int` *name*`);`
	`<unistd.h>`
	name:
	`_SC_ARG_MAX, _SC_CHILD_MAX, _SC_CLM_TCK,`
	`SC_NGROUPS_MAX, _SC_OPEN_MAX, _SC_PASS_MAX,`
	`SC_STREAM_MAX, _SC_TZNAME_MAX, SC_JOB_CONTROL`
	`_SC_SAVED_IDS, _SC_VERSION, _SC_XOPEN_VERSION`
	Returns: corresponding value if OK, -1 on error

`time_t`	**`time`**`(time_t` *`*calptr`*`);`
	`<time.h>`
	Returns: value of time if OK, -1 on error

`clock_t`	**`times`**`(struct` *`tms *buf`*`);`
	`<sys/times.h>`
	Returns: elapsed wall clock time in clock ticks if OK, -1 on error

`mode_t`	**`umask`**`(mode_t` *`cmask`*`);`
	`<sys/types.h>`
	`<sys/stat.h>`
	Returns: previous file mode creation mask

`int`	**`unlink`**`(const char` *`*pathname`*`);`
	`<unistd.h>`
	Returns: 0 if OK, -1 on error

`int`	**`utime`**`(const char` *`*pathname`*`, const struct utimbuf` *`*times`*`);`
	`<sys/types.h>`
	`<utime.h>`
	Returns: 0 if OK, -1 on error

`pid_t`	**`wait`**`(int` *`*statloc`*`);`
	`<sys/types.h>`
	`<sys/wait.h>`
	Returns: process ID if OK, 0, or -1 on error

`ssize_t`	**`write`**`(int` *`filedes`*`, const void` *`*buff`*`, size_t` *`nbytes`*`);`
	`<unistd.h>`
	Returns: #bytes written if OK, -1 on error

Header		*Standards*			*Implementierung*		*Bedeutung*
		ANSI C	POSIX.1	XPG3	SVR4	4.3+BSD	
`<assert.h>`		•			•	•	verify program assertion
`<cpio.h>`			•		•		cpio archive values
`<ctype.h>`		•			•	•	character types
`<dirent.h>`			•	•	•	•	directory entries
`<errno.h>`	*	•			•	•	error codes
`<fcntl.h>`			•	•	•	•	file control
`<float.h>`	*	•			•	•	floating point constants
`<ftw.h>`				•	•		file tree walking
`<grp.h>`			•	•	•	•	group file
`<langinfo.h>`				•	•		language information constants
`<limits.h>`	*	•			•	•	implementation constants
`<locale.h>`		•			•	•	locale categories
`<math.h>`		•			•	•	mathematical constants
`<nl_types.h>`				•	•		message catalogs
`<pwd.h>`			•	•	•	•	password file
`<regex.h>`				•	•	•	regular expressions
`<search.h>`				•	•		search tables
`<setjmp.h>`		•			•	•	nonlocal goto
`<signal.h>`		•			•	•	signals
`<stdarg.h>`		•			•	•	variable argument lists
`<stddef.h>`	*	•			•	•	standard definition representation
`<stdio.h>`		•			•	•	standard I/O library
`<stdlib.h>`		•			•	•	utility functions
`<string.h>`		•			•	•	string operations
`<tar.h>`			•		•		tar archive values
`<termios.h>`			•	•	•	•	terminal I/O
`<time.h>`		•			•	•	time and date
`<ulimit.h>`				•	•		user limits
`<unistd.h>`			•	•	•	•	symbolic constants
`<utime.h>`			•	•	•	•	file times
`<sys/ipc.h>`				•	•	•	IPC
`<sys/msg.h>`				•	•		message queues
`<sys/sem.h>`				•	•		semaphores
`<sys/shm.h>`				•	•	•	shared memory
`<sys/stat.h>`			•	•	•	•	file status
`<sys/times.h>`			•	•	•	•	process times
`<sys/types.h>`			•	•	•	•	primitive system data types
`<sys/utsname.h>`			•	•	•		system name
`<sys/wait.h>`			•	•	•	•	process control

