Wie werden Pipelines von der Shell aufgesetzt?

Hier gibt es eine von der alten Bourne-Shell geprägte traditionelle Vorgehensweise und eine modernere Variante, die insbesondere von der Korn-Shell und der bash praktiziert werden.

Um den Unterschied zu sehen, hilft ein kleines Testprogramm.

Testprogramm für die Shell

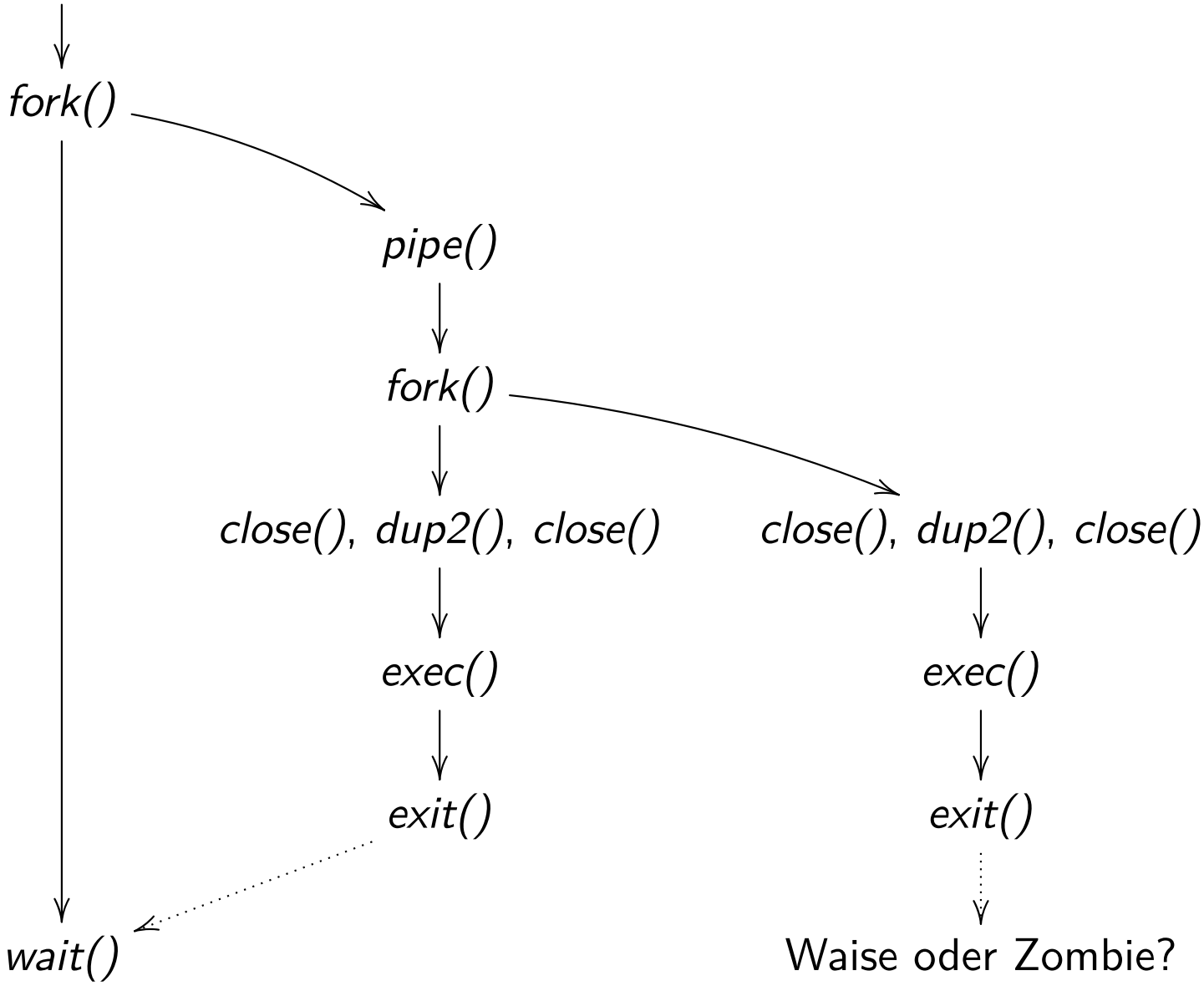
pinfo.c

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char** argv) {
   /* who are we? */
   pid_t pid = getpid();
   pid_t parent = getppid();
   /* copy standard input to standard output if it is not a terminal */
   if (!isatty(0)) {
      char buf[BUFSIZ];
      ssize_t nbytes;
      while ((nbytes = read(0, buf, sizeof buf)) > 0) {
         size_t written = 0;
         while (written < nbytes) {
            ssize_t count = write(1, buf + written, nbytes - written);
            if (count <= 0) break;
            written += count;
         }
      }
   }
