Betriebssysteme
Klausur 2, WS 12/13

Prof. Dr. Marcel Waldvogel, Dr. Stefan Klinger, Roman Byshko, Daniel Kaiser

30. Jan 2015

Matrikelnummer:

_____

Note:

_____

Bearbeitungszeit: 120 Minuten.
Es sind keine Hilfsmittel zugelassen!
Viel Erfolg!

	1	2	3	4	5	6	7	8	9	10	Bonus	Σ
erreicht												
von	18	8	9	8	15	12	10	5	5	10	3	100

Aufgabe 1 — 18 Punkte

1. Was ist Swapping und was der Zweck von Swapping? (1)
2. Welche Schritte führt ein Betriebssystem beim Erstellen eines neuen Pro zesses durch? (1)
3. Warum gibt es zwei verschiedene Modi: Usermode und Kernelmode? (1)
4. Was ist der Unterschied zwischen einem "Mode switch" und einem "Pro cess switch"? (2)
5. Welche Ressourcen werden typischerweise von allen Threads eines Prozes ses geteilt? (1)
6. Was versteht man unter blocking, was unter non blocking im Kontext von Message Passing? (2)
7. Was ist ein Monitor im Kontext nebenläufiger Programmierung? Beschrei ben sie kurz die Funktionsweise eines solchen Monitors. (2)
8. Beschreiben Sie kurz deadlock avoidance, deadlock detection und deadlock prevention. Was sind die Unterschiede? (2)
9. Was ist eine logische Adresse, was eine relative Adresse, was eine physika lische Adresse? Was sind die Unterschiede? (2)
10. Was versteht man unter resident set, was unter working set im Kontext der Speicherverwaltung? Was sind die Unterschiede?(2)
11. Was versteht man unter precleaning, was unter demand cleaning im Kon text der Speicherverwaltung? Was sind die Unterschiede?(2)

Aufgabe 2 - Race Condition — 8 Punkte

Betrachten Sie das folgende Programm:

```
const int n=50;
int anzGesamt;

void total(){
  int count;
  for(count=1; count<=n; count++) {
    anzGesamt++;
  }
}

int main(void) {
  anzGesamt=0;
  parbegin(total(), total());
  write(anzGesamt);
  return 0;
}
```

(a) Was ist eine Race Condition im Allgemeinen? Warum kommt es in diesem Programm zu einer Race Condition?(2)

(b) Bestimmen und begründen Sie die korrekte untere und obere Grenze des Endwerts der Ausgabe der gemeinsamen Variablen *anzGesamt* durch dieses nebenläufige Programm. Nehmen Sie dafür an, dass Prozesse mit jeder beliebigen relativen Geschwindigkeit ausgeführt werden können und dass ein Wert nur erhöht werden kann, nachdem er mit Hilfe eines separaten Maschinenbefehls in ein Register geladen wurde. (4)

(c) Nehmen Sie an, dass unter Anwendung der Voraussetzungen in Teil (a) eine willkürliche Anzahl dieser Prozesse parallel ausgeführt werden darf. Welche Auswirkungen hat diese Änderung auf die Bandbreite der Endwerte von *anzGesamt* ? (2)

Anmerkung: parbegin($P_1, P_2, \ldots, P_n$) bedeutet: Suspendiere die Ausführung des Main Programms, starte die Prozeduren $P_1, P_2, \ldots, P_n$ nebenläufig und führe das Main Programm weiter aus, nachdem alle $P_1, P_2, \ldots, P_n$ terminiert sind. (Dies ist ein von Dijkstra eingeführtes theoretisches Konstrukt. In C/C++ gibt es diesen Befehl nicht.)

Aufgabe 3 - Hierarchical Page Table 9 Punkte

Ein Betriebssystem verwende eine Virtual Paging Speicherverwaltung mit 64 bit virtuellen Adressen und einer Pagegröße von 4 KiB. Jeder Eintrag der Pagetable benötige 64 Bits. Erwünscht ist, dass eine ganze Pagetable in eine Page passt.

a) Wie viele Levels von Pagetables werden benötigt? Begründen Sie bitte. (3)

b) Geben sie für jedes dieser Level die Größe einer Pagetable auf diesem Level an. Begründen Sie bitte. (3)

c) Was ist die minimale Anzahl von Pages, die bei diesem System für die Pagetables benötigt werden? Begründen Sie bitte. (3)

Aufgabe 4 - Page Fault — 8 Punkte

Gegeben sei folgendes Programm:

```
#define Size 64

int main(void) {
        int A[Size][Size], B[Size][Size], C[Size][Size];
        int register i, j;
        for (j = 0; j < Size; j++)
                for (i = 0; i < Size; i++)
                        C[i][j] = A[i][j] + B[i][j];
        return 0;
}
```

