F 4. Übung

F.1 Überblick

- Aufgabe 2: qsort - Fortsetzung
- Infos zur Aufgabe 4: fork, exec

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

F.2 Aufgabe 2: Sortieren mittels qsort (Fortsetzung)

1 wsort - Datenstrukturen (1. Möglichkeit)

- Array von Zeichenketten

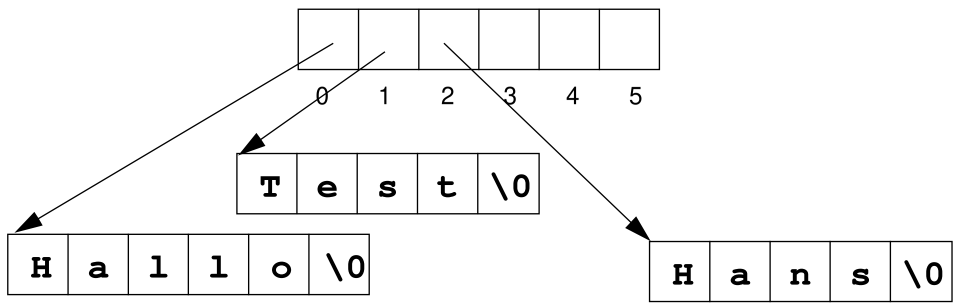
H	a	l	l	o	\0	...	\0	T	e	s	t	\0	\0	...	\0	H	a	n	s	...
0						...	99	100						...	199	200				

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

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

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

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

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

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

4 Vergleichsfunktion

- Problem: qsort erwartet folgenden Funktionstyp:

```
int (*compar) (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);
```