   /* and add info about own process to it */
   printf("pid = %d, parent = %d\n", (int) pid, (int) parent);
}
```

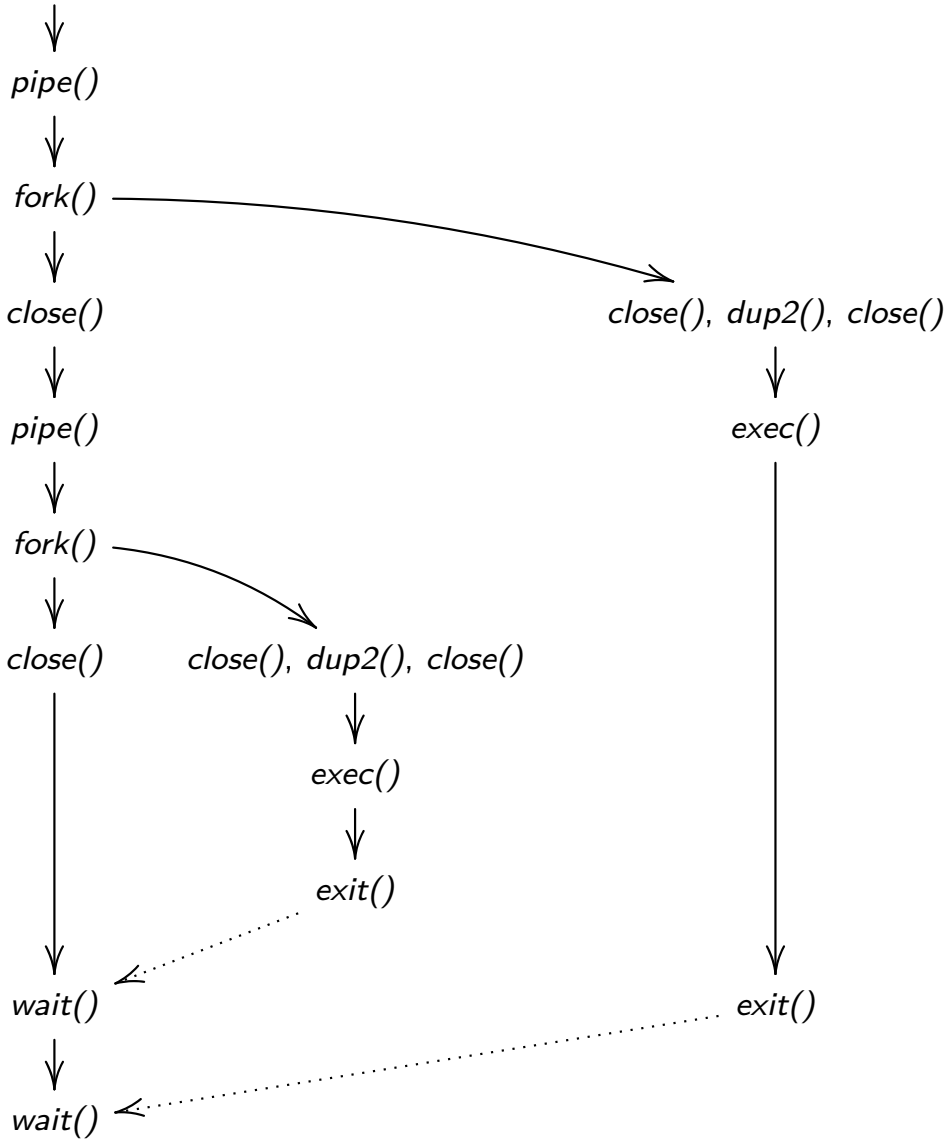
```
theseus$ /bin/sh -c 'echo $$; pinfo | pinfo | pinfo | pinfo'
25877
pid = 25879, parent = 25878
pid = 25880, parent = 25878
pid = 25881, parent = 25878
pid = 25878, parent = 25877
theseus$
```

- Der Shell-Prozess (25877) hat nur einen Kindprozess, der die gesamte Kommandozeile umsetzt (25878).
- Der Kindprozess legt die drei Pipelines an, ruft *fork* für alle Kommandos in der Pipeline mit Ausnahme des letzten, um dann schließlich selbst mit *exec* die Ausführung des letzten Kommandos in der Pipeline zu übernehmen.
- Der Shell-Prozess wartet nur auf den letzten Prozess in der Pipeline und kann nur dessen Exit-Code in Betracht ziehen.


fork()
pipe()
fork()
close(), dup2(), close()
close(), dup2(), close()
exec()
exec()
exit()
exit()
wait()
Waise oder Zombie?

