Betriebssysteme (BS)

Probeklausur

Olaf Spinczyk
Arbeitsgruppe Eingebettete Systemsoftware

Lehrstuhl für Informatik 12
TU Dortmund

http://ess.cs.uni-dortmund.de/

Ablauf

- Probeklausur (45 Minuten)
- Besprechung der Aufgaben
- Auswertung
- Weitere Hinweise zur Vorbereitung

Probeklausur

... in (fast) allen Belangen realistisch:

- Art der Aufgaben
  - Auswahl aus dem **gesamten** Inhalt der Veranstaltung
    - Betriebssystemgrundlagen **und** UNIX-Systemprogrammierung in C
    - alle Vorlesungen und Übungen sind relevant
- Umfang
  - kürzer als das „Original“: ca. 40–45 (statt 60) Minuten
- Durchführung
  - **keine Hilfsmittel** erlaubt (keine Spickzettel, Bücher, ...)
  - bitte **still arbeiten**
  - jeder für sich
- Die Klausur wird **nicht** eingesammelt.

1 – Prozesse und Scheduling

a) UNIX Shell-Operatoren (insgesamt 4 Punkte)

1. „|“ Operator (1,5 Punkte) Beschreiben Sie die Funktionsweise des „|“ Operators.

Der Pipe „|“ Operator verbindet die Standardausgabe eines Prozesses mit der Standardeingabe eines weiteren Prozesses nach dem FIFO-Prinzip.

2. „|“ Operator – Beispiel (1 Punkt) Geben Sie für die Einsatzmöglichkeiten des „|“ Operators ein Beispiel an.

ls -R | grep jpg | wc

1 – Prozesse und Scheduling

a) UNIX Shell-Operatoren (insgesamt 4 Punkte)

3. „<“ **Operator (1,5 Punkte)** Beschreiben Sie die Funktionsweise des „<“ Operators.

Es handelt sich um die Umlenkung der Eingabe für einen Prozess. So ist es z. B. möglich eine Eingabe aus einer Datei statt von der Tastatur zu lesen.

1 – Prozesse und Scheduling

b) Prozesse (2 Punkte) Erklären Sie knapp in eigenen Worten den Unterschied zwischen den Begriffen *Programm* und *Prozess*.

Ein Programm ist eine, nach bestimmten Regeln formulierte, Anweisungsvorschrift.

Ein Prozess ist ein Programm in Ausführung

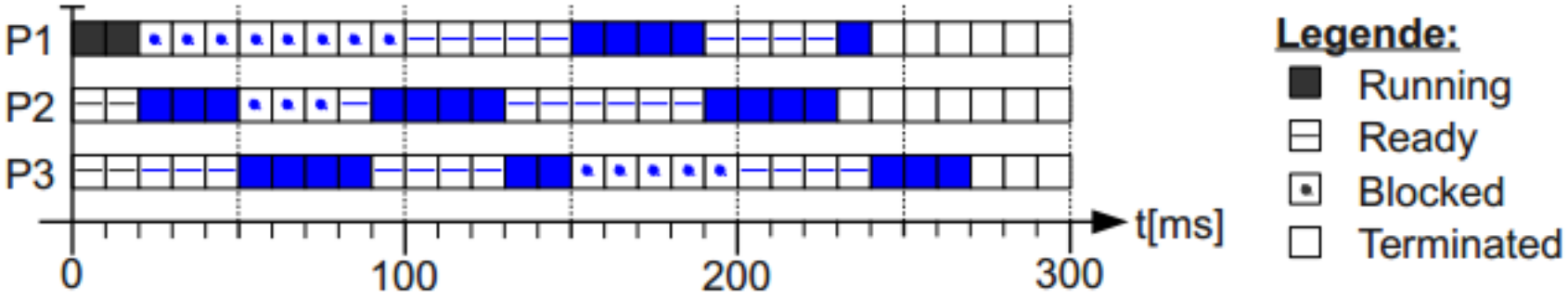
- dynamisch
- Folge von Rechnen und Ein-/Ausgabe
- benötigt Betriebsmittel des Rechners
- wird vom BS kontrolliert

1 – Prozesse und Scheduling

c) Round Robin (6 Punkte) Ein Betriebssystem verwaltet drei Prozesse P1, P2 und P3. Die Prozesse treffen in dieser Reihenfolge im System ein und sind alle zum Zeitpunkt t=0 rechenbereit. Die Bedienzeiten (in ms) der Prozesse und die Zeitpunkte von E/A-Operationen, relativ zur Bedienzeit*, sind in der folgenden Tabelle angegeben:

Prozess	P1	P2	P3
Bedienzeit	70	110	90
E/A-Zeitpunkt	20	30	60
E/A-Dauer	80	30	50

*: Bei der „Bedienzeit" handelt es sich um die reine Rechenzeit. Prozess P1 hat seinen ersten E/A-Zugriff nachdem er 20 Zeiteinheiten gerechnet hat. Prozess P2 wird blockiert nachdem er seine ersten 30 Zeiteinheiten gerechnet hat etc.



2 – Synchronisation & Verklemmung

a) Verklemmungen (6 Punkte) Wenn eine geschlossene Kette wechselseitig wartender Prozesse existiert (*circular wait*, also ein Zyklus im Betriebsmittelbelegungsgraphen), liegt eine Verklemmung vor. Nennen Sie stichpunktartig die drei *Vorbedingungen*, die erfüllt sein müssen, damit es überhaupt zu einer Verklemmung kommen kann, und erklären Sie diese jeweils kurz mit eigenen Worten.

