Probe Klausur SNP FS20

Zeit: 90 Minuten

Bedingungen:

- Die Aufgaben werden auf den ausgeteilten Blättern gelöst.
- Falls der Platz nicht ausreicht, benutzen Sie bitte die Rückseiten und bringen Sie bei der Aufgabe einen entsprechenden Verweis an.
- Blätter **nicht** auseinandernehmen!
- Prüfung ist Open-Book**,** jedoch sind **keine** elektronischen Hilfsmittel erlaubt
- Wo immer möglich soll der Lösungsweg ersichtlich sein.
- Unredliches Verhalten hat die Note 1 zur Folge.

Maximale Punktzahl:

Answers

als "verborgener Text"

Aufgabe 1: 10 Punkte **a) 2P b) 2P c) 2P d) 2P e) 2P**

Ergänzen Sie die Adressen und Inhalte des Arrays txt. Dabei werden die Werte des Arrays von einer Teilaufgabe zur nächsten überragen. Der Array txt werde an der Adresse 104 also HEX 0x68 angelegt.

a) `char txt[3] = {'\0'};` 2 Punkte

Adresse [**HEX**]	00 00 00 68	**00 00 00 69**	**00 00 00 6A**		
Inhalt [**HEX**]	**00**	**00**	**00**		

b) `txt[2] = 80;` 2 Punkte

Inhalt [**HEX**]	**00**	**00**	**50**		

c) 2 Punkte

```
char *p = txt;
*p = *(p + 2) + 3;
```

Inhalt [**HEX**]	**53**	**00**	**50**		

d) `*(p + 1) = (int)txt - 3;` 2 Punkte

Inhalt [**HEX**]	**53**	**65**	**50**		

e) Inhalt von txt als char 2 Punkte

Inhalt [**CHAR**]	**'S'**	**'e'**	**'P'**		

1 P: Position(en)
1 P: Inhalt

Dez	Hex	Okt	
0	0x00	000	NUL
1	0x01	001	SOH
2	0x02	002	STX
3	0x03	003	ETX
4	0x04	004	EOT
5	0x05	005	ENQ
6	0x06	006	ACK
7	0x07	007	BEL
8	0x08	010	BS
9	0x09	011	TAB
10	0x0A	012	LF
11	0x0B	013	VT
12	0x0C	014	FF
13	0x0D	015	CR
14	0x0E	016	SO
15	0x0F	017	SI
16	0x10	020	DLE
17	0x11	021	DC1
18	0x12	022	DC2
19	0x13	023	DC3
20	0x14	024	DC4
21	0x15	025	NAK
22	0x16	026	SYN
23	0x17	027	ETB
24	0x18	030	CAN
25	0x19	031	EM
26	0x1A	032	SUB
27	0x1B	033	ESC
28	0x1C	034	FS
29	0x1D	035	GS
30	0x1E	036	RS
31	0x1F	037	US

Dez	Hex	Okt	
32	0x20	040	SP
33	0x21	041	!
34	0x22	042	"
35	0x23	043	#
36	0x24	044	$
37	0x25	045	%
38	0x26	046	&
39	0x27	047	'
40	0x28	050	(
41	0x29	051	)
42	0x2A	052	*
43	0x2B	053	+
44	0x2C	054	,
45	0x2D	055	-
46	0x2E	056	.
47	0x2F	057	/
48	0x30	060	0
49	0x31	061	1
50	0x32	062	2
51	0x33	063	3
52	0x34	064	4
53	0x35	065	5
54	0x36	066	6
55	0x37	067	7
56	0x38	070	8
57	0x39	071	9
58	0x3A	072	:
59	0x3B	073	;
60	0x3C	074	<
61	0x3D	075	=
62	0x3E	076	>
63	0x3F	077	?

