9. Übung zur Vorlesung „Vernetzte Systeme“ WS 2001/2002

Prof. Dr. F. Mattern

Ausgabedatum: 17. Dez. 2001
Abgabedatum: 7. Jan. 2002

Hinweis: Bitte schreiben Sie immer Ihre Übungsgruppennummer und die Namen der beiden Bearbeiter auf die Lösung!

Hinweise zu den Programmieraufgaben: Schicken Sie die Quelltexte der Programmieraufgaben bitte als Email-*Attachments* an Ihren Tutor und geben Sie sie ausserdem zusammen mit den anderen Übungsaufgaben als ausgedruckte Listings ab. Bitte tragen Sie Ihren *Namen*, die *Übungsgruppe* und die *Aufgabennummer* als Kommentar im Quelltext ein.

Aufgabe 31 (Token-Release-Strategien)

In einem Token-Ring-Netz mit *delayed token release* wartet die sendende Station, bis die von ihr gesendete Nachricht (d.h. das erste Bit dieser Nachricht) einmal den Ring komplett umrundet hat (und wieder an der Sendestation angekommen ist), bevor sie das Token an die nächste Station weitergibt. Dem gegenüber steht das *immediate token release*, bei dem das Token sofort im Anschluss an das gesendete Paket auf den Ring gelegt wird.

a) (2 Punkte) In FDDI Netzen wird das *immediate token release* Verfahren verwendet. Warum?

b) (2 Punkte) Unter welchen Umständen macht es keinen Unterschied, welches der beiden Verfahren verwendet wird?

Aufgabe 32 (Ethernet)

(3 Punkte) Erläutern Sie, inwiefern es ein Problem ist, wenn bei Ethernet eine Kollision erst dann erkannt werden würde, wenn das Datenpaket bereits abgesendet wurde (wieso also Mindestpaketlängen sinnvoll bzw. notwendig sind).

Aufgabe 33 (CSMA/CD und Binary Exponential Backoff)

Wenn im CSMA/CD-Protokoll zwei Stationen ein freies Übertragungsmedium erkennen und „fast“ gleichzeitig mit dem Senden beginnen, kommt es trotz *carrier sense* zu einer Kollision. Sobald dies von einer der Stationen erkannt wird, sendet diese ein kurzes *Jam*-Signal und beginnt nach einer gewissen Wartezeit einen erneuten Sendeversuch.

In der offiziellen Ethernet-Spezifikation (IEEE 802.3) wird diese Wartezeit durch den *binary exponential backoff algorithm* wie folgt definiert:

> The delay is an integral multiple of slot time. The number of slot times to delay before the *n*-th retransmission attempt is chosen as a uniformly distributed random integer r in the range $0 \leq r < 2^K$, where $K = \min(n, 10)$.

a) (3 Punkte) Was ist der Vorteil des *binary exponential backoff algorithm* gegenüber dem Bestimmen der Wartezeit innerhalb fester Grenzen (also K=const, z.B. K=10)?

b) (4 Punkte) In einem IEEE 802.3 (Ethernet) LAN mit 4 Stationen gebe es folgende Sendewünsche:[1]

- Station A: Slot 1, Slot 3, Slot 12
- Station B: Slot 1, Slot 7, Slot 8
- Station C: Slot 5
- Station D: Slot 5

Sendewünsche werden gepuffert, bis sie erfüllt werden können. Die Stationen verwenden eine Zufallszahlenfunktion, die Zufallszahlen zwischen 0 und 16383 liefert, sowie die Modulo-Division, um den Bereich der Zufallszahlen auf das jeweils gültige Intervall einzuschränken. Um z.B. zu bestimmen, wann eine Station ihren zweiten *retransmission attempt* durchführt, wird mit *modulo 4* gerechnet; beim dritten Mal mit *modulo 8*, usw.

Die Reihen der Zufallszahlen seien wie folgt (nicht alle Zahlen werden tatsächlich benötigt):

- Station A: 394, 5453, 13815, 4410, 2883, 6402
- Station B: 777, 2407, 9599, 3037, 5034, 99
- Station C: 1258, 3547, 733, 688, 9234, 2487
- Station D: 944, 386, 7427, 4434, 2348, 4287

[1] Zur Vereinfachung nehmen wir an, dass Sendewünsche im LAN nur jeweils zu Beginn eines Slots auftreten.