- **mutual exclusion**
  die umstrittenen Betriebsmittel sind nur unteilbar nutzbar
- **hold and wait**
  die Prozesse fordern Betriebsmittel an, behalten aber zugleich den Zugriff auf andere
- **no preemption**
  umstrittene Betriebsmittel sind nicht rückforderbar, können nur durch den Prozess freigegeben werden

2 – Synchronisation & Verklemmung

b) Synchronisation (3 Punkte) Welche der folgenden Aussagen zu Synchronisation und Verklemmungen treffen zu? Kreuzen Sie bei **J** (ja, zutreffend) oder **N** (nein, nicht zutreffend) an!

J	N	
	X	Semaphoren basieren auf dem Prinzip des aktiven Wartens.
X		Mit Semaphoren kann gegenseitiger Ausschluss realisiert werden.
	X	Der Bäckerei-Algorithmus ist ein Verfahren zur Vermeidung von Verklemmungen.
X		Wenn Betriebsmittel nicht exklusiv belegt werden, können keine Verklemmungen entstehen.
X		Prozesse in einem *Deadlock* befinden sich im Prozesszustand *blockiert*.
	X	Unterbrechungsanforderungen zählen zu den wiederverwendbaren Betriebsmitteln.

- Semaphoren, basieren auf dem Prinzip des *passivem Wartens*
- Der Bäckerei-Algorithmus kann *nicht* garantieren, dass eine Wartenummer nur an einen Prozess vergeben wird
- Unterbrechungsanforderungen gehören zu den *konsumierbaren* Betriebsmitteln

2 – Synchronisation & Verklemmung

c) Synchronisation (3 Punkte) Betrachten Sie das folgende Codefragment, in dem eine Schlossvariable implementiert wird (Datentyp `Lock` mit Operationen `acquire` und `release`). Warum ist dieses *naive* Verfahren zur Synchronisierung von Prozessen (bei präemptivem Scheduling) nicht geeignet? Welche gängige Betriebssystem-Abtraktion sollte man stattdessen verwenden?

```
1  /* Schlossvariable (Initialwert 0) */
2  typedef unsigned char Lock;
3
4  /* Kritischen Abschnitt betreten */
5  void acquire (Lock *lock) {
6      while (*lock);
7      *lock = 1;
8  }
9
10 /* Kritischen Abschnitt wieder verlassen */
11 void release (Lock *lock) {
12     *lock = 0;
13 }
```

- acquire() ist nicht atomar und damit selbst kritisch
- aktives Warten verschwendet CPU-Zeit
- Gängiger Mechanismus: Semaphoren

2 – Synchronisation & Verklemmung

d) Verklemmungs-Arten (3 Punkte) Erklären Sie stichpunktartig den Unterschied zwischen einem *Deadlock* und einem *Livelock*. Weshalb sind Deadlocks das „geringere Übel"?

- **Deadlock**
  - passives Warten (gut)
  - Prozesszustand blocked
- **Livelock**
  - aktives Warten (schlecht)
  - alle Prozesszustände möglich (auch RUNNING)
  - kein Fortschritt
- Deadlocks sind erkennbar, es existiert eine Basis zur Auflösung

3 – Speicherverwaltung & virt. Speicher

a) Buddy-Verfahren (4 Punkte) Dynamische Speicherverwaltung nach dem *Buddy*-Verfahren: Die jeweils zweite Zeile der folgenden Szenarien zeigt die momentane Speicherbelegung des Speichers der Größe 32 MiB. Ergänzen Sie die folgenden Tabellen um Markierungen für die vorgegebenen Anfragen. **Hinweis:** Falls eine Belegung/Freigabe *nicht* erfüllt werden kann, kennzeichnen Sie die betreffende Zeile geeignet.

Szenario 1: Prozess C belegt 3 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	A	A					**C**	**C**	B	B				

Buddy-Verfahren: auf nächste 2er-Potenz aufrunden!

3 – Speicherverwaltung & virt. Speicher

a) Buddy-Verfahren (4 Punkte) Dynamische Speicherverwaltung nach dem *Buddy*-Verfahren: Die jeweils zweite Zeile der folgenden Szenarien zeigt die momentane Speicherbelegung des Speichers der Größe 32 MiB. Ergänzen Sie die folgenden Tabellen um Markierungen für die vorgegebenen Anfragen. **Hinweis:** Falls eine Belegung/Freigabe *nicht* erfüllt werden kann, kennzeichnen Sie die betreffende Zeile geeignet.

Szenario 2: Prozess D belegt 12 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A							**D**	**D**	**D**	**D**	**D**	**D**	**D**	**D**

3 – Speicherverwaltung & virt. Speicher

a) Buddy-Verfahren (4 Punkte) Dynamische Speicherverwaltung nach dem *Buddy*-Verfahren: Die jeweils zweite Zeile der folgenden Szenarien zeigt die momentane Speicherbelegung des Speichers der Größe 32 MiB. Ergänzen Sie die folgenden Tabellen um Markierungen für die vorgegebenen Anfragen. **Hinweis:** Falls eine Belegung/Freigabe *nicht* erfüllt werden kann, kennzeichnen Sie die betreffende Zeile geeignet.

