

27 Überblick über die 6. Übung

Überblick über die 6. Übung

- Aufgabe 2: Sortieren mittels qsort
- Hinweise zur 4. Aufgabe (fork, exec)

Ü-SP1

Übungen zur Systemprogrammierung 1

© Michael Golm, Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2002

2002-11-13 11.47

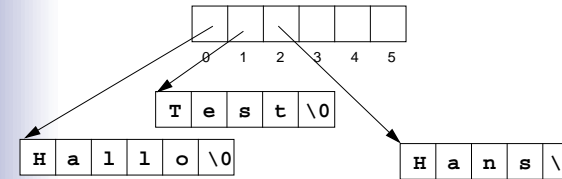
154

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Aufgabe 2: Sortieren mittels qsort (Fortsetzung)

28.2 wsort - Datenstrukturen (2. Möglichkeit)

- Array von Zeigern auf Zeichenketten



- Vorteile:
 - ◆ schnelles Sortieren, da nur Zeiger vertauscht werden müssen
 - ◆ Zeichenketten können beliebig lang sein
 - ◆ sparsame Speichernutzung

Ü-SP1

Übungen zur Systemprogrammierung 1

© Michael Golm, Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2002

2002-11-13 11.47

156

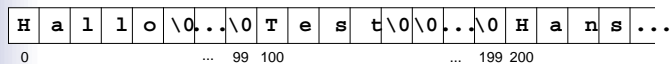
Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

28 Aufgabe 2: Sortieren mittels qsort (Fortsetzung)

Aufgabe 2: Sortieren mittels qsort (Fortsetzung)

28.1 wsort - Datenstrukturen (1. Möglichkeit)

- Array von Zeichenketten



- Vorteile:
 - ◆ einfach
- Nachteile:
 - ◆ hoher Kopieraufwand
 - ◆ Maximale Länge der Worte muß bekannt sein
 - ◆ Verschwendung von Speicherplatz

Ü-SP1

Übungen zur Systemprogrammierung 1

© Michael Golm, Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2002

2002-11-13 11.47

155

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Aufgabe 2: Sortieren mittels qsort (Fortsetzung)

28.3 Speicherverwaltung

- Berechnung des Array-Speicherbedarfs
 - ◆ bei Lösung 1: Anzahl der Wörter * 100 * sizeof(char)
 - ◆ bei Lösung 2: Anzahl der Wörter * sizeof(char*)
- realloc:
 - ◆ Anzahl der zu lesenden Worte ist unbekannt
 - ◆ Array muß vergrößert werden: realloc
 - ◆ Bei Vergrößerung sollte man aus Effizienzgründen nicht nur Platz für ein neues Wort (Lösungsvariante 1) bzw. einen neuen Zeiger (Lösungsvariante 2) besorgen, sondern für mehrere.
 - ◆ Achtung: realloc kopiert möglicherweise das Array (teuer)
- Speicher sollte wieder freigegeben werden
 - ◆ bei Lösung 1: Array freigeben
 - ◆ bei Lösung 2: zuerst Wörter freigeben, dann Zeiger-Array freigeben

Ü-SP1

Übungen zur Systemprogrammierung 1

© Michael Golm, Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2002

2002-11-13 11.47

157

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

28.4 Vergleichsfunktion

- Problem: qsort erwartet folgenden Funktionstyp:

```
int (*compare) (const void *, const void *)
```

- Lösung: "casten"

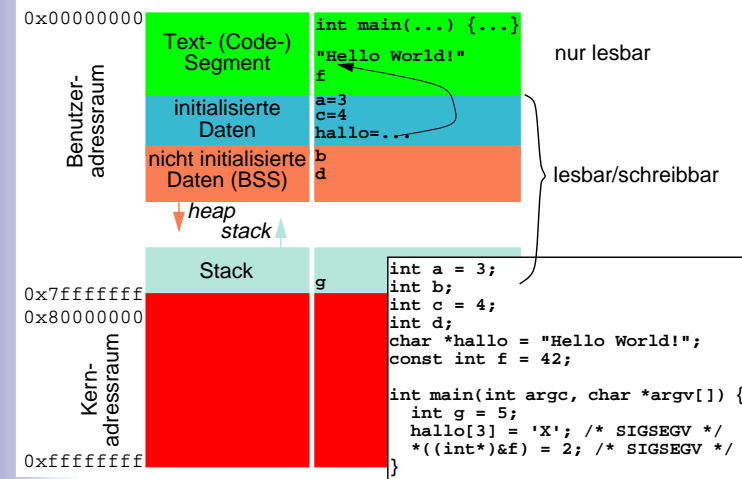
- ◆ innerhalb der Funktion, z.B. (Feld vom Typ char **):

```
int compare(const void *a, const void *b) {
    return strcmp(*(char **)a, *(char **)b);
}
```

- ◆ beim qsort-Aufruf:

```
int compare(char **a, char **b);
...
qsort( field, nel, sizeof(char *),
      (int (*)(const void *, const void *))compare);
```

