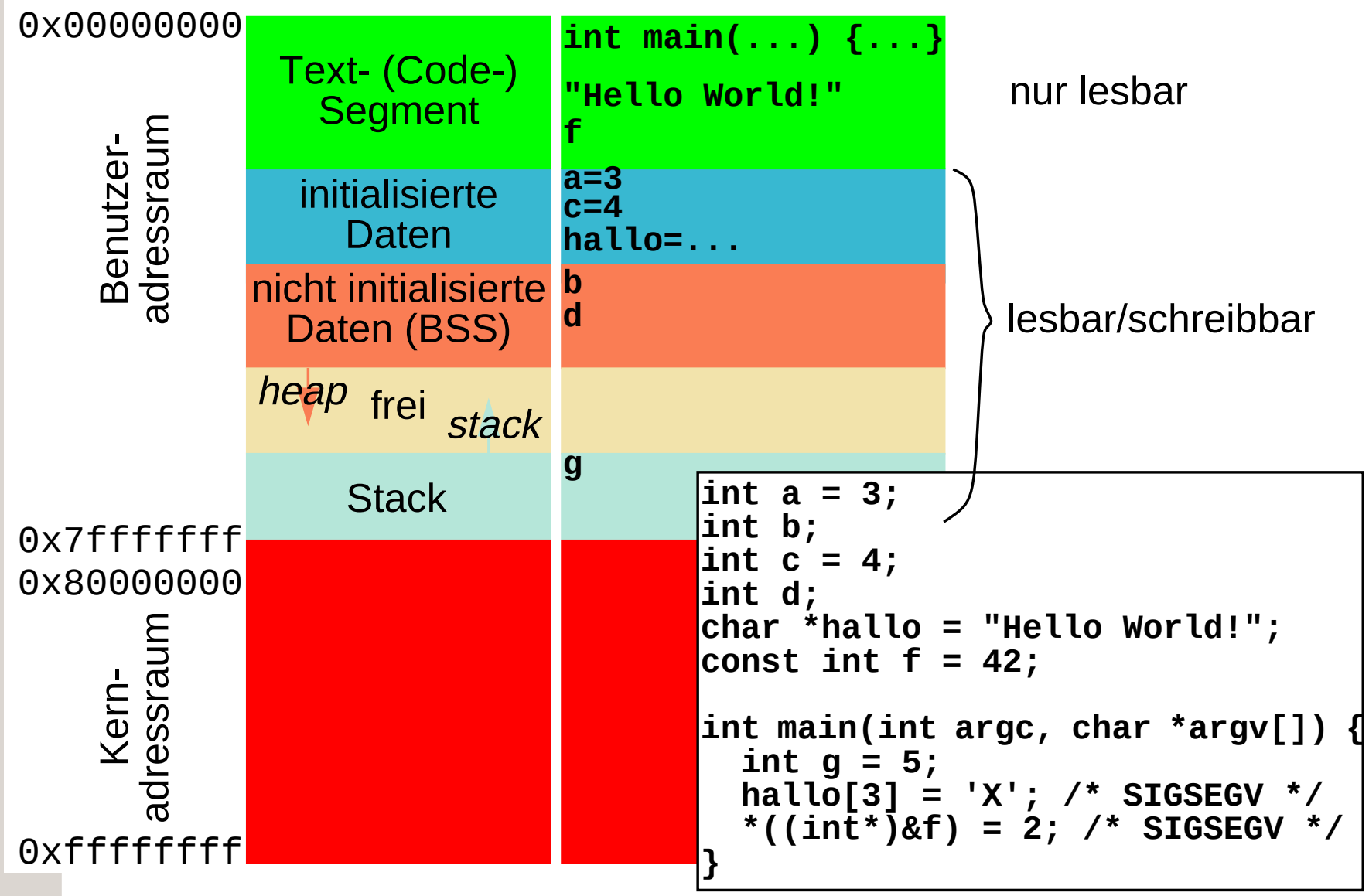
4. Tafelübung

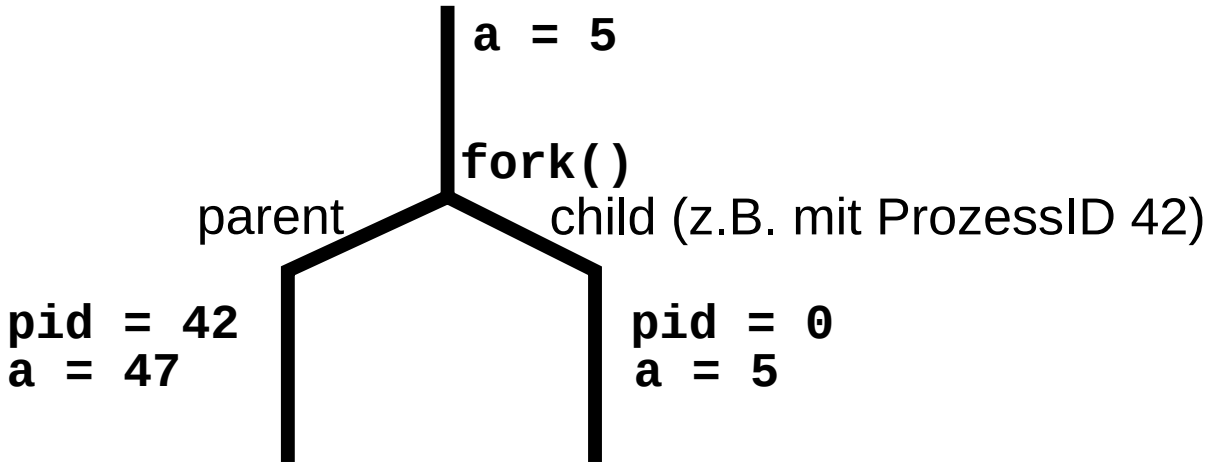
- Lösung der dir-Aufgabe
- fork, exec, wait

Aufbau der Daten eines Prozesses



fork

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



fork

- Vererbung von
  - Textsegment
  - Filedeskriptoren
  - Arbeitsverzeichnis
  - Benutzer- und Gruppen-ID (uid, gid)
  - Umgebungsvariablen
  - Signalbehandlung
  - ...
- Neu:
  - Prozess-ID
  - Datensegment, Stack
  - ...

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", 0);
```

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[]);
```

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)

wait

- warten auf Statusinformationen von Kind-Prozessen
  - **`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 */
  }
}
```