Dez	Hex	Okt	
64	0x40	100	@
65	0x41	101	A
66	0x42	102	B
67	0x43	103	C
68	0x44	104	D
69	0x45	105	E
70	0x46	106	F
71	0x47	107	G
72	0x48	110	H
73	0x49	111	I
74	0x4A	112	J
75	0x4B	113	K
76	0x4C	114	L
77	0x4D	115	M
78	0x4E	116	N
79	0x4F	117	O
80	0x50	120	P
81	0x51	121	Q
82	0x52	122	R
83	0x53	123	S
84	0x54	124	T
85	0x55	125	U
86	0x56	126	V
87	0x57	127	W
88	0x58	130	X
89	0x59	131	Y
90	0x5A	132	Z
91	0x5B	133	[
92	0x5C	134	\
93	0x5D	135	]
94	0x5E	136	^
95	0x5F	137	_

Dez	Hex	Okt	
96	0x60	140	`
97	0x61	141	a
98	0x62	142	b
99	0x63	143	c
100	0x64	144	d
101	0x65	145	e
102	0x66	146	f
103	0x67	147	g
104	0x68	150	h
105	0x69	151	i
106	0x6A	152	j
107	0x6B	153	k
108	0x6C	154	l
109	0x6D	155	m
110	0x6E	156	n
111	0x6F	157	o
112	0x70	160	p
113	0x71	161	q
114	0x72	162	r
115	0x73	163	s
116	0x74	164	t
117	0x75	165	u
118	0x76	166	v
119	0x77	167	w
120	0x78	170	x
121	0x79	171	y
122	0x7A	172	z
123	0x7B	173	{
124	0x7C	174	|
125	0x7D	175	}
126	0x7E	176	~
127	0x7F	177	DEL

Aufgabe 2 : 10 Punkte **a) 4P b) 1P c) 1P d) 4P**

Gegeben ist die Funktion:

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

void inverse(char * const);

int main(void){
  char c[] = "SEPFS16";
  char *p = c;
  (void) printf("\n%d", sizeof(c));
  (void) printf("\n%c", p[0]);
  (void) printf("\n%d", p[0] - p[2]);
  (void) printf("\n%s", p + p[0] - p[2]);
  inverse(p);
  (void) printf("\n%s", p);
  return EXIT_SUCCESS;
}
```

a) Was ist die Ausgabe von: 4 Punkte

```
(void) printf("\n%d", sizeof(c));
```
1P : 8

```
(void) printf("\n%c", p[0]);
```
1P : S

```
(void) printf("\n%d", p[0] - p[2]);
```
1P : 3

```
(void) printf("\n%s", p + p[0] - p[2]);
```
1P : FS16

b) Ist dieser Ausdruck Pointerarithmetik ? 1 Punkt

```
(void) printf("\n%d", p[0] - p[2]);
```
1P : NO

c) Ist dieser Ausdruck Pointerarithmetik ? 1 Punkt

```
(void) printf("\n%s", p + p[0] - p[2]);
```
1P : YES

d) Schreiben Sie die Funktion inverse um die Buchstaben in c[ ] von hinten nach vorne umzutauschen. 4 Punkte

```
1  void inverse(char * const pc) {
2    int i = 0, len = 0, temp = 0;
3    for(i = 0; *(pc + i); i++, len++);
4    for(i = 0; i < len/2; i++) {
5      temp = *(pc + i);
6      *(pc + i) = *(pc + len - i - 1);
7      *(pc + len - i - 1) = temp;
8    }
9
10
11 }
```

1P: Länge bestimmen
1P: Iteration zur Mitte
1P: Temp Variable
1P: korrekter Tausch

Aufgabe 3 : 6 Punkte — a) 2 P b) 4 P

Schreiben Sie entsprechenden Source in ANSI C

a) Bestimmen Sie mithilfe des Modulo-Operators die zweitletzte Ziffer einer mindestens dreistelligen Integer Zahl. Sie dürfen auch weitere, zusätzliche Operatoren verwenden. — 2 Punkte

```
1  int z = 123;
2
3  z = (z % 100) - (z % 10)) / 10 ;
4
5
6
7
```

1P: modulo verwendet

1P: Resultat korrekt

b) Zur exakten Festlegung der Schaltjahre dienen die folgenden Regeln: — 4 Punkte
► ist die Jahreszahl durch 4 teilbar, so ist das Jahr ein Schaltjahr.

Diese Regel hat allerdings eine Ausnahme:
► ist die Jahreszahl durch 100 teilbar, so ist das Jahr kein Schaltjahr.

Diese Ausnahme hat wiederum eine Ausnahme:
► ist die Jahreszahl durch 400 teilbar, so ist das Jahr doch ein Schaltjahr.

Erstellen Sie ein C Source Sequenz, die berechnet, ob eine Jahreszahl ein Schaltjahr bezeichnet oder nicht. Geben Sie das Resultat als "YES" oder "NO" auf der Konsole aus.