Szenario 3: Prozess E belegt 14 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
		B	B									A	A		

kann nicht erfüllt werden – keine 16 MiB am Stück verfügbar

3 – Speicherverwaltung & virt. Speicher

a) Buddy-Verfahren (4 Punkte) Dynamische Speicherverwaltung nach dem *Buddy*-Verfahren: Die jeweils zweite Zeile der folgenden Szenarien zeigt die momentane Speicherbelegung des Speichers der Größe 32 MiB. Ergänzen Sie die folgenden Tabellen um Markierungen für die vorgegebenen Anfragen. **Hinweis:** Falls eine Belegung/Freigabe *nicht* erfüllt werden kann, kennzeichnen Sie die betreffende Zeile geeignet.

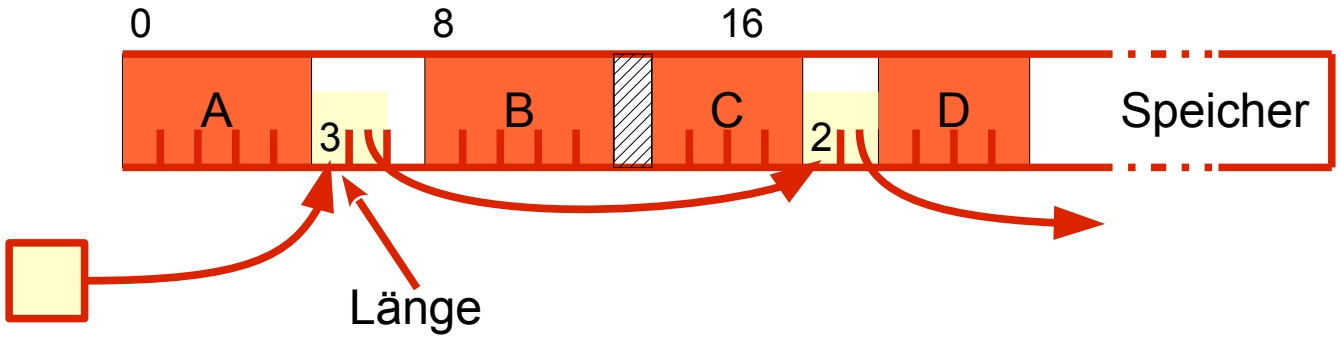
Szenario 4: Prozess F belegt 7 MiB

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
A	A	A	A	B	B			**F**	**F**	**F**	**F**				

3 – Speicherverwaltung & virt. Speicher

b) Verschnitt (4 Punkte) Bei den behandelten Speicherverwaltungs-Platzierungsstrategien kann es zu *externem* und *internem Verschnitt* kommen. Erklären Sie die beiden Begriffe stichpunktartig und nennen Sie jeweils eine Platzierungsstrategie (nicht zwei Mal dieselbe!), bei der das Problem auftritt.

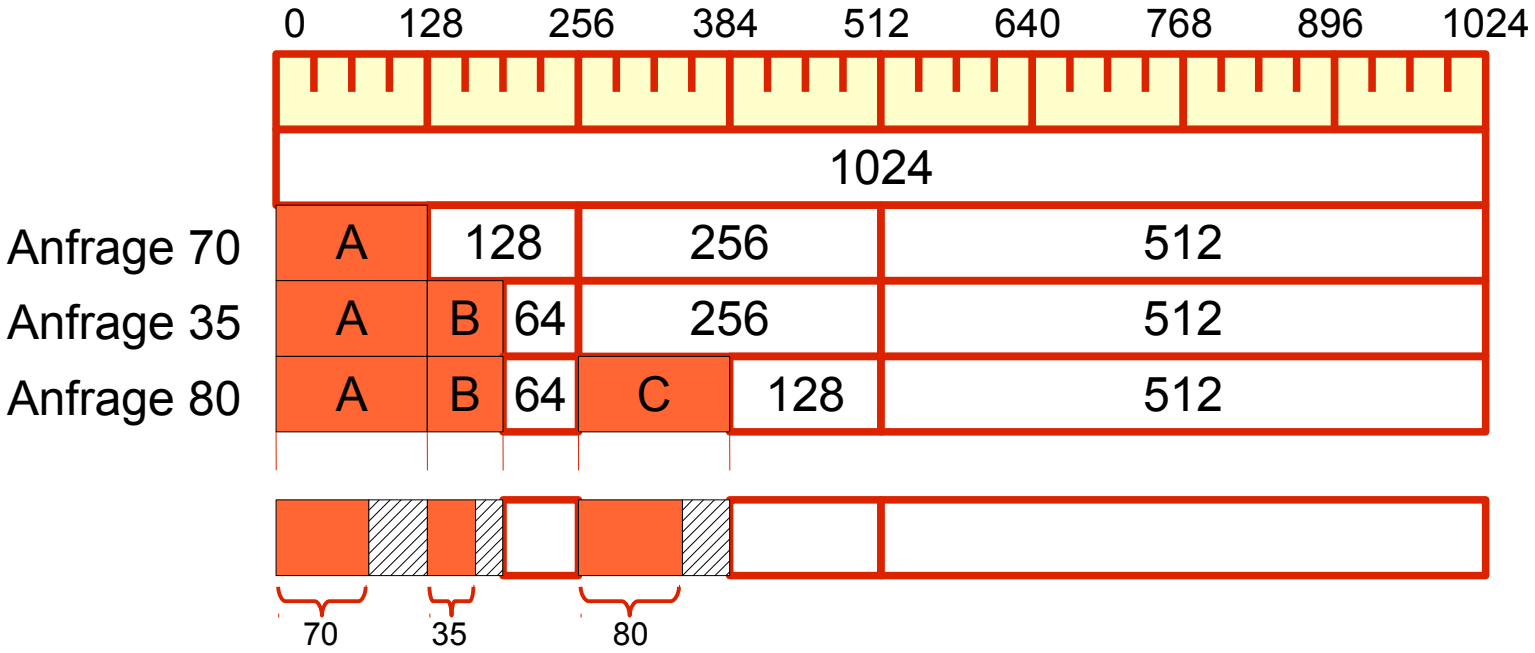
- **Externer Verschnitt**
  - Außerhalb der zugeteilten Speicherbereich entstehen Speicherfragmente, die nicht mehr genutzt werden können.
  - Passiert bei den listenbasierten Strategien wie ***First Fit***, ***Best Fit***, **...**