* nur Datentypen und Makrodefinitionen

Header in `/usr/include/...`

Header File	*Namensraum*
<ctype.h>	All symbols starting with is or to
<dirent.h>	All symbols starting with d_
<errno.h>	All symbols starting with E followed by any uppercase letter or a digit
<fcntl.h>	All symbols starting with l_ Symbols starting with F_, O_, or S_ may be used if #undef is done for each symbol prior to any other use
<grp.h>	All symbols starting with gr_
<limits.h>	All symbols starting with _MAX
<locale.h>	All symbols starting with LC_ followed by an uppercase letter.
<math.h>	The names of existing math functions followed by an f or an l..
<pwd.h>	All symbols starting with pw_.
<signal.h>	All symbols starting with sa_. Symobls starting with SIG or SA_ may be used if an #undef is done for each symbol prior to any other use.
<string.h>	All symbols starting with mem, str, or wcs.
<sys/stat.h>	All symbols starting with st_. Symbols starting with S_ may be used if an #undef is done for each symbol prior to any other use.
<sys/times.h>	All symbols starting with tms_.
<termios.h>	All symbols starting with c_. Symbols starting with V, I, O or TC may be used if an #undef is done for each symbol prior to any other use. Symbols starting with B followed by a digit may be used if an #undef is done for each symbol prior to any other use.
Any POSIX header	All symbols ending with _t.

Reservierter Namensraum Standard C

Headerfile	*Namensraum*	*Typ*		*Reserviert*	
		declared [1]	#defined [2]	POSIX.1	POSIX.4
<aio.h>	aio_	•			•
	lio_	•			•
	AIO_		•		•
	LIO_		•		•
<dirent.h>	d_	•		•	
<fcntl.h>	l_	•		•	
	F_		•	•	
	O_		•	•	
	S_		•	•	
<grp.h>	gr_	•		•	
<limits.h>	_MAX (suffix)		•	•	
<locale.h>	LC_[A-Z]		•	•	
<mqueue.h>	mq_	•			•
	MQ_		•		•
<pwd.h>	pw_	•		•	
<sched.h>	sched_	•			•
	SCHED_		•		•
<semaphore.h>	sem_	•			•
	SEM_		•		•
<signal.h>	sa_	•		•	
	si_	•			•
	sigev_	•			•
	sival_	•			•
	SA_		•	•	
	SI_		•		•
	SIG_		•	•	
<termios.h>	c_	•		•	
	V		•	•	
	I		•	•	
	O		•	•	
	TC		•	•	
	B[0-9]		•	•	
<sys/mman.h>	shm_	•			•
	MAP_		•		•
	MCL_		•		•
	MS_		•		•
	PROT_		•		•
<sys/stat.h>	st_	•		•	
	S_		•	•	
<sys/times.h>	tms_	•		•	
any	_t (suffix)	• (types)		•	
	_	•		•	

1: declared: functions, variables ...

2 : #defined: constants, macros

Reservierter Namensraum POSIX

Typ	*Bedeutung*
caddr_t	core address
clock_t	counter of clock ticks (processtime)
comp_t	compressed clock ticks
dev_t	device numbers (major and minor)
fd_set	file descriptor sets
fpos_t	file position
gid_t	numeric group IDs
ino_t	i-node numbers
mode_t	file type, file creation mode
nlink_t	linkcounts for directory entries
off_t	file sizes and offsets (signed) (lseek())
pid_t	process IDs and process group IDs (signed)
ptrdiff_t	result of subtracting two pointers (signed)
rlim_t	resource limits
sig_atomic_t	data type that can be accessed atomically
sigset_t	signal set
size_t	sizes of objects (such as strings) (unsigned)
ssize_t	functions that return a count of bytes (signed) (read(), write())
time_t	counterof seconds of calendar time
uid_t	numeric user IDs
wchar_t	can represent all distinct character codes

Typen der Systemdaten in <sys/types.h>

Variable	*Standards*		*Implementierung*		*Bedeutung*
	POSIX.1	XPG3	SVR4	4.3+BSD	
HOME	•	•	•	•	home directory
LANG	•	•	•		name of language
LC_ALL	•	•	•		name of locale
LC_COLLATE	•	•	•		name of locale for collation
LC_CTYPE	•	•	•		name of locale for character classification
LC_MONETARY	•	•	•		name of locale for monetary editing
LC_NUMERIC	•	•	•		name of locale for numeric editing
LC_TIME	•	•	•		name of locale date/ time formatting
LOGNAME	•	•	•	•	login name
NLSPATH		•	•		sequence of templates for message catalogs
PATH	•	•	•	•	list of path prefixes to search for executable file
TERM	•	•	•	•	terminal type
TZ	•	•	•	•	time zone information