Vervollständigen Sie die folgende Tabelle, indem Sie einen erfolgreichen Sendeversuch mit **S** markieren; einen erfolglosen Sendeversuch (d.h. eine Kollision) mit **X**, eine wartende Station mit **W** und eine inaktive Station mit – markieren:

Zeitschlitz	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Station A																				
Station B																				
Station C																				
Station D																				

Aufgabe 34 (Sliding-Window Empfänger in Java)

Nachdem Sie sich in Aufgabe 30 Gedanken über die Anforderungen einer Ringpuffer-Implementation des Empfangsfensters im Sliding-Window-Protokoll gemacht haben, sollen Sie dies nun in einem Java-Programm umsetzen.

a) (6 Punkte) Laden Sie sich dazu von der Vorlesungs-Web-Seite das Gerüst einer Klasse für einen Sliding-Window Empfänger (`SlidingWindowReceiver.java`) herunter und implementieren Sie die darin vorgegebenen Methoden entsprechend den in Aufgabe 30 gegebenen Anforderungen:

`http://www.inf.ethz.ch/vs/edu/WS0102/VS/index.html#aufgaben`

Wir nehmen an, dass die Sequenznummer eines Paketes in einem dafür vorgesehenen Feld fester Grösse im Paket-Header gespeichert wird. Die Sequenznummern können also nicht beliebig gross sein, sondern sind durch die Anzahl Bits in diesem Feld beschränkt. Ist das Sequenznummernfeld beispielsweise 4 bit gross, können die Sequenznummern in einem Bereich von 0 bis 15 gewählt werden. Der Sequenznummernbereich wird neben der Grösse des Empfangsfensters im Konstruktor des Empfängers angegeben und reicht von 0 bis `seqNrRange`-1.

Folgende Methoden sollen implementiert werden:

- `int nextBufferIndex(int index)`
  Liefert die nächste Index-Position im Puffer-Array nach einer gegebenen Index-Position `index`.
- `int nextSeqNr(int seqNr)`
  Liefert die nächste Sequenznummer nach einer gegebenen Sequenznummer `seqNr`. Beachten Sie bitte, dass nur die Sequenznummern modulo `seqNrRange` gültig sind.
- `Object getNextPacket()`
  Liefert das nächste Datenpaket in der Sequenz zurück. Falls das nächste Paket noch nicht im Puffer gespeichert wurde, soll `null` zurückgeliefert werden.
- `int handlePacket(int seqNr, Object data)`
  Speichert gegebenenfalls den Inhalt `data` des Datenpaketes mit der Sequenznummer `seqNr` im Array und liefert die soeben erhaltene Sequenznummer als *ACK* zurück. Falls kein *ACK* gesendet werden soll, muss `-1` zurückgegeben werden.

Testen Sie dann Ihre Ringpuffer-Implementation mittels eines kleinen Testprogramms (`SlidingWindowReceiverTest.java`), welches ebenfalls von der Web-Seite der Vorlesung heruntergeladen werden kann. Das Testprogramm schickt eine Beispielsequenz von Wörtern an das als Ringpuffer implementierte Empfangsfenster und liest es in unregelmässigen Abständen aus (es simuliert also praktisch die darüber und darunter liegenden Protokollschichten).

Schicken Sie Ihre Version der Empfänger-Klasse bitte als Email-*Attachment* an Ihren Tutor und geben Sie sie ausserdem zusammen mit den anderen Übungsaufgaben als ausgedrucktes Listing ab. Bitte tragen Sie Ihren *Namen*, die *Übungsgruppe* und die *Aufgabennummer* als Kommentar im Quelltext ein.

Bitte beachten Sie, dass keinerlei Veränderungen an dem Testprogramm nötig sein sollten, um Ihre Implementation zu testen!

b) (4 Bonuspunkte) Für all diejenigen, die noch Lust haben und ein paar Extra-Punkte sammeln wollen, hier noch eine optionale[2] Erweiterung des Sliding Window Empfängers:

Implementieren Sie zusätzlich die Methode `show()`, die eine String-Repräsentation des momentanen Zustandes des Ringpuffers zurückliefert. Folgende Informationen sollten aus der (kompakten, einzeiligen) Repräsentation leicht ablesbar sein:

- Momentane Position von *nfe*.
- Belegt-Anzeige für einzelne Pufferplätze (ob leer oder zum Auslesen bereit).
- Momentane Sequenznummern der einzelnen Pufferplätze.

Ein Beispiel für solch eine Repräsentation ist:

```
8   9   *[5]   6   [7]
```

Hier ist die Position des *nfe* mit einem * gekennzeichnet, die Zahlen geben die jeweilige Sequenznummer des Pufferplatzes an, zum Auslesen bereite (d.h. belegte) Speicherplätze sind mit `[]` Klammern umgeben. Im Beispiel wären also an der dritten und fünften Position Pakete im Puffer mit den Sequenznummern 5 bzw. 7. Das nächste ausgelesene Paket wäre 5 (an der dritten Position), während die Pakete mit den Sequenznummern 6, 8 und 9 noch ausstehen.

[2] Die 4 Punkte zählen also nicht in der „Gesamtwertung“ zum Testat – wer diese Aufgabe nicht bearbeitet, verliert also keine Prozentpunkte – sie können aber zur Aufbesserung des Punktekontos verwendet werden.