Interaktionsdiagramme

Aufgabe K1: Interaktionsmuster Client/Server

Erstellen Sie Interaktionsdiagramme nach den folgenden Beschreibungen.
Jede Teilaufgabe erweitert die vorhergehende.

a) Zwei Clients beauftragen einen Server A. Ein Client schickt seinen Auftrag asynchron, der andere synchron.

b) Server A schickt einen asynchronen Auftrag an einen Server B. Sowohl Senden als auch Empfangen erfolgen in der Arbeitsphase von Server A.

c) Server B bearbeitet Aufträge mit zwei parallelen Prozessen. (Reproduktion)

d) Zwei weitere Clients beauftragen den Server B. Ein Client schickt seinen Auftrag synchron, der andere asynchron. *Tipp: Diese gegenüberliegend zu Teil a zeichnen.*

e) Server A bearbeitet Aufträge ebenfalls mit zwei reproduzierten Prozessen.

Aufgabe K2: Interaktionsmuster Erzeuger/Verbraucher

Erstellen Sie Interaktionsdiagramme nach den folgenden Beschreibungen.

a) In einem Erzeuger–Verbraucher–System zirkulieren p Nachrichten. Die Initialisierung übernimmt der Verbraucher. Sowohl Erzeuger als auch Verbraucher beauftragen in ihrer Arbeitsphase je einmal einen gemeinsamen Server. Der Erzeuger schickt seinen Auftrag asynchron, der Verbraucher nicht.

b) Erzeuger und Verbraucher aus Teilaufgabe a sind reproduziert.

   Tipp: Erstellen Sie eine neue Zeichnung. Übernehmen Sie jeweils einen Prozess mit den Kanälen, die er aufruft. Reproduzieren Sie diesen Prozess. Wählen Sie als nächstes einen Prozess, der einen bereits übernommenen Kanal aufruft.

c) Welches Problem gibt es im Interaktionsmuster aus Teilaufgabe b?
   Was könnte man dagegen tun? *Ohne Zeichnung!*

d) Mehrere Clients schicken jeweils einen Auftrag und erwarten dann die Antwort. Anstelle eines Servers bearbeitet ein Erzeuger–Verbraucher–System die Aufträge. Der Erzeuger empfängt sie und beginnt mit der Bearbeitung. Der Verbraucher schließt die Bearbeitung ab und schickt die Antworten. Im Erzeuger–Verbraucher–System zirkulieren p Nachrichten, die Initialisierung übernimmt der Erzeuger.

e) Erzeuger und Verbraucher aus Teilaufgabe d sind reproduziert.

Aufgabe K3: Interaktionsmuster Fließband

Erstellen Sie Interaktionsdiagramme nach den folgenden Beschreibungen.

a) In einem dreistufigen, geschlossenen Fließband A zirkulieren p Nachrichten. Die erste Stufe übernimmt die Initialisierung. Die zweite Stufe schickt pro Durchgang einen asynchronen Auftrag und empfängt die Antwort. Diese Aufträge bearbeitet ein zweistufiges, offenes Fließband B.

b) Jeder Prozess aus Teilaufgabe a ist reproduziert.

Tipp: Erstellen Sie eine neue Zeichnung. Übernehmen Sie jeweils einen Prozess mit den Kanälen, die er aufruft. Reproduzieren Sie diesen Prozess. Wählen Sie als nächstes einen Prozess, der einen bereits übernommenen Kanal aufruft.

c) Der implementierte Algorithmus erlaubt es, das Empfangen der Antworten in die dritte Stufe zu verschieben. Die Antworten müssen weiterhin zum Auftrag passen. Ist das technisch möglich? Falls ja, wie? Falls nein, warum nicht? *Ohne Zeichnung!*

Aufgabe K4: Interaktionsmuster Pufferung

Erstellen Sie Interaktionsdiagramme nach den folgenden Beschreibungen.

a) Ein Client schickt mit unbeschränkter Pufferung n Aufträge an einen Server, der mit k Prozessen reproduziert ist. Ein anderer Client schickt m Aufträge, gepuffert mit Limit p, an den gleichen Server.

b) Beide Clients aus der ersten Teilaufgabe werden aufgeteilt, jeweils in zwei Prozesse. Der erste Prozess übernimmt das Erstellen und Senden der Aufträge, der zweite das Empfangen und Verarbeiten der Antworten. Der Server wird zur mittleren Stufe in der Kette. Der zweite Client puffert weiterhin mit Limit p.

c) Wie unterscheiden sich die entstandenen Interaktionsmuster von Fließbändern?

