Prüfungsvorbereitung 1

Modul Systemprogrammierung (syspr)

Simon Wächter

2017

# Setcard

## Speicheroperationen

* **void \*malloc(int n x sizeof(item)**: Alloziert n Mal die Grösse von item an Bytes und gibt diesen zurück
* **void free(void \*ptr)**: Löscht den allozierten Speicher
* **void \*memcpy(void \*desptr, void \*srcptr, size\_t length)**: Kopiert den Eingangsbuffer in den Ausgangsbuffer, sofern die Länge nicht überschritten wird. Zurückgeben wird der Ausgangsbuffer

## Stringoperationen

* **char \*strcpy(char \*destptr, char \*srcptr)**: Kopiert den Eingangsstring in den Ausgangsstring. Zurückgegeben wird der Ausgangsstring
* **size\_t strlen(char \*ptr)**: Gibt die Länge eines Strings zurück
* **int sprintf(char \*destptr, char \*specifiers, va\_list arguments)**: Formatiert einen String und schreibt ihn in den Ausgangsbuffer

## Terminaloperationen

* **int scanf(char \*specifiers, va\_list arguments)**: Liest Werte von der Standardeingabe
* **int printf(char \*specifiers, va\_list arguments)**: Schreibt Werte auf die Standardausgabe

## Dateioperationen

* **FILE \*fopen(char \*path, char \*mode)**: Öffnet eine Datei im Modus (r=Read, w+=Read) und gibt ein Handle zurück
* **int fclose(FILE \*handle)**: Schliesst einen Handle
* **int fgetc(FILE \*handle)**: Liest einen Chrakter als Integer aus der Datei. Als Ende wird EOF zurückgegeben
* **int fputc(FILE \*handle, char characer)**: Schreibt einen Charakter in die Datei
* **int rename(char \*oldpath, char \*newpath)**: Benennt eine Datei um
* **FILE \*dup(FILE \*handle)**: Dupliziert einen Handle

## Verzeichnisoperationen

* **DIR \*opendir(char \*path)**: Öffnet ein Verzeichnis und gibt ein Handle zurück
* **dirent \*readdir(DIR \*handle)**: Gibt als Iterator alle Verzeichniseinträge als dirent oder aber als NULL zurück
* **int closedir(DIR \*handle)**: Schliesst einen handle
* **int mkdir(char \*path, mode\_t mode)**: Erstellt ein Verzeichnis mit dem Mode (z.B. 0777)
* **int rmdir(char \*path)**: Löscht ein Verzeichnis
* **struct dirent**: Verzeichnisstruktur mit folgenden Elementen:
  + d\_name: Name des Verzeichnisses oder der Datei
  + d\_type: Typ des Verzeichnisses (z.B. DT\_DIR)

## Prozessoperationen

* **void exit(int status)**: Terminiert den Prozess mit dem angebenen Status
* **pid\_t fork()**: Forkt einen Prozess und gibt für den Parent die PID des Childs und für das Child selber 0 zurück
* **int execl(char \*path, char\*arg, char \*env[])**: Führt einen neuen Prozess in der aktuellen prozesshülle aus
* **pid\_t getpid()**: Gibt die aktuelle Prozessnummer zurück
* **pid\_t getppid()**: Gibt die aktuelle Prozessnummer des Parent zurück
* **int kill(pid\_t pid)**: Terminiert einen Prozess
* **unsigned int sleep(int seconds)**: Lässt einen Prozess schlafen
* **pid\_t wait(0)**: Wartet auf den nächsten Prozess
* **pid\_t waitpid(pid\_t pid)**: Wartet auf einen bestimmten Prozess

## Pipeoperationen (Unnamed)

* **int pipe(int[2] pipe)**: Öffnet eine Pipe (0=Reader, 1=Writer)
* **int close(int pipeentry)**: Schliesst eine Ende
* **ssize\_t read(int pipeentry, char \*buffer, size\_t buffersize)**: Liest von der Pipe
* **ssize\_t write(int pipeentry, char \*buffer, size\_t buffersize)**: Schreibt auf die Pipe

## Pipeoperationen (Named)

* **int mknod(char \*path, mode\_t mode, rdev\_t device\_id)**: Erstellt eine normale Datei oder Pipe
* **int mkfifo(char \*path, mode\_t mode)**: Erstellt eine FIFO Pipe mit dem Mode (z.B. 0777)
* **int open(char \*path, int oflag)**: Öffnet eine Pipe in Mode (z.B. O\_RDONLY oder O\_WRONLY)
* **int close(int pipeentry)**: Schliesst eine Ende
* **ssize\_t read(int pipeentry, char \*buffer, size\_t buffersize)**: Liest von der Pipe
* **ssize\_t write(int pipeentry, char \*buffer, size\_t buffersize)**: Schreibt auf die Pipe

## Signaloperationen

* **void (\*signal(int sig, void (\*func)(int)))(int)**: Installiert eine neuen Signalhandler und gibt den alten zurück. Gängige Signale sind SIGABRT, SIGFPE, SIGINT oder SIGALARM
* **int raise(int signal)**: generiert ein Signal und schickt es dem Betriebssystem
* **unsigned int alarm(unsigned int seconds)**: Generiert nach der angegebenen Zeit ein Alarmsignal

## Sprungoperationen

* **int setjmp(jmp\_buf environment)**: Setzt den Buffer auf das lokale Environment und initialisiert dieses für einen Sprung. Wird die Funktion direkt aufgerufen, wird 0 zurückgegeben. Wird die Funktion nach einem Jump via longjmp aufgerufen, wird ein Wert ungleich 0 zurückgegeben
* **longjmp(jmp\_buf environment, int value)**: Führt einen Sprung basierend auf dem Environment aus und übergibt den angegeben Wert an die setjmp Funktion (z.B. Errorcode)
* **struct jmp\_buf**: Liefert ein Environment für die Jumps

## Fehlerbehandlung

* **void perror(char \*message)**: Gibt die angegebene nachricht mit dem letzten Fehler (erno) auf der Standardausgabe aus