```
1  int jahr = 2016;
2  if ( jahr % 4 == 0)
3    if( jahr % 100 == 0)
4      if( jahr % 400 == 0) printf ("yes");
5      else printf ("no");
6    else printf ("yes");
7  else printf ("no");;
8
9
10
11
12
13
14
15
16
17
18
```

1P: pro korrektes printf ► 4P

Aufgabe 4: 8 Punkte **a) 2 P b) 2 P c) 2 P d) 2 P**

Gegeben ist der folgende Programmcode:

```
1
2  int *pi1, *pi2, i;
3
4  pi1 = pi2 + i;
5  pi1 = i + pi2;
6  i = pi1 * pi2;
7  i = pi1 - pi2;
8
9
```

Erklären Sie die Bedeutung der Zeilen mit den folgenden Nummern:

a) Zeile 4: `pi1 = pi2 + i;` 2 Punkte

1P: pi2 ist ein Pointer, also dessen Inhalt eine Adresse
1P: dazu wird i addiert und als neue Adresse in pi1 gespeichert

b) Zeile 5: `pi1 = i + pi2;` 2 Punkte

1P: i ist eine integer Zahl, dazu wird eine Adresse pi2 addiert
1P: das Resultat wird als neue Adresse in pi1 gespeichert

c) Zeile 6: `i = pi1 * pi2;` 2 Punkte

1P: die beiden Inhalte von pi1 und pi2 (Adressen) werden multipliziert
1P: das macht keinen Sinn, Pointer-Multiplikation und -Division sind nicht erlaubt

d) Zeile 7: `i = pi1 - pi2;` 2 Punkte

1P: die beiden Inhalte von pi1 und pi2 (Adressen) werden subtrahiert
1P: resultiert im Offset als sizeof(int) der beiden Pointer

Aufgabe 5: Prozesse 4P

Gegeben ist der Folgende Code sowie erzeugter Prozessbaum. Gehen Sie davon aus, dass alle `forks()` erfolgreich ausgeführt werden.

```
int main(void) {
    fork();
    fork();
    fork();
    fork();
    exit(0);
}
```

```
ProcA3.e─┬─ProcA3.e─┬─ProcA3.e─┬─ProcA3.e──ProcA3.e
         │          │          └─ProcA3.e
         │          ├─ProcA3.e──ProcA3.e
         │          └─ProcA3.e
         ├─ProcA3.e─┬─ProcA3.e──ProcA3.e
         │          └─ProcA3.e
         ├─ProcA3.e──ProcA3.e
         └─ProcA3.e
```

Zeichnen Sie den Prozessbaum auf, nachdem der Code wie folgt modifiziert wurde:

```
int main(void) {
    fork();
    fork();
    fork();
    pid_t pid = fork();
    if (pid == 0) {
       fork();
    }
    exit(0);
}
```