Aufgabe K5: Interaktionsdiagramm

Erstellen Sie ein Interaktionsdiagramm nach der folgenden Beschreibung:

Ein Client sendet n Aufträge mit p-facher Pufferung und empfängt die Antworten. Die Aufträge bearbeitet ein zweistufiges, offenes Fließband. Die Stufen vergeben während der Bearbeitung je einen Unterauftrag an einen weiteren Server und empfangen dessen Antwort. Beide Stufen senden ihre Unteraufträge an den gleichen Server. Dieser Server ist mit insgesamt zwei Prozessen reproduziert.

Aufgabe K6: Interaktionsdiagramm

Erstellen Sie ein Interaktionsdiagramm nach der folgenden Beschreibung:

Ein Client A schickt drei Aufträge asynchron an ein dreistufiges Fließband A. Anschließend arbeitet er noch eine Weile und empfängt dann die Antworten. Ein Client B schickt n Aufträge gepuffert mit Limit p an ein dreistufiges Fließband B. Beide Fließbänder verwenden eine gemeinsame zweite Stufe, die mit insgesamt zwei Prozessen reproduziert ist.

Aufgabe K7: Interaktionsmuster Team

Erstellen Sie Interaktionsdiagramme nach den folgenden Beschreibungen.

a) Ein Client beauftragt asynchron ein Team mit Verteiler. Das Team bearbeitet zwei Arten von Aufträgen. Aufträge vom Typ A übernimmt ein mit zwei Prozessen reproduzierter Server. Für Aufträge vom Typ B ist ein zweistufiges, offenes Fließband zuständig.

   Tipp: Zeichnen Sie den Client links und lassen Sie rechts viel Platz für die zweite Teilaufgabe.

b) Das Team aus Teilaufgabe a wird um einen dritten Auftragstyp C ergänzt, den der Verteiler nicht kennt. Ein zweiter Client schickt mit einer neuen Bibliothek, welche ohne Verteiler arbeitet, einen asynchronen Auftrag an das Team. Aufträge vom Typ C bearbeitet ein Erzeuger–Verbraucher–System, das höchstens p Aufträge gleichzeitig annimmt. Es wird vom Verbraucher initialisiert.

Aufgabe K8: Interaktionsdiagramm

Erstellen Sie ein Interaktionsdiagramm nach der folgenden Beschreibung:

Ein Client sendet n Aufträge mit p-facher Pufferung an ein Team mit Verteiler und empfängt die Antworten. Das Team kennt zwei Auftragstypen. Typ A wird von zwei reproduzierten Prozessen bearbeitet, Typ B von einem offenen, zweistufigen Fließband. Ein anderer Client sendet m Aufträge an das gleiche Team, aber ohne Verteiler. Dieser Client empfängt die Antwort direkt nach dem Senden eines Auftrags.

Aufgabe K9: Interaktionsdiagramm

Erstellen Sie ein Interaktionsdiagramm nach der folgenden Beschreibung:

In einem Erzeuger/Verbraucher–System zirkulieren p Nachrichten. Die Initialisierung übernimmt der Erzeuger. Während der Bearbeitung einer Nachricht vergibt der Verbraucher n Aufträge mit unbeschränkter Pufferung und empfängt die Antworten. Diese Aufträge bearbeitet ein Team mit Verteiler. Es kennt zwei Auftragstypen. Typ A wird von zwei reproduzierten Prozessen bearbeitet, Typ B von einem zweistufigen Fließband.