```
theon$ /bin/ksh -c 'echo $$; pinfo | pinfo | pinfo | pinfo'
29704
pid = 29705, parent = 29704
pid = 29706, parent = 29704
pid = 29707, parent = 29704
pid = 29708, parent = 29704
theon$
```

- Der Shell-Prozess (29704) kontrolliert alle Kindprozesse.
- Vorteil: Es bleiben keine Waisen oder Zombies zurück, die Shell kann sich um alle von ihr erzeugten Kommandos kümmern.


pipe()
fork()
close()
pipe()
fork()
close()
close(), dup2(), close()
exec()
exit()
wait()
wait()
close(), dup2(), close()
exec()
exit()

Auf wen wird unmittelbar gewartet? 177

```
theon$ date; /bin/sh -c 'sleep 3 | true'; date
Mon May 27 13:24:29 CEST 2019
Mon May 27 13:24:29 CEST 2019
theon$ date; /bin/ksh -c 'sleep 3 | true'; date
Mon May 27 13:24:35 CEST 2019
Mon May 27 13:24:35 CEST 2019
theon$ date; /bin/bash -c 'sleep 3 | true'; date
Mon May 27 13:24:38 CEST 2019
Mon May 27 13:24:41 CEST 2019
theon$
```

- Die Bourne-Shell wartet nur auf das letzte Glied der Pipeline (auf die anderen Prozesse kann ohnehin nicht mehr gewartet werden).
- Die Korn-Shell wartet ebenfalls nur auf das letzte Glied der Pipeline – sie verhindert nur, dass die anderen Glieder der Pipeline verwaisen oder zu Zombies werden.
- Die *bash* wartet, bis alle Glieder der Pipeline fertig sind.

```
theon$ exit 1 | exit 2 | exit 3
theon$ echo ${PIPESTATUS[0]} ${PIPESTATUS[1]} ${PIPESTATUS[2]} $?
1 2 3 3
theon$
```

- In der Shell steht „$?“ für den Exit-Code des letzten Kommandos.
- Bei einer Pipeline bezieht sich das auf das letzte Glied der Kette.
- Die *bash* unterstützt über das *PIPESTATUS*-Array auch den Abruf der anderen Exit-Codes.