Umgebungsvariable

```
#include   <unistd.h>

int execl  (const char *pathname, const char *arglist, ...);
int execlp (const char *filename,  const char *arglist, ...);
int execle (const char *pathname, const char *arglist, ..., char *const envp[]);
int execv  (const char *pathname, char *const argv[]);
int execvp (const char *filename,  char *const argv[] ;
int execve (const char *pathname, char *const argv[] ; char *const envp[]);
```

Return: -1 on error, no return on success

Aufrufvarianten `exec...()`

call	*call-Argumente*					
	Suchpfad		*Parameterübergabe*		*Umgebung*	
	pathname	*filename*	`Arg list`	*argv* []	`environ`	*envp* []
execl()	•		•		•	
execlp()		•	•		•	
execle()	•		•			•
execv()	•			•	•	
execvp()		•		•	•	
execve()	•			•		•
Namensbildung mit	-	p	l	v	-	e
Charakteristik	komplett	PATH	Liste	Vektor	automatisch	extra

Parameterunterschiede `exec...()`

- process ID and parent process ID (PID, PPID)
- real user ID and real group ID (UID, GID)
- supplementary group IDs
- process group ID
- session ID
- controlling terminal (tty ...)
- time left until alarm clock
- current working directory
- root directory
- file mode creation mask
- file locks
- process signal mask
- pending signals
- resource limits
- `tms¸utime`, `tms¸stime`, `tms¸cutime` and `tms¸ustime` values

Vererbte Eigenschaften von Elternprozeß an Kindprozeß nach Aufruf von `exec...()`

Macro	*Filetyp*	*Symbol*	*Bedeutung*
`S_ISREG()`	regular file	-	plain file/ Standardfile
`S_ISDIR()`	directory file	d	directory/ Verzeichnis
`S_ISCHR()`	character special file	c	character/ Zeichenkanal
`S_ISBLK()`	block special file	b	block/ Blockkanal
`S_ISFIFO()`	pipe or FIFO	p	pipe
`S_ISLNK()`	symbolic link (not in POSIX.1 or SVR4)	l	link
`S_ISSOCK()`	socket (not in POSIX.1 or SVR4)	s	socket

Filetyp-Macros in `<sys/stat.h>`

Flag	*Bedeutung*	*Wirkung auf Standardfile*	*Wirkung auf Directory*	*Position*
`S_ISUID`	set-user-ID	set effective user ID on execution	(not used)	1-- --- --- ---
`S_ISGID`	set-group-ID	if group-execute set then set effective group ID on execution; otherwise enable mandatory record locking	set group ID of new files created in directory to group ID of directory	-1- --- --- ---
`S_ISVTX`	sticky bit	save program text in swap area (if supported)	restrict removal and remaining of files in directory	--1 --- --- ---
`S_IRUSR`	user-read	user permission to read file	user permission to read directory entries	r-- --- ---
`S_IWUSR`	user-write	user permission to write file	user permission to remove and create files in directory	-w- --- ---
`S_IXUSR`	user-execute	user permission to execute file	user permission to search for given pathname in directory	--x --- ---
`S_IRGRP`	group-read	group permission to read file	group permission to read directory entries	--- r-- ---
`S_IWGRP`	group-write	group permission to write file	group permission to remove and create files in directory	--- -w- ---
`S_IXGRP`	group-execute	group permission to execute file	group permission to search for given pathname in directory	--- --x ---
`S_IROTH`	other-read	other permission to read file	other permission to read directory entries	--- --- r--
`S_IWOTH`	other-write	other permission to write file	other permission to remove and create files in directory	--- --- -w-
`S_IXOTH`	other-execute	other permission to execute file	other permission to search for given pathname in directory	--- --- --x

Filezugriffsbits (file access permission bits)

Ein Flag obligatorisch:	`O_RDONLY`	Open file for reading only
	`O_WRONLY`	Open file for writing only
	`O_RDWR`	Open file for both reading and writing
Zusätzliche Flags logisch "OR":	`O_APPEND`	All writes occur at end of file
	`O_NDELAY`	Old-style "nodelay" mode (system V)
	`O_NONBLOCK`	New-style "nonblocking" mode (POSIX)
	`O_SYNC`	Make all writes synchronous
	`O_CREAT`	If the file does not exist, create it
	`O_EXCL`	With `O_CREATE`, fail if the file exists
	`O_TRUNC`	Truncate the file to zero length
	`O_NOCTTY`	Do not allocate a controlling terminal