Aufgabe K10: Interaktionsdiagramm

Erstellen Sie ein Interaktionsdiagramm nach der folgenden Beschreibung:

Ein Team mit Verteiler bearbeitet zwei Arten von Aufträgen, A und B. Die Aufträge vom Typ A übernehmen insgesamt zwei reproduzierte Prozesse. Für Aufträge vom Typ B ist ein zweistufiges, offenes Fließband zuständig. Ein Client schickt n Aufträge mit beschränkter Pufferung und Limit p an den Verteiler.
Ein Erzeuger/Verbraucher–System schickt ebenfalls Aufträge an das Team, aber ohne den Verteiler. Der Erzeuger schickt asynchron nur Aufträge vom Typ A, der Verbraucher synchron nur Aufträge vom Typ B. Beide senden und empfangen während ihrer Arbeitsphase je einmal. Im Erzeuger/Verbraucher–System zirkulieren q Nachrichten. Die Initialisierung übernimmt der Verbraucher.

Zeichenvorschlag: Team in der Mitte, Client links, Erzeuger/Verbraucher rechts.

Aufgabe K11: Interaktionsdiagramm

Erstellen Sie ein Interaktionsdiagramm nach der folgenden Beschreibung:

Ein Client schickt n Aufträge mit beschränkter Pufferung und Limit p an ein Team mit Verteiler. Dieses Team kennt zwei Auftragstypen, A und B. Aufträge vom Typ A bearbeiten insgesamt zwei reproduzierte Prozesse.
Aufträge vom Typ B übernimmt ein einzelner Prozess. Während seiner Arbeitsphase schickt er jeweils einen synchronen Auftrag an ein weiteres Team. Das zweite Team arbeitet ohne Verteiler. Es kennt zwei Auftragstypen, C und D. Aufträge vom Typ D übernimmt ein offenes Fließband mit zwei Stufen.
Aufträge vom Typ C bearbeitet ein einzelner Prozess. Während seiner Arbeitsphase schickt er jeweils einen asynchronen Auftrag und empfängt die Antwort. Dieser Auftrag ist vom Typ A und geht an das erste Team, ohne dessen Verteiler zu benutzen.

Ablaufbeschreibungen

Aufgabe K12: Zelle

Ein Kernobjekt vom Typ *Zelle* ist entweder leer oder enthält einen Wert. Die Operation `give` übergibt einen Wert an die Zelle. Enthält sie schon einen Wert, wird dieser überschrieben. Die Operation `take` nimmt den Wert aus einer Zelle und leert sie damit. Ist die Zelle bereits leer, muss der Aufrufer warten, bis er einen neu übergebenen Wert erhält. Die Implementierung des Kernobjekts verwendet folgende Attribute:

Flag, ob die Zelle einen Wert enthält. Initial gelöscht, d.h. Zelle leer.

Inhalt, der Wert des letzten Aufrufs von `give`. Nur gültig wenn das Flag gesetzt ist. Der initiale Wert spielt deshalb keine Rolle.

Wartemenge für Prozesse. Initial leer.

a) Welche ungültigen Fälle müssen ausgeschlossen werden?

b) Beschreiben Sie den Ablauf von `give`. Aufrufargument ist der Wert.

c) Beschreiben Sie den Ablauf von `take`. Ergebnis des Aufrufs ist der Wert.

Aufgabe K13: Rendezvous mit Wertaustausch

Ein Kernobjekt vom Typ *Rendezvous* ermöglicht je zwei Prozessen, sich zu synchronisieren und dabei einen Wert auszutauschen. Es bietet eine Operation `rendezvous` an, die synchron arbeitet. Sie erwartet einen Wert als Argument und gibt einen Wert als Ergebnis zurück. Der erste Prozess, der die Operation aufruft, muss auf den zweiten warten. Dann erfolgt der Austausch der Werte. Danach ist das Kernobjekt wieder im Ausgangszustand, bereit für das nächste Rendezvous.

a) Welche Attribute benötigt das Kernobjekt? Wie sind sie initialisiert? Welche ungültigen Fälle müssen ausgeschlossen werden?

b) Beschreiben Sie den Ablauf von `rendezvous`. Argument ist ein Aufrufwert. Zudem gibt es einen Rückgabewert.

