#uni/semester3/Betriebssysteme/chapter5/code_fragen

Dokumentation:

1. What value is the variable in the child process?
   ➜ 100

   What happens to the variable when both the child and parent change the value of x?
   ➜ Wenn child und parent den Wert von x ändern, dann ändern sie es nur für sich, da ein Kindprozess eine Kopie vom Elternprozess ist. Also jeder hat sein eigener Adressbereich und kann den anderen somit nicht beeinflussen.
   ➜ Antwort auf die Frage, was das Kind nicht vom Eltern dupliziert : Gute Erklärung - sharp_c-tudent

2. Can both the child and parent access the file descriptor returned by open()?
   ➜ Ja, Kind bekommt eine Kopie von der file descriptor table. Die file description die mit dem file descriptor assoziiert ist, wird nicht kopiert. Gute Erklärung - Alan Curry
   ➜ Beide können auch lesen, aber der zweite der liest, muss den Offset wiederherstellen

   What happens when they are writing to the file concurrently, i.e., at the same time?
   ➜ Beide können dadrauf schreiben, ohne den ganzen Text zu überschreiben. Das passiert so, weil write denn offset der Datei inkrementiert, und der nächste write dann dort weiter macht. Das geschieht undeterministisch, meistens der Eltern zuerst. Mit wait in Eltern dann schreib das Kind zuerst.

3. The child process should print “hello”; the parent process should print “goodbye”. You should try to ensure that the child process always prints first; can you do this without calling wait() in the parent?
   ➜ Ja, mithilfe von kill(parent_pid, SIGCONT); in child
   ➜ Also Kind schickt ein Signal an Eltern. Eigentlich um ein Signal schicken zu können, muss man privilegiert sein. In Fall von SIGCONT reicht es wenn Sendern und Empfänger zu der gleichen Sitzung gehören. SIGCONT sagt einem Prozess weiter zu machen wo es gestoppt wurde. (Gegenstück SIGSTOP).
   ➜ signal(SIGCONT, sig_handler); in parent
   ➜ Wenn signal SIGCONT geliefert wird, dann wird der signal handler ausgeführt. SIGCONT wird dann diesem handler (benutzerdefinierte Funktion) als Parameter gegeben.
   ➜ pause() um Prozess zu pausieren

4. Why do you think there are so many variants of the same basic call?

➜ Es gibt so viele Varianten, weil es mehrere Kombinationen gibt.

➜ L: Parameter einzeln geben als Liste (execl, execle, execlp)

➜ V: Parameter als ein Array von char* (execv, execvp, execvpe)

➜ E: Die Versionen mit das am Ende, erlauben ein extra Array von char*, also Strings die zum process environment hinzugefügt werden

➜ P: Verwendet environment variable PATH um nach der executable Datei zu suchen. Die Versionen ohne P brauchen entweder einen Absolutpfad oder relativer Pfad, wenn executable Datei nicht in gleichem Ordner.

5. What does wait() return? What happens if you use wait() in the child?

➜ wait() in parent, gibt die pid vom child zurück

➜ wait() in child, liefert -1, also ein Error. Man kann nicht in child warten. (Siehe atexit())

6. When would waitpid() be useful?

➜ wait() blockiert den Aufrufer, bis ein Kindprozess terminiert, während waitpid() eine Option hat, die das Blocken verhindert.

➜ waitpid() wartet nicht auf das Kind das zuerst terminiert, sondern hat mehrere Optionen, die kontrollieren auf welchen Prozess es wartet. Wenn *options* 0 ist, dann blockiert es, wenn WNOHANG dann blockiert nicht.

➜ Also waitpid() ist nützlich, wenn man den Aufrufer nicht blockieren möchte oder wenn man mehrere Kinder hat, und man möchte speziell auf ein Kind warten.

➜ Zombie Wait & Waitpid ist ein Prozess, das terminiert hat, aber dessen Eltern noch nicht auf ihn gewartet hat. Mit wait oder waitpid kann man Zombies entfernen.

7. What happens if the child calls printf() to print some output after closing the descriptor?

➜ Kind kann kein Text mehr an die Ausgabe schicken. File descriptor wird jedoch nicht für den Parent blockiert, sodass dieser noch Text ausgeben kann.

➜ Wenn man den Rückgabewert von printf anschaut, dann werden die geschriebenen Bytes zurück gegeben. Also er printet schon in Buffer, aber da es blocked buffer ist (also bis ein block voll ist, dann schicken) passiert nichts. Wenn man schreibt, öffnet ein neues file descriptor und flusht, dann wird der Text in die Datei geschrieben. **fflush(stdout);** Erklärung was mit Buffer passiert

8. Ein child erstellt man im ersten if, den zweiten in dem if von parent. Einer schreibt und der andere liest. Die file descriptors erstellt man mit pipe().