Flags für Filezugriff

Name	*Nr.*	*Default*	*Beschreibung*	*Taste (ASCII)*
SIGHUP	1	*Exit*	Hangup, to all associated processes with terminal	
SIGINT	2	*Exit*	Interrupt, to all associated processes with terminal	CTRL - C (ETX)
SIGQUIT	3	*Core*	Quit	CTRL - \ (FS)
SIGILL	4	*Core*	Illegal Instruction	
SIGTRAP	5	*Core*	Trace/Breakpoint Trap	
SIGABRT	6	*Core*	Abort (abort())	
SIGEMT	7	*Core*	Emulation Trap	
SIGFPE	8	*Core*	Arithmetic Exception	
SIGKILL	9	*Exit*	Killed (unconditionally)	
SIGBUS	10	*Core*	Bus Error	
SIGSEGV	11	*Core*	Segmentation Fault	
SIGSYS	12	*Core*	Bad System Call	
SIGPIPE	13	*Exit*	Broken Pipe	
SIGALRM	14	*Exit*	Alarm Clock (alarm())	
SIGTERM	15	*Exit*	Terminated (nicely)	
SIGUSR1	16	*Exit*	User Signal 1	
SIGUSR2	17	*Exit*	User Signal 2	
SIGCHLD	18	*Ignore*	Child Status Changed	
SIGPWR	19	*Ignore*	Power Fail/ Restart	
SIGWINCH	20	*Ignore*	Window Size Change	
SIGURG	21	*Ignore*	Urgent Socket Condition	
SIGPOLL	22	*Exit*	Pollable Event (poll())	
SIGSTOP	23	*Stop*	Stopped (signal) - CONT continues	
SIGTSTP	24	*Stop*	Stopped (user) Foreground processes	CTRL - Z (SUB)
SIGCONT	25	*Ignore*	Continued	
SIGTTIN	26	*Stop*	Stopped (tty input)	
SIGTTOU	27	*Stop*	Stopped (tty output)	
SIGVTALRM	28	*Exit*	Virtual Timer Expired (setitimer())	
SIGPROF	29	*Exit*	Profiling Timer Expired	
SIGXCPU	30	*Core*	CPU time limit exceeded (setrlimit())	
SIGXFSZ	31	*Core*	File size limit exceeded (setrlimit())	
SIGWAITING	32	*Ignore*	Process's LWPs are blocked	
SIGLWP	33	*Ignore*	Special signal used by thread library	
SIGFREEZE	34	*Ignore*	Check point Freeze	
SIGTHAW	35	*Ignore*	Check point Thaw	
SIGRTMIN	*	*Exit*	First real time signal	
SIGRTMIN+1	*	*Exit*	Second real time signal	
...			...	
SIGRTMAX-1	*	*Exit*	Second-to-last real time signal	
SIGRTMAX	*	*Exit*	Last real time signal	

Signalübersicht <signal.h> (Konkrete Implementierung beachten: Solaris 2.5)

Core: Abbruch + Speicherabzug
Exit: Abbruch
Ignore: ohne Reaktion
Stop: Halt

Typ	*Funktion*	*Sockets (BSD, SVR4)*	*TLI (SVR4)*	*Messages*	*FIFOs*
Server:	Speicherplatz zuweisen		`t_alloc()`		
	Erzeugung eines Endpunktes	`socket()`	`t_open()`	`msgget()`	`mknod()` `open()`
	Binden einer Adresse	`bind()`	`t_bind()`		
	Bestimmung einer Warteschlange	`listen()`			
	Warten auf eine Verbindung	`accept()`	`t_listen()`		

	für Verbindung eine neuen Fileskriptor laden		`t_open()`
			`t_bind()`
			`t_accept()`		

Client:	Speicherplatz zuweisen		`t_alloc()`		
	Erzeugung eines Endpunktes	`socket()`	`t_open()`	`msgget()`	`open()`
	Binden einer Adresse	`bind()`	`t_bind()`		
	mit einem Server Verbinden	`connect()`	`t_connect()`		

	Daten übertragen	`read()`
		`write()`
		`recv()`
		`send()`
			`read()`
			`write()`
			`t_rcv()`
			`t_snd()`
				`msgcrv()`
				`msgsnd()`
					`read()`
					`write()`