Das beschriebene Verhalten entspricht in etwa dem von `rendezvous` in Plan 9:
`http://man.cat-v.org/plan_9/2/rendezvous`

Aufgabe K14: Semaphor mit Zusatzmethoden

Ausgangspunkt ist ein als Kernobjekt implementiertes Semaphor mit den Operationen `P` und `V`, wie im Begleitbuch beschrieben. Die Methoden heißen hier `Psyn` und `Vasyn`.

Attribute:

- **Zähler**, nicht negativ. Initialwert beim Erzeugen anzugeben.
- **Wartemenge** für Prozesse. Initial leer.

Dateninvariante:

- Wenn **Zähler** > 0, dann **Wartemenge** leer.
  (d.h. ungültiger Fall: **Zähler** > 0 und **Wartemenge** nicht leer)

Methoden:

`Psyn`: Runterzählen um 1, synchron, wartet bei Zählerstand 0

- **Zähler** > 0 ?

ja: **Zähler** vermindern um 1
nein: Aufrufer blockieren und in **Wartemenge** stecken

`Vasyn`: Hochzählen um 1, asynchron

- **Wartemenge** leer?

ja: **Zähler** erhöhen um 1
nein: Prozess aus **Wartemenge** nehmen und deblockieren

Die Implementierung soll um die unten aufgeführten Methoden ergänzt werden. Welche davon lassen sich einfach, d.h. ohne Änderungen an den Attributen oder der Dateninvariante, implementieren? Beschreiben Sie für diese jeweils den Ablauf. Begründen Sie bei den anderen, warum keine einfache Implementierung möglich ist.

a) `Ptry`: Versuchendes Runterzählen um 1. Erfolg falls Zähler > 0.

b) `Vtry`: Versuchendes Hochzählen um 1. Erfolg falls ein Prozess wartet.

c) `PNsyn`: Runterzählen um mehr als 1, synchron. Argument N, positive Zahl.

d) `VNasyn`: Hochzählen um mehr als 1, asynchron. Argument N, positive Zahl.

e) `PNtry`: Versuchendes Runterzählen um mehr als 1. Erfolg falls Zähler groß genug.

f) `VNtry`: Versuchendes Hochzählen um mehr als 1. Erfolg falls mindestens so viele Prozesse warten, dass sie um N runterzählen.

g) `P0asyn`: Runterzählen bis 0.

h) `V0asyn`: Hochzählen bis Wartemenge leer.

Aufgabe K15: Semaphor mit Argumenten

Ein erweitertes Semaphor ist als Kernobjekt implementiert. Die Operationen zum Hoch- und Runterzählen erhalten jeweils ein Argument, um wieviel der Zählerstand verändert werden soll. Das Kernobjekt bietet folgende Operationen an:

`decrementSyn`: Runterzählen synchron. Argument ist eine positive Zahl R.

`decrementTry`: Runterzählen versuchend. Argument ist eine positive Zahl R. Rückgabewert ist entweder Erfolg oder Misserfolg.

`incrementAsyn`: Hochzählen asynchron. Argument ist eine positive Zahl H.

Die Implementierung des Kernobjekts verwendet folgende Attribute:

Z: Zählerstand, nicht negativ. Initial wählbar.

WR: Wartemenge für Runterzähler. Initial leer.

Aufrufe der `decrement`–Operationen werden reihenfolgetreu durchgeführt. Ein Aufrufer darf also keinen anderen, noch blockierten Aufrufer überholen.

a) Welche Strategie verwendet die Wartemenge WR? Welche zusätzlichen Funktionen muss WR unterstützen, gegenüber einem Semaphor ohne Argumente R und H?
Hinweis: Die zusätzlichen Funktionen ergeben sich aus den Abläufen.

b) Welches sind die ungültigen Fälle des Kernobjekts?

c) Beschreiben Sie den Ablauf von `decrementSyn` mit Argument R.

d) Beschreiben Sie den Ablauf von `incrementAsyn` mit Argument H.

e) Beschreiben Sie den Ablauf von `decrementTry` mit Argument R.

f) Statt der Reihenfolge soll eine andere Strategie zum Einsatz kommen, die Aufrufer mit niedrigem R bevorzugt. Wo und wie ändern sich dabei die Abläufe?
Hinweis: Nur allgemein erklären, keine neuen Ablaufbeschreibungen!

g) Statt der bisherigen soll eine Strategie zum Einsatz kommen, welche die Prioritäten der aufrufenden Prozesse berücksichtigt. Was ist zu bedenken?
Hinweis: Nur allgemein erklären, keine neuen Ablaufbeschreibungen!