3 – Speicherverwaltung & virt. Speicher

Interner Verschnitt

- Innerhalb der zugeteilten Speicherbereiche gibt es ungenutzten Speicher.
- Passiert z.B. bei ***Buddy***, da die Anforderungen auf die nächstgrößere Zweierpotenz aufgerundet werden.



4 – Ein-/Ausgabe und Dateisysteme

a) Dateioperationen (6 Punkte) Beantworten Sie die folgenden Fragen zu dem kurzen C-Programm. #include der Systemheader und Fehlerabfragen wurden der Einfachheit halber weggelassen. Gehen Sie von einer fehlerfreien Abarbeitung aller Systemaufrufe aus.

Die Datei datei.txt hat folgenden Inhalt:

```
Live␣for␣today!
```

```
1  int main() {
2      struct stat stats;
3      FILE *filestream;
4      char *block="****";
5
6      stat("datei.txt", &stats);
7      if(!S_ISREG(stats.st_mode)) return 1;
8
9      filestream = fopen("datei.txt", "r+");
10     fwrite(block, 2, 1, filestream);
11     fseek(filestream, -5, SEEK_END);
12     fwrite(block + 2, 2, 1, filestream);
13     fclose(filestream);
14
15     return 0;
16 }
```

1. stat (2 Punkte) Wozu dienen die Zeilen 6 und 7?

Hier wird überprüft, ob es sich bei „datei.txt“ um eine reguläre Datei handelt. Falls „datei.txt“ keine reguläre Datei ist, bricht das Programm ab.

4 – Ein-/Ausgabe und Dateisysteme

a) Dateioperationen (6 Punkte) Beantworten Sie die folgenden Fragen zu dem kurzen C-Programm. #include der Systemheader und Fehlerabfragen wurden der Einfachheit halber weggelassen. Gehen Sie von einer fehlerfreien Abarbeitung aller Systemaufrufe aus.

Die Datei `datei.txt` hat folgenden Inhalt:

```
Live␣for␣today!
```

```
1  int main() {
2      struct stat stats;
3      FILE *filestream;
4      char *block="****";
5
6      stat("datei.txt", &stats);
7      if(!S_ISREG(stats.st_mode)) return 1;
8
9      filestream = fopen("datei.txt", "r+");
10     fwrite(block, 2, 1, filestream);
11     fseek(filestream, -5, SEEK_END);
12     fwrite(block + 2, 2, 1, filestream);
13     fclose(filestream);
14
15     return 0;
16 }
```

2. fseek (2 Punkte) Beschreiben Sie die genaue Funktion des `fseek`-Aufrufs in Zeile 11.

Dieser Aufruf setzt die Schreib-/Leseposition, in dem angegebenen Stream vom Ende der aus Datei, um 5 Bytes zurück.

4 – Ein-/Ausgabe und Dateisysteme

a) Dateioperationen (6 Punkte) Beantworten Sie die folgenden Fragen zu dem kurzen C-Programm. #include der Systemheader und Fehlerabfragen wurden der Einfachheit halber weggelassen. Gehen Sie von einer fehlerfreien Abarbeitung aller Systemaufrufe aus.

Die Datei `datei.txt` hat folgenden Inhalt:

```
Live␣for␣today!
```

```
1  int main() {
2      struct stat stats;
3      FILE *filestream;
4      char *block="**##";
5
6      stat("datei.txt", &stats);
7      if(!S_ISREG(stats.st_mode)) return 1;
8
9      filestream = fopen("datei.txt", "r+");
10     fwrite(block, 2, 1, filestream);
11     fseek(filestream, -5, SEEK_END);
12     fwrite(block + 2, 2, 1, filestream);
13     fclose(filestream);
14
15     return 0;
16 }
```