	Datagramme	`recvfrom()` `sendto()`	`t_rcvudata()` `t_sndudata()`		

	Beenden	`close()`
		`shutdown()`
			`t_close()`
			`t_sndrel()`
			`t_sbddis()`
				`msgctl()`	`close()`
					`unlink()`

Funktionsvergleich Sockets, Transport Layer Interface TLI, Messages und FIFOs

Program management and analysis tools

`ar(1)`	library management
`as(1)`	translate assembly code to machine code
`cpp(1)`	C preprocessor
`cc(1), gcc`	C compiler
`CC(1)`	C++ compiler
`cb(1)`	C program beautifier, formatter
`lint(1)`	C program verifier, checker
`ld(1)`	invoke the link (load) editor
`make(1)`	generate programming system, maintain groups of programs

Profiling and debugging tools

`gprof(1)`	display call graph profile data
`prof(1)`	display profile data, time consumption
`monitor(3C)`	prepare execution profile
`adb(1)`	absolute general-purpose debugger
`cdb, xdb(1)`	symbolic debugger (C, C++; FORTRAN, Pascal)
`dbx(1)`	standard debugger
`sar(1)`	system activity reporter
`strip(1)`	remove extra code after debugging and profiling from object code
`time, timex(1)`	prints elapsed time real-user-sys
`truss(1)`	trace system calls and signals (SVR4)

System tools

`file(1)`	determine filetype
`cxref(1)`	C program cross-reference
`nm(1)`	output symbol table
`od(1)`	output object or binary file in character-, octal-, hex-format
`SCCS`	source code control system - release management (admin, get, delta, rmdel, prs)
`size(1)`	measurement program size
`strings(1)`	find printable strings in object or binary file

Miscellaneous

`matherr(3M)`	trap math errors
`fpgetround(3M)`	floating point mode control
`crt0(3)`	execution startup routine
`end(3C)`	symbol of the last locations in program

Werkzeuge zur Programmentwicklung

Source Code Portability

Several criteria will be used to determine the portability of the source code:

- A portable source file cannot access machine or architecture specific data, tables, vectors, ect. except by a defined interface.
- The file cannot contain machine-specific `#ifdefs`, machine specific variable usage, nor machine specific knowledge of architecture.
- Any defined `#ifdefs` should be based on characteristics and not machine names.

Portable source must:

- Compile cleanly on all system architecture under consideration.
- Be free of any hardware dependencies, such as assuming the address of physical memory.
- Be free of any architectural layouts, such as directions of stack growth, or a particular set of alignment constraints.
- Not contain any machine instructions, such as assembler language statements and functions.
- Be free of dependencies on CPU characteristics, such as byte ordering, unless all possibilities are accounted for.
- Not perform memory mapping. Memory mapping, such as physical virtual, is functionality closely tied to a particular architecture. Portable files will not make assumptions about, nor perform such conversions unless by way of a portable interface.
- Be free of bogus `#include` statements. A source file is allowed to `#include` other source files if and only if the other files span across the source code level. For example, if a file is defined at the platform level, but not defined for all platforms of the family, the family source code should not reference that file.

Below is a checklist containing some of more settle points for writing portable code. This list is by no means exhaustive.

- Write ANSI C code. In particular, use function prototypes since they can assist in locating assumptions about type compatibility that may not be true when the code is ported.
- Do not presume the sign of the type `"char"`. Define a character explicitly `"unsigned"` or `"signed"` if the signedness matter.
- Explicitly type integral constants (or cast them) when their size is important.
- Do not assume that pointers to data objects of different types have the same size and representation.
- Do not depend on the behaviour for right shift of negative numbers.
- Do not rely on the order of evaluation of arguments in a function call.
- Do not use constructs which rely on the number of bits in an integer type. For example, use `"x & ~077"` instead of `"x & 0177700"`.
- Do not rely on implementation to have a particular auto variable allocation scheme. In particular, do not rely auto variables being allocated on the stack in exactly the same order in which they are declared.
- Do not assume the ordering of bit-fields is left-to-right or right-to-left.
- Do not assume that an enumerated type is represented as an `"int"`, it may be any integral type. Note that enumeration members are defined in ANSI C to be type `"int"`.
- Do not attempt to modify a "const" object by means of a pointer to a type without a `"const"` attribute.
- Run `lint()` to find portability problems.

Quelle: CDE-Consortium, 1993.