Aufgabe K16: Client/Server–Kanal

Ein spezieller Kanal für synchrone Aufträge an Server bietet folgende Operationen an:

`callSyn`: Ein Client sendet einen Auftrag und wartet auf ein Ergebnis.

`takeSyn`: Ein Server nimmt einen Auftrag entgegen.

`replyAsyn`: Ein Server beantwortet einen Auftrag.

Der Kanal verwendet ausschließlich die Referenzablage. Beim Zustellen eines Auftrags an einen Server generiert er eine Auftragskennung. Diese Kennung ist Rückgabewert von `takeSyn` und Argument von `replyAsyn`. Die Implementierung verwendet folgende Datenstrukturen:

CmA: Wartemenge für Clients mit Auftrag. Initial leer.
Enthält Tupel aus (Client, Nachrichtenreferenz, Antwortpufferreferenz).

SoA: Wartemenge für Server ohne Auftrag. Initial leer.
Enthält Tupel aus (Server, Auftragspufferreferenz).

CoE: Wartemenge bzw. Tabelle für Clients ohne Ergebnis. Initial leer.
Enthält Tupel aus (Kennung, Client, Antwortpufferreferenz).

In CoE speichert der Kanal, welcher Client auf das Ergebnis für welche Kennung wartet. Der Zugriff auf die Elemente erfolgt über die Kennung.

Diese Art von Kanal gibt es zum Beispiel in QNX Neutrino:
`https://www.qnx.com/developers/docs/7.0.0/#com.qnx.doc.neutrino.sys_arch/topic/ipc_Sync_messaging.html`

a) Beschreiben Sie den Ablauf der Client–Operation `callSyn`. Argumente sind die Referenzen auf eine Auftragsnachricht und einen Empfangspuffer für die Antwort.

b) Beschreiben Sie den Ablauf der Server–Operation `takeSyn`. Argument ist die Referenz auf einen Empfangspuffer für den Auftrag, Rückgabewert ist die Auftragskennung.

c) Beschreiben Sie den Ablauf der Server–Operation `replyAsyn`. Argumente sind die Auftragskennung und eine Referenz auf die Antwortnachricht.

Aufgabe K17: Kanal mit Durchschlag

Ein spezieller Kernobjekttyp „Durchschlagskanal“ stellt Nachrichten zweimal zu: erst als Original an einen Empfänger, dann als Durchschlag an einen Auditor. Der Auditor erhält zusätzlich die Kennungen von Sender und Empfänger. Um Manipulationen des Durchschlags zu verhindern, wird jede Nachricht zuerst in einen beschränkten Kanalpuffer kopiert. Ist der Puffer voll, scheitert das Senden. Dieser Kanaltyp bietet folgende Operationen an:

`sendAsyn`: sendet eine Nachricht, ohne zu warten

`receiveSyn`: empfängt eine Nachricht im Original, wartet bis zum Eintreffen

`receiveAuditSyn`: empfängt eine Nachricht als Durchschlag, wartet bis zum Eintreffen

Die Implementierung verwendet folgende Datenstrukturen, initial sind alle leer:

P: Pufferspeicher für Nachrichteninhalte

MO: Menge für Originale, also neue Nachrichten

MD: Menge für Durchschläge, also zugestellte Nachrichten

WE: Wartemenge für Empfänger von Originalen

WA: Wartemenge für Auditoren, also Empfänger von Durchschlägen

a) Welche Informationen speichern die Elemente der vier Mengen?
Hinweis: Die Antwort ergibt sich aus den Ablaufbeschreibungen.

b) Beschreiben Sie den Ablauf des Durchschlagempfangens. (`receiveAuditSyn`)
Argument ist die Referenz auf einen Auditpuffer mit Platz für Nachricht, Sender- und Empfängerkennung.

c) Beschreiben Sie den Ablauf des synchronen Empfangens. (`receiveSyn`)
Argument ist die Referenz auf einen Empfangspuffer mit Platz für eine Nachricht.

d) Beschreiben Sie den Ablauf des asynchronen Sendens. (`sendAsyn`)
Argument ist die Referenz auf eine Nachricht im Adressraum des Senders.