3. Textdatei (2 Punkte) Geben Sie an, was nach der Ausführung des Programms in der Datei `datei.txt` steht.

**ve for t##ay!

4 – Ein-/Ausgabe und Dateisysteme

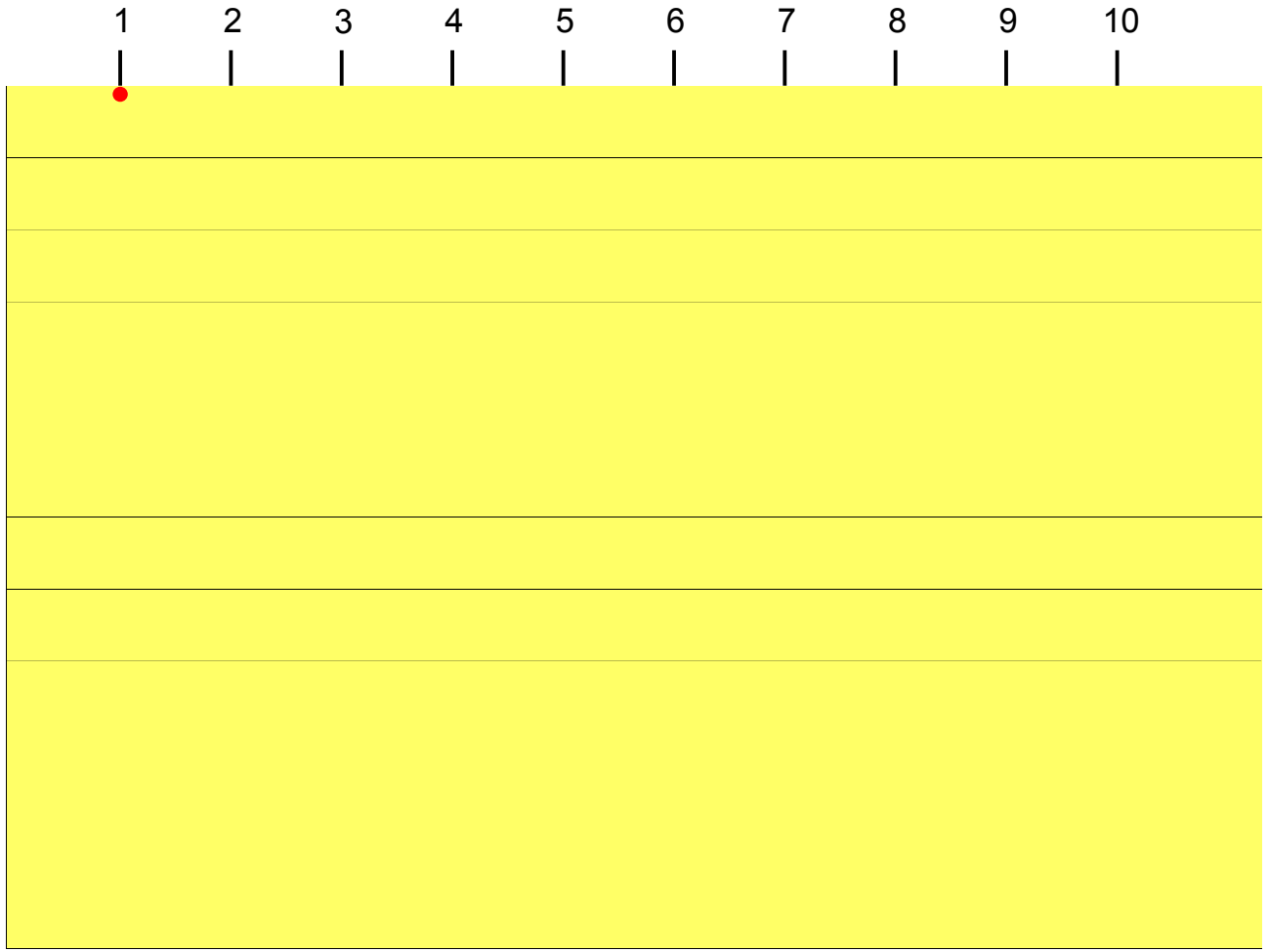
b) E/A-Scheduling (4 Punkte) Erläutern Sie die E/A-Scheduling-Verfahren „Elevator“ und „Shortest Seek Time First“ (SSTF). Welches Problem besteht bei SSTF im Gegensatz zu „Elevator“? Erklären Sie beispielhaft wie es zu dem Problem kommen kann.

- a) E/A-Scheduling
  - ***Shortest Seek Time First***:
    - Auftrag mit der **kürzesten Positionierzeit** wird vorgezogen
  - ***Elevator***:
    - **Plattenarm** bewegt sich in **eine Richtung** bis keine Aufträge mehr vorhanden sind
  - Problem bei *SSTF* im Gegensatz zu *Elevator*?
    - eine Anfrage könnte „nie“ beantwortet werden, falls immer eine Anfrage eintrifft, bei der die Positionierzeit kürzer ist

