Technische Informatik 2
Tutor: Marc/Bingbin

WS 2016/17

C05

Tabea Eggers
Jan Fiedler
Florian Pflüger
Jonas Schmutte

Übungsblatt 3

1	2	3	4	5	$\sum$

Lösungsvorschlag
Abgabe: 21.11.2016

Aufgabe 1

Beim Start gibt es einen Page-fault, weil ls nicht geladen ist, dies löst eine Trap aus, welche einen Interrupt auslöst, der einen Leseauftrag an die Platte sendet. Sobald fertig geladen ist, wird ein Interrupt ausgelöst, dass die Platte fertig ist, nun wird ein Signal gesendet, dass der Prozess abgearbeitet werden kann.

Dann muss die Systemuhr weitergestellt werden, dies löst einen Interrupt aus.

ls löst einen Page-Fault und damit eine Trap aus, da es Bereiche des Speichers einlesen will, die nicht im Hauptspeicher vorhanden sind, dies löst einen Interrupt aus, der einen Leseauftrag an die Platte sendet. Sobald fertig geladen ist, wird ein Interrupt ausgelöst, dass die Platte fertig ist, nun wird ein Signal gesendet, dass der Prozess abgearbeitet werden kann.

Wenn ls beginnt in die Datei Ausgabe zu schreiben, wird durch den Systemaufruf von write() gewollt eine Trap ausgelöst.

Strg-Z löst zunächst einen Interrupt aus der die eingegebenen Zeichen rettet. Nun wird das Signal SIGSTOP ausgelöst, welches ls stoppt.

bg löst einen Interrupt aus der die eingegebenen Zeichen rettet. Dann kommt es zu einem Page-fault welcher eine Trap auslöst, welche einen Interrupt auslöst, der einen Leseauftrag an die Platte sendet. Sobald fertig geladen ist, wird ein Interrupt ausgelöst, dass die Platte fertig ist, nun wird ein SIGCONT-Signal an den ls-Prozess gesendet, wodurch dieser fortgeführt wird.

Ein letzter Interrupt wird ausgelöst wenn die Platte fertig ist mit schreiben und dies meldet.

Aufgabe 2

```
7  using namespace std;
8
9  /*
10  * Die Methode die bestimmt wie mit den Signalen umgegangen wird.
11  * @param signr - Die Nummer des eingegangenen Signals
12  * @param siginfo - Struktur mit Informationen ueber das eingegangene
        Signal
13  * Der dritte Parameter wird in diesem Fall nicht benoetigt.
14  */
15 static void handel(int signr, siginfo_t* siginfo, void*){
16     cout << "Signalnummer: " << signr << endl;
17     cout << "Prozess ID: " << siginfo->si_pid << endl;
```

```
18      cout << "Real User ID: " << siginfo->si_uid << endl;
19  }
20
21  int main() {
22      // Legt unsere sigaction an
23      struct sigaction sigact;
24      // Setzt die Variante sa_sigaction zum Setzen unseres handlers
25      sigact.sa_flags = SA_SIGINFO;
26      // Setzt die Methode zum umgehen mit Signalen auf handel()
27      sigact.sa_sigaction = &handel;
28      // Die sigaction sigact soll benutzt werden um SIGUSR1 Signale zu
            handeln
29      if (sigaction(SIGUSR1, &sigact, NULL) == -1) cout << errno <<
            endl;
30      // Wartet in einer Endlosschleife auf eingehende Signale
31      while (true) {
32          pause();
33      }
34  }
```

Tabelle 1: Tests

```
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ g++ -o ./sigserver sigserver.cc
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ ./sigserver&
[1] 5259
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ echo $BASHPID
1947
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ id -ur jonas
1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -10 5259
Signalnummer: 10
Prozess ID: 1947
Real User ID: 1000
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ kill -9 5259
[1]+  Getötet                 ./sigserver
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ jobs
jonas@jonas-ThinkPad-T400 ~/Git/ti2-c05/03_C05 $ 
```

Zuerst wird der Prozess gestartet und ihm ein SIGUSR1-Signal geschickt. Seine Ausgabe gilt es zu überprüfen, Die Signalnummer ist richtig. Mit 'echo $BASHPID' wird die ProzessID der Bash(dem hier aufrufenden Prozess) und mit 'id -ur jonas' die Real User ID des Benutzers. Diese stimmen jeweils mit der Ausgabe von 'sigserver' überein.

Danach wird wiederholt das SIGUSR1-Signal an den laufenden 'sigserver'-Prozess geschickt, um zu zeigen, dass er endlos auf auf Signale wartet.