Recherchen und Verschiedenes

Aufgabe K18: Prozessverwaltung (Nachlese)

Die Zuteilung von Rechenzeit durch den Aufgreifer (engl.: *scheduler*) ist eng verzahnt mit der Verwaltung von Prozessen in der Bereitmenge. Der Aufgreifer muss schnell bestimmen, welcher Prozess als nächstes Rechenzeit erhalten soll. Dazu ordnet er die bereitstehenden Prozesse nach Prioritäten. Eine Tabelle in der englischsprachigen Wikipedia nennt die Algorithmen, die in verschiedenen Betriebssystemen zum Einsatz kommen:

`https://en.wikipedia.org/wiki/Scheduling_(computing)#Summary`

Besonders häufig ist die *multilevel feedback queue* (MLFQ) aufgeführt. Lesen Sie nach, was es damit auf sich hat und beantworten Sie die folgenden Fragen.

a) Welche Auszeichnung erhielt der Erfinder der MLFQ?

b) In welcher Größenordnung liegt die Anzahl der *Level*?

c) Woher kommt das *Feedback*? Wie wirkt es sich aus?

d) Vergleichen Sie die MLFQ mit der Datenstruktur aus Aufgabe V2 „Bereitmenge und Aufgreifstrategie“.

e) Eignet sich die MLFQ auch für andere Situationen, in denen Prozesse verwaltet werden? Zum Beispiel bei blockierten Prozessen in einer Wartemenge?

Aufgabe K19: Prozessmodell (Recherche)

Recherchieren Sie das *five-state process model* und vergleichen Sie es mit dem Prozessmodell aus der Vorlesung.

a) Welche Zustände entsprechen direkt einem im Prozessmodell der Vorlesung?

b) Wie passen die restlichen Zustände zum Prozessmodell der Vorlesung?

c) Welches Verhalten von Prozessen bildet das Prozessmodell aus der Vorlesung ab, das *five-state process model* aber nicht?

Aufgabe K20: Links in Dateisystemen (Recherche)

Das POSIX–Programm `ln` kann zwei verschiedene Arten von Links in Unix–Dateisystemen erzeugen. Das Windows–Programm `mklink.exe` kann drei verschiedene Arten von Links (und sogenannte Junctions) in NTFS–Dateisystemen erzeugen. Recherchieren Sie die verschiedenen Arten von Links.