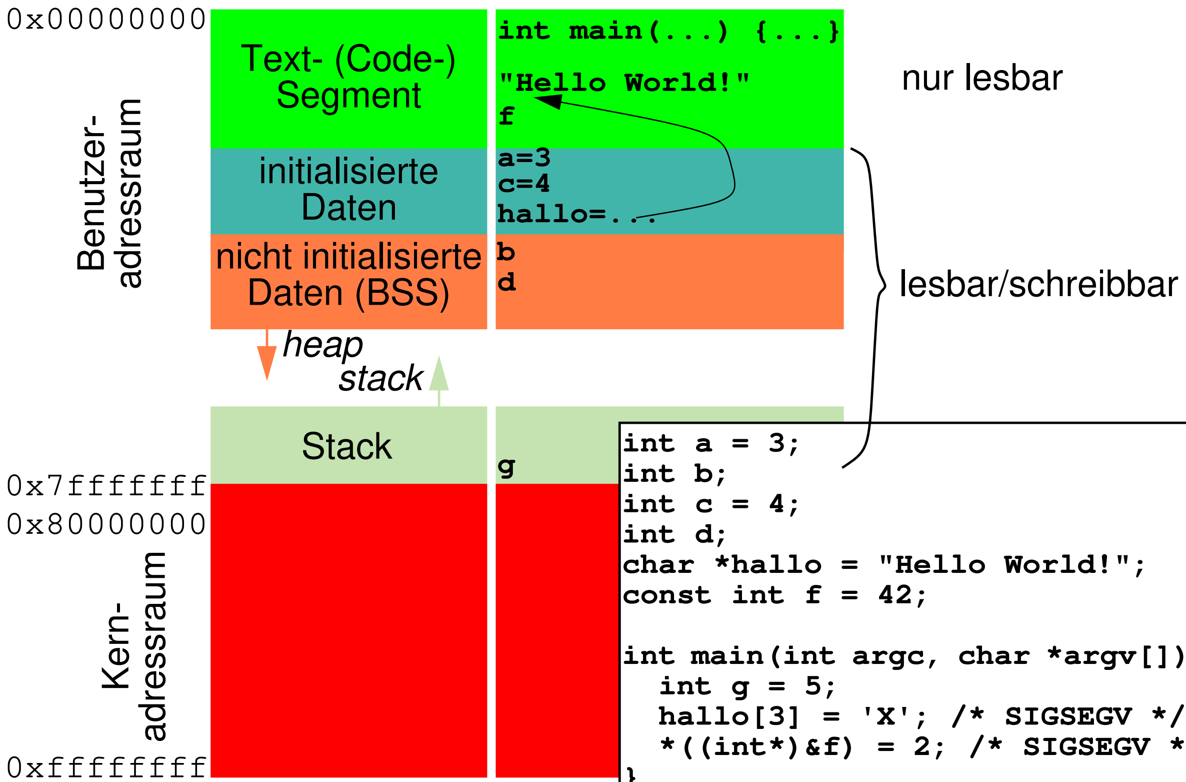

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.3 Hinweise zur 4. Aufgabe

- Prozesse
- fork, exec
- exit
- wait

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

F.4 Aufbau der Daten eines Prozesses




```c
int a = 3;
int b;
int c = 4;
int d;
char *hallo = "Hello World!";
const int f = 42;

int main(int argc, char *argv[]) {
  int g = 5;
  hallo[3] = 'X'; /* SIGSEGV */
  *((int*)&f) = 2; /* SIGSEGV */
}
```


© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2003
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

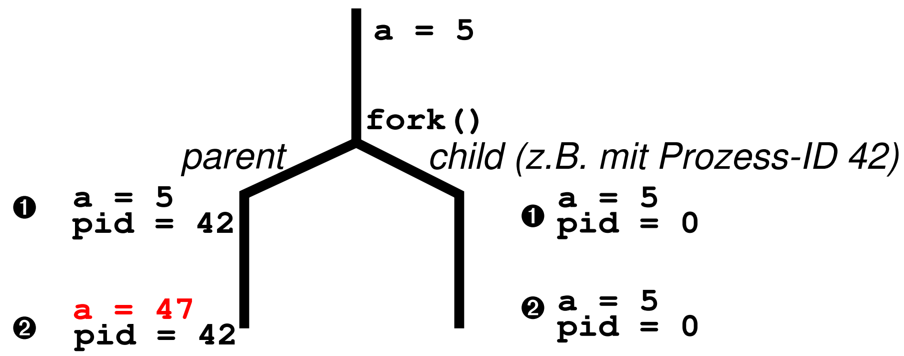
F.5 fork

- 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

© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2003
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.5 fork

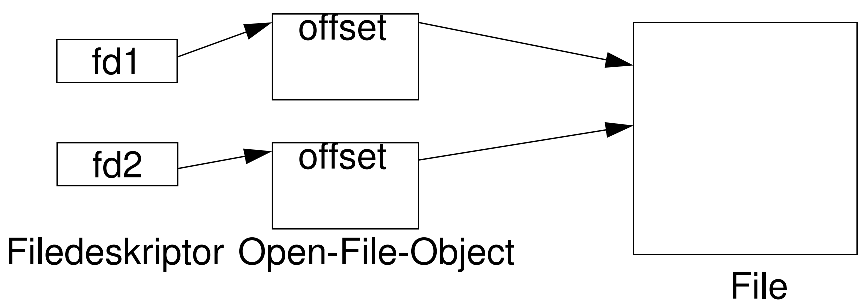
```
int a;
a = 5;
pid_t pid = fork();
 ❶
a += pid; ❷
if (pid == 0) {
    ...
} else {
    ...
}
```



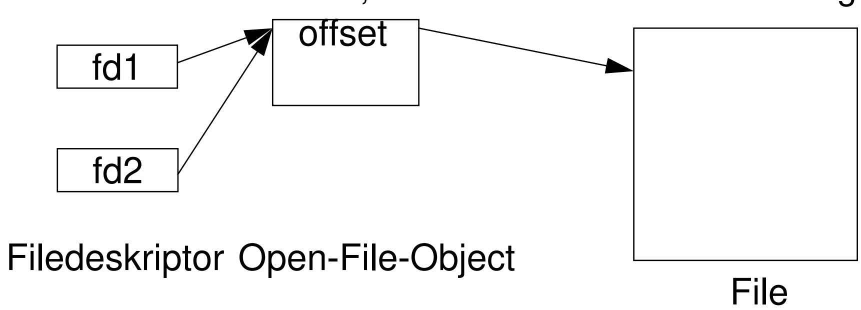
© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2003
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.6 fork und Filedeskriptoren

- erneutes Öffnen eines Files



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



© Jürgen Kleinöder • Universität Erlangen-Nürnberg • Informatik 4, 2003
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.7 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);
```


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

F.7 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 execle(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[]);
```


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

F.8 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)

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

F.9 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 */
  }
}
```


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