      → **Aushungerung**

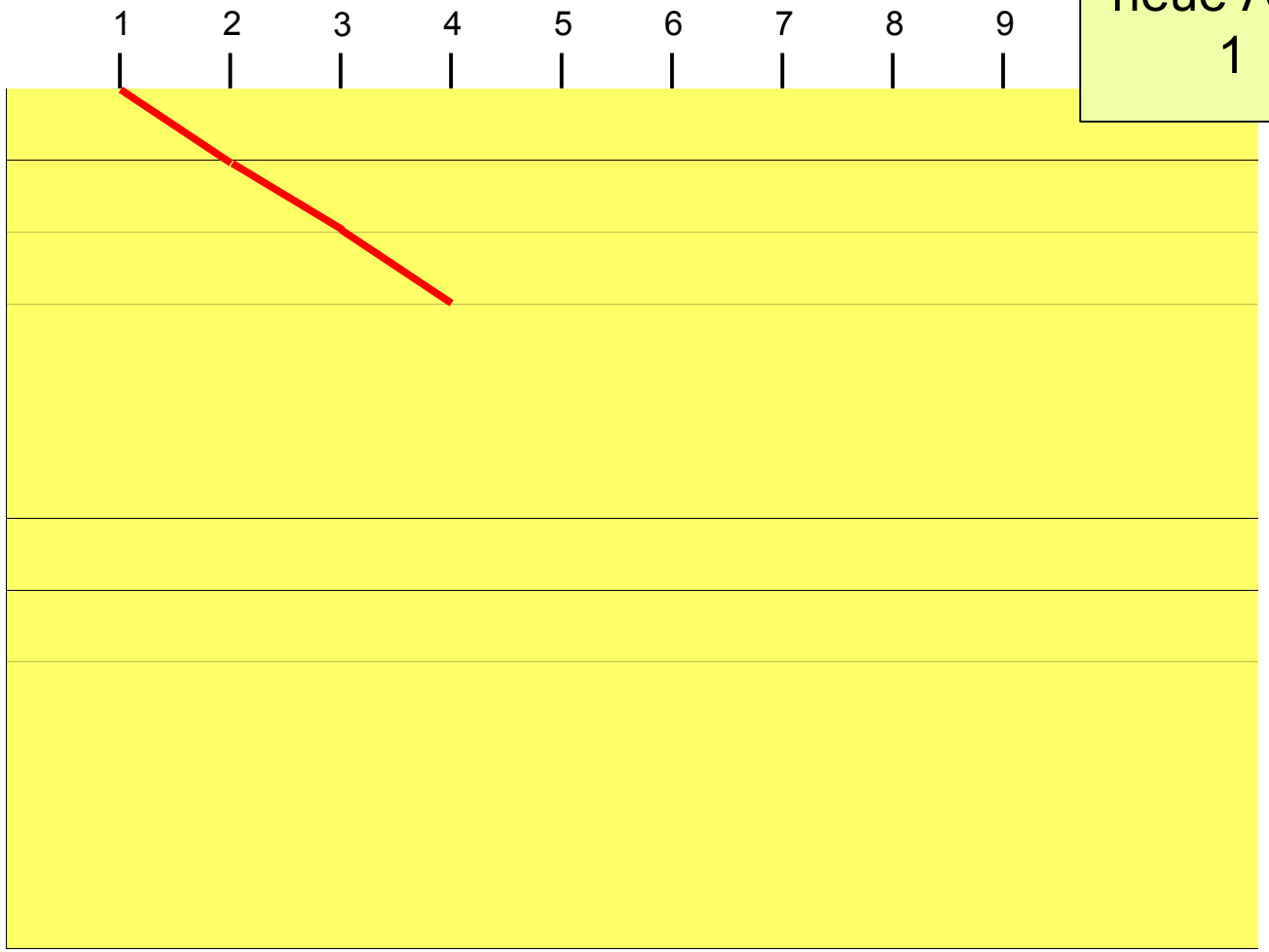
4 – Ein-/Ausgabe und Dateisysteme

- a) E/A-Scheduling
  - Beispiel für *SSTF – Referenzfolge*: 2 3 4 10



4 – Ein-/Ausgabe und Dateisysteme

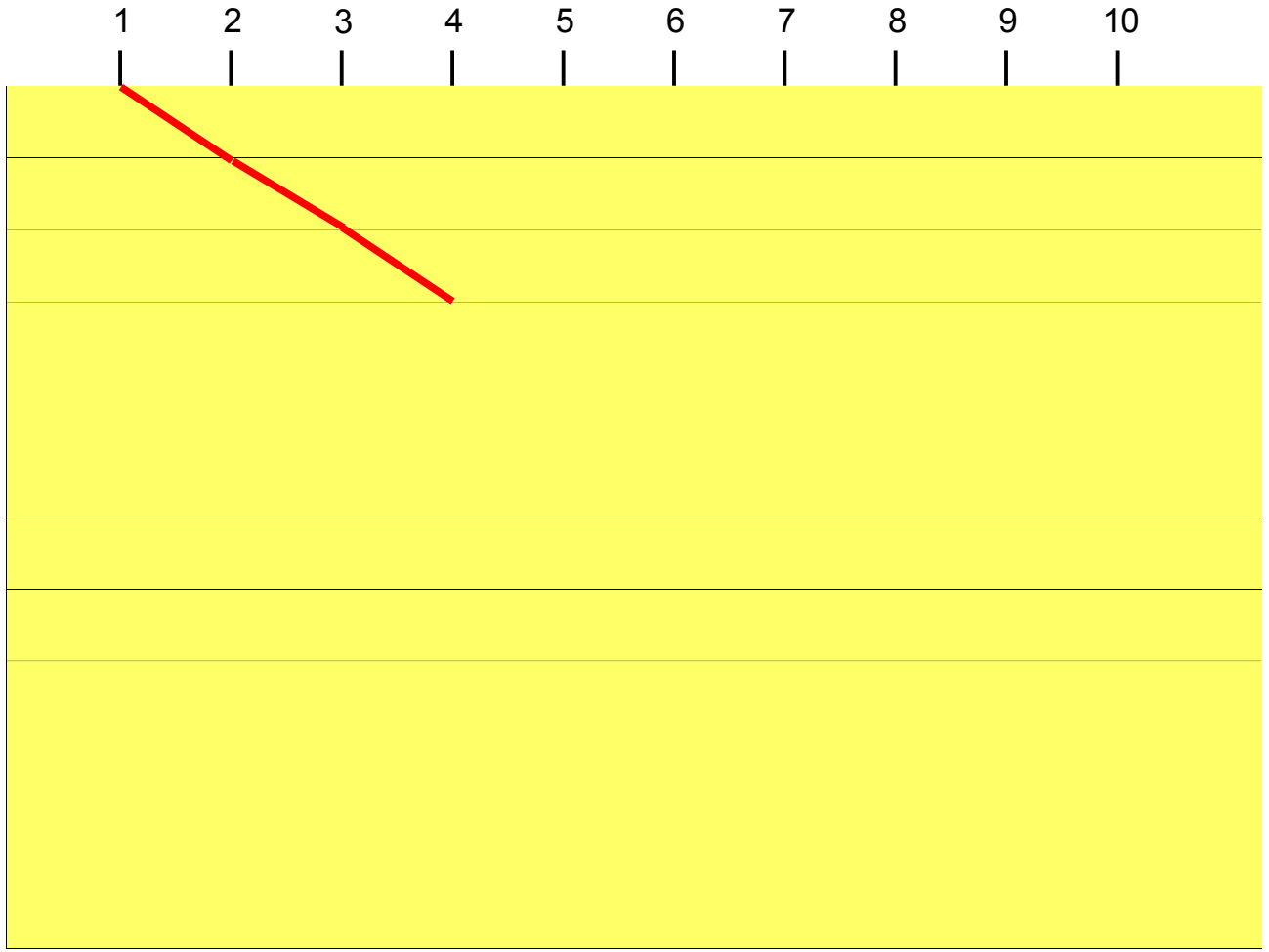
- a) E/A-Scheduling
  - Beispiel für *SSTF – Referenzfolge*: 10