A) `ln -s`

B) `ln`

C) `mklink`

D) `mklink /d`

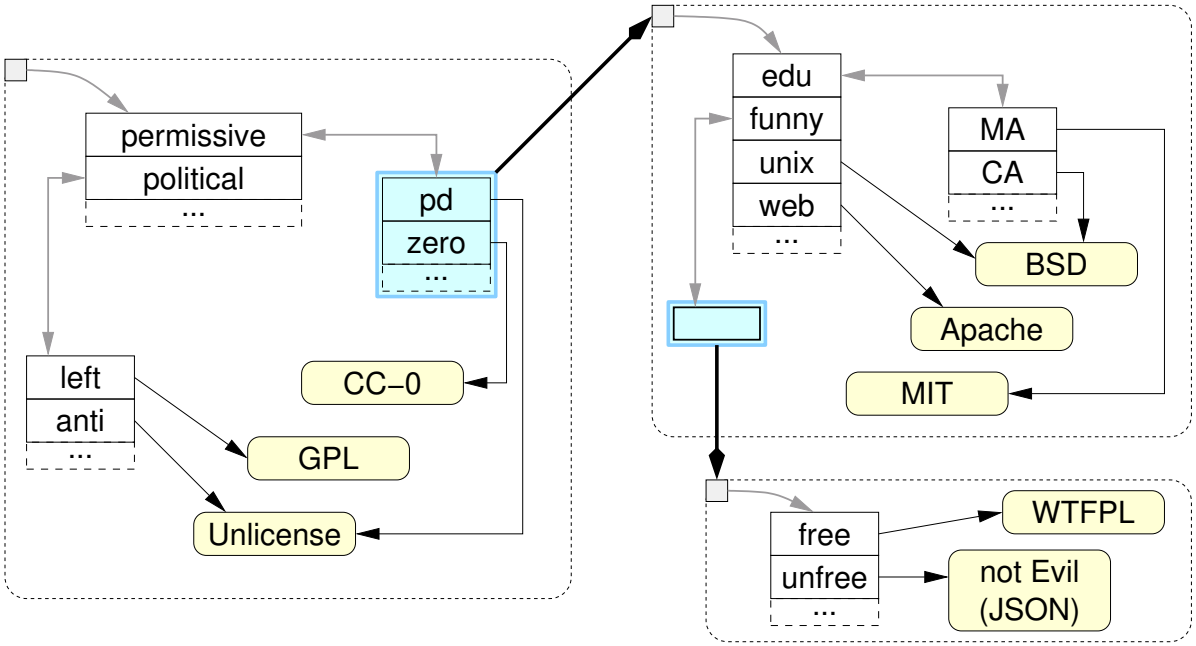
E) `mklink /h`

Welche der folgenden Aussagen gelten für welche Arten von Links?

1. Kann auf eine Datei zeigen.
2. Kann auf ein Verzeichnis zeigen.
3. Kann brechen, d.h. ins Leere zeigen.
4. Kann in ein anderes Dateisystem (Speichermedium) zeigen.
5. Der Link funktioniert weiter, wenn man das Ziel verschiebt.
6. Das Ziel kann sich ändern, wenn man den Link verschiebt.

Aufgabe K21: Dateien und Verzeichnisse

Die folgende Abbildung zeigt drei Dateisysteme mit insgesamt acht Dateien. Das zweite ist beim ersten mit dem Pfad `/permissive` eingehängt, das dritte beim zweiten unter `/permissive/funny`. Die Dateiinhalte sind Namen von Software–Lizenzen, zum Beispiel GPL oder BSD. Geben Sie Pfade ohne Umwege an, also ohne überflüssige `../` oder `./`



a) Welche absoluten Pfade führen zu den acht Dateien?

b) Welcher absolute Pfad führt ins Verzeichnis des dritten Dateisystems? Über welche relativen Pfade erreicht man von dort die Dateien mit Inhalt „Apache“ und „GPL“?

c) Welche relativen Pfade führen aus dem Verzeichnis `political` zu „Unlicense“, „MIT“ und „WTFPL“?

d) Was muss man tun, um auf „CC-0“ zuzugreifen?

e) Zu welchen Dateien kann man Hard Links in `/permissive/edu/` anlegen?

Aufgabe K22: Synchronisation in Java (Recherche)

Die Java–Klasse `Cell` auf der folgenden Seite ähnelt dem Kernobjekt „Zelle“ aus früheren Aufgaben. Eine Zelle ist entweder leer oder enthält einen Integer–Wert. Die Methode `give` legt einen Wert in die Zelle. Enthält die Zelle bereits einen Wert, wird dieser überschrieben. Die Methode `take` entnimmt den aktuellen Wert aus der Zelle. Ist die Zelle gerade leer, fliegt eine Exception.