Zum Schluss schicken wir das SIGKILL-Signal an den 'sigserver'-Prozess um zu überprüfen, dass andere Signale ihre Standardwirkung nicht verlieren. Und der Prozess wird tatsächlich getötet.

Aufgabe 3

zu a)
Wenn der CPU-Kontostand nicht halbiert werden würde, würden die Kontostände im Laufe der Zeit immer höher werden. Dies hätte zur Folge, dass auch die Prioritäten immer geringer werden würden, weil diese auch immer höher werden. Somit würden sich die Prozesse zwar immer abwechseln, aber die Prioritäten immer mehr sinken. Käme jetzt ein neuer Prozess hinzu, würde dieser immer wieder aufgerufen werden solange, bis die Priorität eines vorherigen Prozesses

wieder höher ist. Kommen nun aber immer wieder neue Prozesse hinzu, werden die ersten Prozesse also nie wieder aufgerufen, da ihre Priorität zu gering ist. Im schlimmsten Fall sind diese Prozesse aber sehr wichtig und es würde somit zu Fehlern kommen, wenn diese nicht mehr aufgerufen werden. Deshalb ist es sinnvoll Prozesse 'veraltern' zu lassen.

zu b)

	0	1	2	3	4	5	6	7	8	9	10	11	12
Konto A	0	0	0	50	25	12	56	28	14	57	28	14	57
Nutzung A	0	0	100	0	0	100	0	0	100	0	0	100	0
Konto A'	0	0	50	25	12	56	28	14	57	28	14	57	28
Prio A	56	56	106	81	68	112	84	70	113	84	70	113	84
Konto B	0	0	50	25	62	81	40	70	85	42	71	85	42
Nutzung B	0	100	0	100	100	0	100	100	0	100	100	0	100
Konto B'	0	50	25	62	81	40	70	85	42	71	85	42	71
Prio B	10	60	35	72	91	50	80	95	52	81	95	52	81
welcher Prozess läuft gerade?	-	B	A	B	B	A	B	B	A	B	B	A	B

Ja, das Ergebnis lässt eine Regelmäßigkeit erkennen. Zuerst fällt auf, dass es einen Rhythmus gibt, in dem sich die Prozesse abwechseln (Prozess B läuft zwei mal, dann Prozess A, usw.). Schaut man sich dann die einzelnen Werte für einen Prozess an, fällt auf, dass diese in einem Abstand von 4 Zeitscheiben wieder in einem ähnlichen Wertebereich liegen (bei Prozess A ab 3 und bei Prozess B ab 4).

zu c)

	0	1	2	3	4	5	6	7	8	9
Konto A	0	0	50	75	87	93	96	98	99	99
Nutzung A	0	100	100	100	100	100	100	100	100	100
Konto' A	0	50	75	87	93	96	98	99	99	99
Prio A	0	50	75	87	93	96	98	99	99	99
Konto B	0	0	0	0	0	0	0	0	0	0
Nutzung B	0	0	0	0	0	0	0	0	0	0
Konto' B	0	0	0	0	0	0	0	0	0	0
Prio B	99	99	99	99	99	99	99	99	99	99
welcher Prozess läuft gerade?	-	A	A	A	A	A	A	A	A	A

Der Prozess B würde ab einer Basispriorität von 99 (wenn man annimmt das bei gleicher Priorität immer A die CPU bekommt, ansonsten ab einer Basispriorität von 100) nie die CPU bekommen, da Prozess A bei einer Basispriorität von 0 nicht höher als Priorität von 99 kommt.

zu d)

Annahme: Eine Zeitscheibe dauert 200ms.

Läuft ein Prozess wird er nicht unterbrochen. Prozess B muss also warten bis zur nächsten Zeitscheibe.

	0	1	2	3	4	5	6	7	8	9	10	11	12
Konto A	0	0	50	61	80	76	88	80	90	81	90	81	90
Nutzung A	0	100	73	100	73	100	73	100	73	100	73	100	73
Konto A'	0	50	61	80	76	88	80	90	81	90	81	90	81
Prio A	0	50	61	80	76	88	80	90	81	90	81	90	81
Konto B	0	0	0	13	6	16	8	17	8	17	8	17	8
Nutzung B	0	0	27	0	27	0	27	0	27	0	27	0	27
Konto B'	0	0	13	6	16	8	17	8	17	8	17	8	17
Prio B	15	15	28	21	31	23	32	23	32	23	32	23	32