Angenommen, das Programm läuft auf einem System, welches demand paging verwendet und die Pagegröße ist ein KiB. Ein Integer ist 4 Byte lang. Die Elemente der Arrays liegen aufeinander folgend im Speicher. Jedes der zweidimensionalen Arrays benötigt 16 Pages. In der ersten Page von Array A liegen A[0][0] bis A[0][63], A[1][0] bis A[1][63], A[2][0] bis A[2][63] und A[3][0] bis A[3][63]. Die anderen Seiten von A und die Seiten der anderen Arrays sind analog gefüllt. Nehmen Sie an, dass das System für diesen Prozess ein working set von 4 Pages wählt. Eine der Pages wird für das Programm gewählt, die anderen für Daten.

(a) Wie häufig tritt ein Pagefault im Verhältnis zur Anzahl der Ausführungen von C[i][j] = A[i][j] + B[i][j] auf? (2)

(b) Modifizieren Sie das Programm, sodass es weniger Pagefaults produziert. (4)

(c) Wie häufig treten Pagefaults nach Ihrer Modifikation auf? (2)

Aufgabe 5 - Dining Philosophers Monitor 15 Punkte

Lösen Sie das Dining Philosophers Problem mit Hilfe eines Monitors; geben Sie Pseudocode an. Keiner der Philosophen darf verhungern.

Aufgabe 6 - Virtual Memory Paging — 12 Punkte

Beschreiben Sie, welche Schritte durchgeführt werden, wenn der Prozessor in einem System, welches Virtual Memory Paging verwendet, einen load Befehl ausführt (Der load Befehl lade den Inhalt der angegebenen Speicherstelle in ein Register). Gehen Sie dabei auf die Übersetzung von logischen Adressen in physikalische Adressen, den Translation Lookaside Buffer und den Cache des Prozessors ein. Beschreiben Sie alle möglichen Szenarien. Sie können ein Diagramm zur Veranschaulichung zeichnen.

Aufgabe 7 — 10 Punkte

Es sei ein Array a definiert:

```
#define SIZE 10

double a[SIZE] = {0};
```

Weisen Sie jedem zweiten Element des Arrays a den Wert 7 zu und benutzen Sie dabei:

- eine `while` Schleife und Index Notation; (5)
- eine `for` Schleife und Zeiger Notation. (5)

Aufgabe 8 — 5 Punkte

Was gibt das folgende Programm aus? Begründen Sie bitte.

```
#include <stdio.h>

int a = 1;
static int b = 2;

void func(int d, int e, int f)
{
  d = 3;
  e = 2;
  f = 1;
}

int main(void)
{
  int c = 3;

  func(a, b, c);

  printf("a␣=␣%d\n", a);
  printf("b␣=␣%d\n", b);
  printf("c␣=␣%d", c);

  return 0;
}
```

Aufgabe 9 — 5 Punkte

Es sei der folgende Abschnitt von C Code gegeben.

```
#define SIZE 10

int a[SIZE] = {0};
int *pi;

pi = a;
*pi++;
pi++;

char *pc = (char *) pi;
pc   = sizeof(int);

pi = (int *) pc;
```

Auf welches Element des Arrays zeigt der Zeiger `pi` am Ende dieses Abschnitts? Begründen Sie ihre Antwort.

Aufgabe 10 — 10 Punkte

Beschreiben Sie *detailliert* was im folgenden Programm passiert.

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <sys/wait.h>
5
6  int main(void)
7  {
8      pid_t  pid;
9      int status;
10
11     if ((pid = fork()) < 0) {
12         perror("fork :");
13         exit(1);
14     } else if (pid == 0) {
15         execlp("ls", "ls", " l", NULL);
16         perror("execle:");
17         exit(1);
18     }
19     if (wait(&status) < 0){
20         perror("wait :");
21         exit(1);
22     } else if (WIFEXITED(status)) {
23         printf("Child exit status was: %d", WEXITSTATUS(status));
24     }
25
26     exit(0);
27 }
```

Bonus Aufgabe — 3 Punkte

Gegeben sei folgendes C Programm:

```
int global = 0;
typedef struct list {
        struct list *next;
        double val;
} *list;

void function(list l) {
        list p;
        for (p = l; p; p = p >next)
                if (p >val > 0.0)
                        ++global;
}
```

a) Was macht die Funktion function(list l)? (1)

b) Welche Probleme können auftreten, wenn zwei parallel laufende Threads die se Funktion aufrufen und ihr jeweils eine Liste übergeben, die ausschließlich negative Zahlen enthält? (2)