Die Implementierung von `Cell` ist noch nicht gegen gleichzeitige Aufrufe durch Java–Threads abgesichert. Das sollen Sie im Verlauf dieser Aufgabe ändern.

a) Was könnte schief gehen, wenn zwei Java–Threads die Methoden einer Zelle gleichzeitig aufrufen? Betrachten sie alle drei Kombinationen der beiden Methoden.

b) Recherchieren Sie die Bedeutung von `synchronized` in Java. Was ist der Unterschied zwischen `synchronized` an einer Methode und vor einem Codeblock?

c) Ändern Sie die Methoden in `Cell` so ab, dass mehrere Java–Threads die gleiche Zelle sicher verwenden können.

d) Recherchieren Sie die Methoden `wait()`, `notify()` und `notifyAll()` in Java. Wie hängen sie mit `synchronized` zusammen? Was sind *spurious wakeups*?

e) Schreiben Sie `Cell` so um, dass `take()` auf einen Wert wartet, statt eine Exception zu werfen. Nutzen Sie das Objekt `guard` zur Synchronisation.

f) Welchen Vorteil bringt die Synchronisation an `guard` statt `this`?

```
public class Cell
{
  private boolean holds_value;
  private int     stored_value;

  private final Object guard = new Object();

  public void give(int value)
  {
    stored_value = value;
    holds_value  = true;
  }

  public int take()
  {
    if (!holds_value)
      throw new IllegalStateException("empty");

    holds_value = false;
    return stored_value;
  }
}
```

Aufgabe K23: Übersetzungstabelle

Auf einem fiktiven Rechnersystem sind Adressen 20 Bit lang. Die Seitengröße liegt bei 4 KB, also 12 Bit. Eine fiktive ausführbare Programmdatei für dieses Rechnersystem definiert folgende Segmente:

Zweck	Anfang	Größe
Code	`0x10000`	`0x3000`
Konstanten	`0x13000`	`0x1000`
Statische Daten	`0x14000`	`0x2000`

Der Programmlader des Rechnersystems lädt die Segmente in der genannten Reihenfolge. Anschließend legt er folgende Bereiche an, ebenfalls in der genannten Reihenfolge:

Zweck	Anfang	Größe
Stapel	`0xee000`	`0x2000`
Halde	`0x20000`	`0x4000`

Die Seiten jedes Segments werden vom Anfang zum Ende hin angefordert. Die Speicherverwaltung liefert freie Seiten des Hauptspeichers in folgender Reihenfolge:

`53 54 6c 6a b7 91 55 76 a3 85 84 83 ab ba b3 9f` (*Hexzahlen*)

Erstellen Sie die Übersetzungstabelle für den Programmstart.
Flags: C = Code, D = Daten, L = nur Lesen, S = Schreiben und Lesen

Logisch	**Physisch**	**Flags**

Aufgabe K24: Übersetzungstabelle

Auf einem fiktiven Rechnersystem sind Adressen 20 Bit lang. Die Seitengröße liegt bei 4 KB, also 12 Bit. Eine fiktives Programm für dieses Rechnersystem verwendet eine dynamisch gebundene Bibliothek als Plugin. Die Programm- bzw. Bibliotheksdateien definieren folgende Segmente:

	Programm		Bibliothek	*Lader wählt*
Zweck	Anfang	Größe	Größe	Anfang
Code	`0x01000`	`0x2000`	`0x2000`	`0xe0000`
Konstanten	`0x04000`	`0x1000`	`0x1000`	`0xe3000`
Statische Daten	`0x06000`	`0x1000`	`0x2000`	`0xe5000`

Der Programm- und Bibliothekslader bearbeitet Segmente in der genannten Reihenfolge. Nach dem Laden des Programms legt er folgende Segmente an, wiederum in der genannten Reihenfolge. Nach dem Start veranlasst die Instanz das Laden des Plugins.

Zweck	Anfang	Größe
Halde	`0x10000`	`0x3000`
Stapel	`0xfd000`	`0x2000`

Die Seiten jedes Segments werden vom Anfang zum Ende hin angefordert. Die Speicherverwaltung liefert freie Seiten des Hauptspeichers in folgender Reihenfolge:

`89 b6 35 30 7a 42 c5 c0 dd d3 84 ab 85 50 53 62 a4` (*Hexzahlen*)

Erstellen Sie die Übersetzungstabelle nach dem Laden der Bibliothek.
Flags: C = Code, D = Daten, L = nur Lesen, S = Schreiben und Lesen

Logisch	**Physisch**	**Flags**