neue Anfragen:
1 2 3

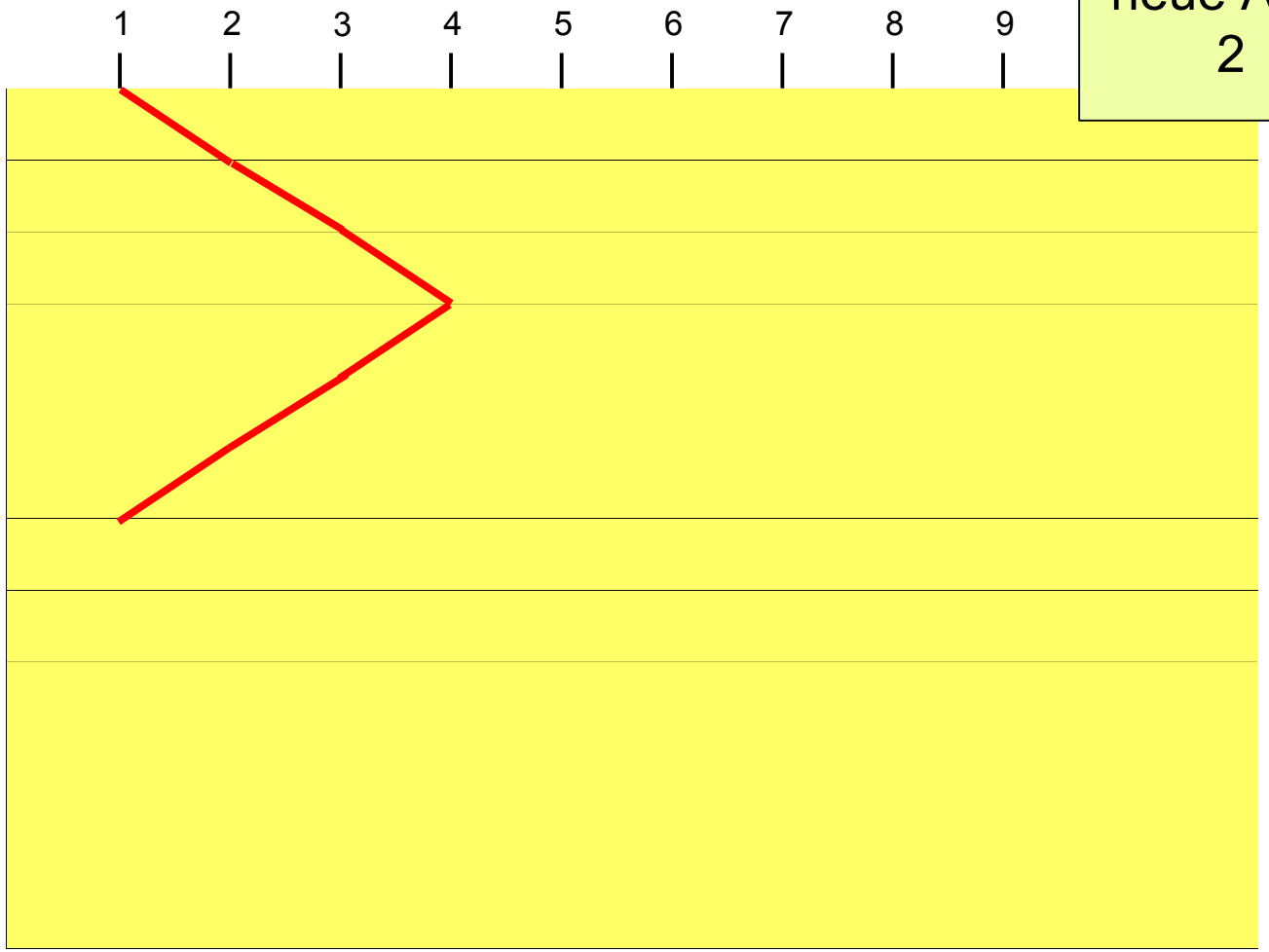
4 – Ein-/Ausgabe und Dateisysteme

- a) E/A-Scheduling
  - Beispiel für *SSTF – Referenzfolge*: 3 2 1 10



4 – Ein-/Ausgabe und Dateisysteme

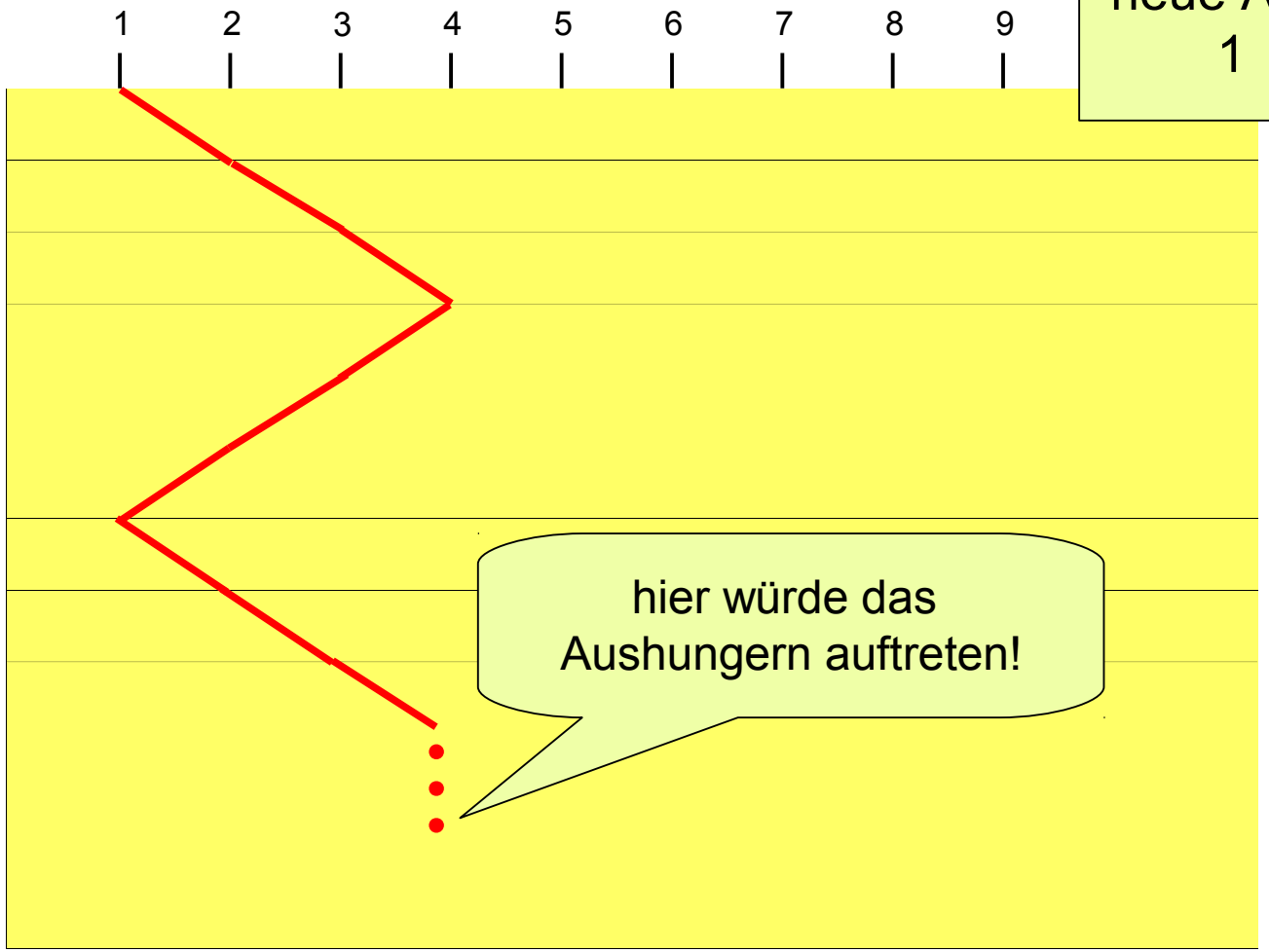
- a) E/A-Scheduling
  - Beispiel für *SSTF – Referenzfolge*: 10



neue Anfragen:
2 3 4

4 – Ein-/Ausgabe und Dateisysteme

- a) E/A-Scheduling
  - Beispiel für *SSTF – Referenzfolge*: 10



Auswertung

- Bitte schnell einmal die Punkte zusammenzählen …
- Notenspiegel:

Punkte	Note
38,5–45	1
33,5–38	2
28–33	3
22,5–27,5	4
0–22	5

Weitere Hinweise zur Vorbereitung

- Inhalt der Folien lernen
  - Klassifizieren: Was muss ich **lernen**? Was muss ich **begreifen**?
- Übungsaufgaben verstehen, C und UNIX „können“
  - ASSESS-System bleibt mindestens bis zur Klausur offen
    - bei Fragen zur Korrektur melden
  - Am besten die Aufgaben noch einmal lösen
  - Optionale Zusatzaufgaben bearbeiten
- Literatur zur Lehrveranstaltung durchlesen
- BS-Forum nutzen

Empfohlene Literatur

[1] A. Silberschatz et al. *Operating System Concepts*. Wiley, 2004. ISBN 978-0471694663

[2] A. Tanenbaum: Modern Operating Systems (2nd ed.). Prentice Hall, 2001. ISBN 0-13-031358-0

[3] B. W. Kernighan, D. M. Ritchie. *The C Programming Language*. Prentice-Hall, 1988. ISBN 0-13-110362-8 (paperback) 0-13-110370-9 (hardback)

[4] R. Stevens, *Advanced Programming in the UNIX Environment*, Addison-Wesley, 2005. ISBN 978-0201433074

Viel Erfolg bei der Klausur!