```
 ProcA3.e─┬─ProcA3.e─┬─ProcA3.e─┬─ProcA3.e──ProcA3.e──ProcA3.e
          │          │          └─ProcA3.e──ProcA3.e
          │          ├─ProcA3.e──ProcA3.e──ProcA3.e
          │          └─ProcA3.e──ProcA3.e
          ├─ProcA3.e─┬─ProcA3.e──ProcA3.e──ProcA3.e
          │          └─ProcA3.e──ProcA3.e
          ├─ProcA3.e──ProcA3.e──ProcA3.e
          └─ProcA3.e──ProcA3.e
```

Aufgabe 6: Signale 6 Punkte

Ergänzen Sie den folgenden Code so, dass folgende Anforderungen erfüllt sind:
Wenn ein Benutzer Ctrl-C drückt, dann soll

- der Parent-Prozess den Text „User Interrupt“ ausgeben
- der Child-Prozess diese Benutzer-Aktion ignorieren

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>

#define PERROR_AND_EXIT(M) do{perror(M);exit(EXIT_FAILURE);}
while(0)
void do_work() { for(;;){} }

 // snp/theorie/SNP_10_Sys_Inter_Prozess_Kommunikation/
 // SNP_Sys_Inter_Prozess_Kommunikation.pdf page 18

 static void handler(int sig, siginfo_t *siginfo, void
 *context) {
     printf("User Interrupt\n");
 }


int main(void) {
    pid_t cpid = fork();
    if (cpid == -1) { PERROR_AND_EXIT("fork"); }
    if (cpid > 0) {

          struct sigaction a = { 0 };
          a.sa_flags = SA_SIGINFO;
          a.sa_sigaction = handler;
          if (sigfillset(&a.sa_mask) == -1) {
              PERROR_AND_EXIT("sigfillset");
          }
          if (sigaction(SIGINT, &a, NULL) == -1) {
              PERROR_AND_EXIT("sigaction");
          }

do_work();
    } else {

           struct sigaction a = { 0 };
           a.sa_flags = SA_SIGINFO;
           a.sa_sigaction = SIG_IGN;
           if (sigfillset(&a.sa_mask) == -1) {
               PERROR_AND_EXIT("sigfillset");
           }
           if (sigaction(SIGINT, &a, NULL) == -1) {
               PERROR_AND_EXIT("sigaction");
           }

do_work();
    }
}
```

Aufgabe 7: Deadlocks 3 Punkte

Ein Rechnersystem besitzt zwei Tapestationen (T_1, T_2) und zwei Disks (D_1, D_2). Zur Zeit laufen drei Prozesse (P_1, P_2, P_3), wobei folgendes gilt:

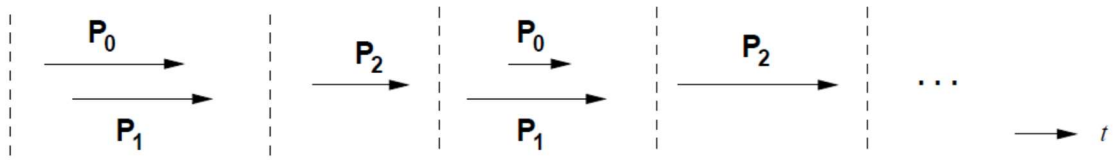
- Prozess P_1 kopiert Daten von Disk D_1 auf die Tapestation T_2 und möchte Daten auf den Disk D_2 schreiben
- Prozess P_2 hat Tapestation T_1 alloziert und möchte Daten auf Disk D_2 schreiben
- Prozess P_3 hat Disk D_2 alloziert und möchte Daten nach Tapestation T_2 kopieren

Ist das eine Deadlocksituation, wenn die Ressourcen exklusiv alloziert werden und wenn *möchte schreiben* das Gleiche wie anfordern bedeutet? Begründen Sie Ihre Antwort.

There is a deadlock situation between P1 and P3 because P1 wants D2 which is owned by P3 which wants T2 which is owned by P1. This is a circular wait so all four conditions necessary for a deadlock are given.

Aufgabe 8: Semaphore 4 Punkte

Gegeben sind drei Prozess P_0, P_1, und P_2 die nach folgendem Schema abgearbeitet werden soll:



Die Verarbeitung startet mit den beiden Prozessen P_0 und P_1, die parallel verarbeitet werden sollen (es spielt kein Rolle, welcher der beiden Prozesse zuerst mit seiner Verarbeitung beginnt oder aufhört). Wenn beide Prozesse eine Iteration ihrer Funktion working(x) beendet haben, folgt Prozess P_2, etc.

Schreiben Sie Pseudocode mit maximal 3 Semaphoren S0, S1 und S3, der garantiert, dass die oben skizzierte Reihenfolge eingehalten wird. Verwenden Sie dazu **ausschliesslich** Befehle der Form up(S0) und down(S0), etc. Geben Sie an, wie die Semaphore initialisiert werden müssen.

P_0	P_1	P_2

Sem S0 — Sem s0 = 1;
Sem S1 — Sem s1 = 1;
Sem S2 — Sem s2 = 0;
while{1} {
down(s0);
working(0);
up(s2);
}
	while{1} {
	down(s1);
	working(1);
	up(s2);
	}
		while{1} {
		down(s2);
		down(s2);
		working(2);
		up(s0);
		up(s1);
		}