29.1 Aufbau der Daten eines Prozesses



29 Hinweise zur 4. Aufgabe

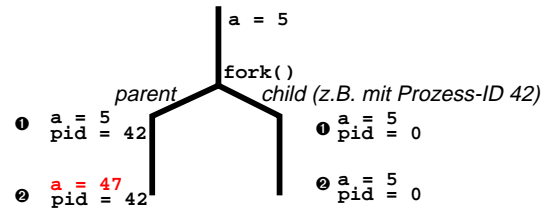
- Prozesse
- fork, exec
- exit
- wait

29.1 fork

- Vererbung von
 - ◆ Datensegment (neue Kopie, gleiche Daten)
 - ◆ Stacksegment (neue Kopie, gleiche Daten)
 - ◆ Textsegment (gemeinsam genutzt, da nur lesbar)
 - ◆ Filedeskriptoren
 - ◆ Arbeitsverzeichnis
 - ◆ Benutzer- und Gruppen-ID (uid, gid)
 - ◆ Umgebungsvariablen
 - ◆ Signalbehandlung
 - ◆ ...
- Neu:
 - ◆ Prozess-ID

29.2 fork

```
int a;
a = 5;
pid_t pid = fork();
❶
a += pid; ❷
if (pid == 0) {
    ...
} else {
    ...
}
```



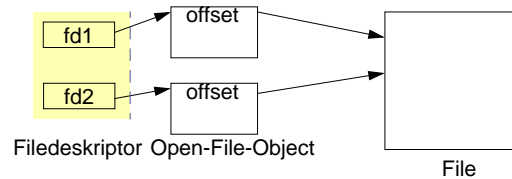
29.4 exec

- Lädt Programm zur Ausführung in den aktuellen Prozess
- ersetzt Text-, Daten- und Stacksegment
- behält: Filedeskriptoren, Arbeitsverzeichnis, ...
- Aufrufparameter:
 - ◆ Dateiname des neuen Programmes (z.B. `"/bin/cp"`)
 - ◆ Argumente, die der `main`-Funktion des neuen Programms übergeben werden (z.B. `"cp"`, `"/etc/passwd"`, `"/tmp/passwd"`)
 - ◆ evtl. Umgebungsvariablen
- Beispiel

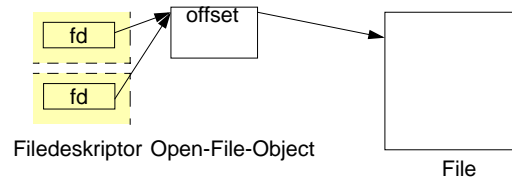
```
execl("/bin/cp", "cp", "/etc/passwd", "/tmp/passwd", NULL);
```

29.3 fork und Filedeskriptoren

- erneutes Öffnen eines Files



- bei fork werden FD vererbt, aber Files werden nicht neu geöffnet!



29.4 exec Varianten

- mit Angabe des vollen Pfads der Programm-Datei in `path`

```
int execl(const char *path, const char *arg0, ...,
          const char *argn, char * /*NULL*/);
```

```
int execv(const char *path, char *const argv[]);
```

- mit Umgebungsvariablen in `envp`

```
int execl(const char *path, char *const arg0, ... , const char
          *argn, char * /*NULL*/, char *const envp[]);
```

```
int execve(const char *path, char *const argv[], char *const
           envp[]);
```

- zum Suchen von `file` wird die Umgebungsvariable `PATH` verwendet

```
int execlp(const char *file, const char *arg0, ..., const char
           *argn, char * /*NULL*/);
```

```
int execvp(const char *file, char *const argv[]);
```

29.5 **exit**

- beendet aktuellen Prozess
- gibt alle Ressourcen frei, die der Prozess belegt hat, z.B.
 - ◆ Speicher
 - ◆ Filedeskriptoren (schließt alle offenen Files)
 - ◆ Kerndaten, die für die Prozessverwaltung verwendet wurden
- Prozess geht in den *Zombie*-Zustand über
 - ◆ ermöglicht es dem Vater auf den Tod des Kindes zu reagieren (wait)

29.6 **wait**

- warten auf Statusinformationen von Kind-Prozessen (Rückgabe: PID)
 - ◆ `wait(int *status)`
 - ◆ `waitpid(pid_t pid, int *status, int options)`
- Beispiel:

```
int main(int argc, char *argv[]) {
    int pid;
    if ((pid=fork()) > 0) {
        /* parent */
        int status;
        wait(&status); /* ... Fehlerabfrage */
        printf("Kindstatus: %d", status);
    } else if (pid == 0) {
        /* child */
        execl("/bin/cp", "cp", "/etc/passwd", "/tmp/passwd", 0);
        /* diese Stelle wird nur im Fehlerfall erreicht */
    } else {
        /* pid == -1 --> Fehler bei fork */
    }
}
```