8 Quest

8.1 Aufgabe

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 100
#define size 500

int main(int argc, char **argv[])
{
  int a[size][size], b[size][size], result[size][size], h, i, j, k;
  double time;
  clock_t begin, end;

  srand(1);
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
    {
      a[i][j] = rand() % max;
      b[i][j] = rand() % max;
    }

  begin = clock(); /* start stopping time */
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
    {
      h = 0;
      for (k = 0; k < size; k++)
        h += a[i][k] * b[k][j];
      result[i][j] = h;
    }

  end = clock();  /* stop stopping time */
  time = (double) (end - begin)/(double) CLOCKS_PER_SEC;

  printf("result:\n\n");
  for (i = 0; i < size; i++)
  {
    for (j = 0; j < size; j++)
      printf("%5d ", result[i][j]);
    printf("\n");
  }

  printf("time needed for calculation: %g seconds\n", time);
  exit(0);
}
```

8.2 Aufgabe

Die Anwendung des Algorithmus von Dijkstra zeigt, dass der gegebene Zustand sicher ist.

1. (0 0 1 0) an Prozess A ⇒ Prozess A terminiert ⇒ A = (3 0 2 2)
2. (1 0 1 1) an Prozess D ⇒ Prozess D terminiert ⇒ A = (5 1 2 2)
3. (2 1 1 0) an Prozess E ⇒ Prozess E terminiert ⇒ A = (5 1 3 2)
4. (3 1 0 0) an Prozess C ⇒ Prozess C terminiert ⇒ A = (6 2 4 2)
5. (0 1 1 2) an Prozess B ⇒ Prozess B terminiert ⇒ A = (6 3 4 2)

Allerdings kann ein Deadlock etwa wie folgt eintreten:

1. (0 0 1 0) an Prozess A ⇒ Prozess A terminiert ⇒ A = (3 0 2 2)
2. (0 0 1 2) an Prozess B ⇒ warte auf Ressourcen ⇒ A = (3 0 1 0)
3. (3 0 0 0) an Prozess C ⇒ warte auf Ressourcen ⇒ A = (0 0 1 0)
4. (0 0 1 0) an Prozess D ⇒ warte auf Ressourcen ⇒ A = (0 0 0 0)

Hierbei wurde die folgende Strategie verfolgt:
“Prozesse sequentiell abarbeiten, jeder Prozess nimmt alle benötigten Betriebsmittel, die vorhanden sind”

8.3 Aufgabe

Unter Fragmentierung versteht man die Zerstückelung des freien Adressraums in kleine Bereiche.
Bei externer Fragmentierung treten Lücken im benutzten Speicherbereich innerhalb eines Adressraums auf. Sie entstehen dadurch, dass die angeforderte Speichergröße oft nicht der Größe der freien Speicherbereie entsprechen.
Interne Fragmentierung ergibt sich durch die Festlegung von Blockgrößen. Wenn beispielsweise eine Festplatte die Blockgröse 512 Byte hat, so beansprucht die Speicherung eines einzelnen Bytes einen solchen Block, die restlichen 511 Byte bleiben ungenutzt, werden jedoch blockiert.

Man spricht auch von Fragmentierung, wenn eine große Datei nicht in einem zusammenhängenden Bereich gespeichert wird, sondern dieser in mehrere, kleinere Blöcke aufgeteilt ist.

8.4 Aufgabe

$$t = \text{Suchzeit} + \text{Latenz} + \text{Übertragungszeit} \approx 0.016s + 0.007s + \frac{60 \cdot 1024}{1.5 \cdot 1024^2}s = 0.062065s$$

8.5 Aufgabe

1. First Fit: 3 Speicherplatzanforderungen können nicht erfüllt werden (20, 25, 15)
2. Best Fit: 2 Speicherplatzanforderungen können nicht erfüllt werden (20, 25)
3. Worst Fit: 3 Speicherplatzanforderungen können nicht erfüllt werden (25, 25, 15)

	B1	B2	B3	B4	B5	B6	B7
1.	30 - 20 - 0	20 - 5	30 - 10 - 0	40 - 15 - 0	60 - 60 - 20 - 5	40 - 10	50 - 25 - 5
2.	30 - 10 - 0	20 - 10	30 - 15 - 0	40 - 20 - 0	60 - 35 - 15 - 0	40 - 15 - 0	50 - 30 - 0
3.	30 - 10	20 - 5	30 - 10	40 - 20 - 0	60 - 50 - 30 - 0	40 - 15 - 5	50 - 35 - 20 - 0

8.6 Aufgabe

Der Translation Lookaside Buffer (TLB) speihert die physischen Adressen von gewissen virtuellen Seiten. Bei jedem Zugriff auf eine virtuelle Adresse wird dann kontrolliert, ob die zugehörige virtuelle Seite im TLB gespeichert ist. Falls dies der fall ist, so kann die Seitenadresse aus dem TLB gelesen werden. Im anderen Fall muss die Adreßabbildung ausgeführt werden. Es ist oft ratsam diese virtuelle Seite in den TLB aufzunehmen nachdem eine andere virtuelle Seite (etwa nach LRU) aus dem TLB zu entfernt wurde.