Institute of Computer Science, University of Bern
Distributed Algorithms, Spring 2020

Prof. Christian Cachin
Jovana Mićić

Solution for exercise 2

2.1 Pizza or pasta? (4pt)

An Italian restaurant has three employees: Alice, Bob and Carol. Alice is a waitress and Bob and Carol are cooks. The restaurant offers two dishes: pizza and pasta. Bob is responsible for making a pizza and Carol for cooking pasta. Cooks can prepare one dish at the time but there is a limit on how many orders they can have in the waiting list. Bob can have 3 pizzas and Carol 7 portions of pasta in the waiting list.

Using terminology as shown in the lecture for the *JobHandler* [CGR11, Sec. 1.4.3], implement the order-processing protocol of the Italian restaurant as shown in Figure 1. It should contain three modules (Alice, Bob, and Carol) and ensure that

- every customer will eventually get his/her order;
- no dish is delivered if it was not ordered;
- every customer gets each order exactly once.

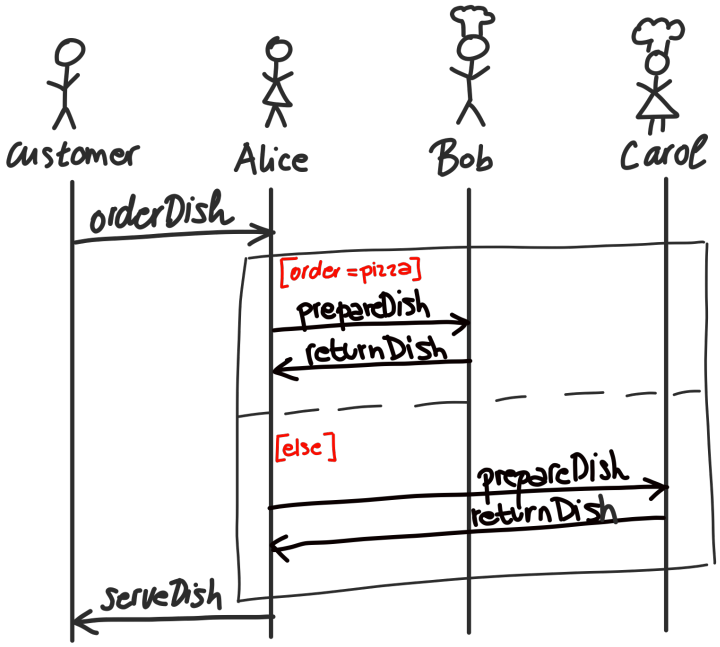

Figure 1. Overview of an order processing in an Italian restaurant.

Algorithm 1 Alice implementation

Implements:
Alice, **instance** a.

upon event $< a,\ Init >$ **do**
buffer $:= \emptyset$

upon event $< a,\ orderDish \mid dish >$ **do** // *dish* is either "pizza" or "pasta"
buffer $:=$ *buffer* $\cup \{dish\}$

upon *buffer* $\neq \emptyset$ **do**
buffer $:=$ *buffer* $\setminus \{dish\}$
if *dish* = "pizza" **then**
trigger $< b,\ prepareDish \mid dish >$
else
trigger $< c,\ prepareDish \mid dish >$

upon event $< a,\ wait \mid dish >$ **do**
buffer $:=$ *buffer* $\cup \{dish\}$

upon event $< a,\ serveDish \mid dish >$ **do**
trigger $<$ *customer*, $serveDish \mid dish >$

Algorithm 2 Bob implementation

Implements:
Bob, **instance** b.

upon event $< b,\ Init >$ **do**
buffer $:= \emptyset$
ctr $:= 0$ // counter *ctr* ensures that ordered dishes are unique

upon event $< b,\ prepareDish \mid dish >$ **do**
if $|$*buffer*$| \geq 3$ **do**
trigger $< a,\ wait \mid dish >$
else
ctr $:=$ *ctr* $+ 1$
buffer $:=$ *buffer* $\cup \{(dish, ctr)\}$

upon exists $(dish, c) \in$ *buffer* **do**
buffer $:=$ *buffer* $\setminus \{(dish, c)\}$
prepare(*dish*)
trigger $< a,\ returnDish \mid dish >$

Algorithm 3 Carol implementation

Implements:
Carol, **instance** c.

upon event $< c,\ Init >$ **do**
buffer $:= \emptyset$
ctr $:= 0$

upon event $< c,\ prepareDish \mid dish >$ **do**
if $|buffer| \geq 7$ **do**
trigger $< a,\ wait \mid dish >$
else
ctr $:= ctr + 1$
buffer $:=$ *buffer* $\cup \{(dish, ctr)\}$

upon exists $(dish, c) \in$ *buffer* **do**
buffer $:=$ *buffer* $\setminus \{(dish, c)\}$
prepare(*dish*)
trigger $< a,\ returnDish \mid dish >$

2.2 Safety and liveness (3pt)

On the opposing sides of a canyon, there are two armies, each led by a different general. General A and General B want to attack a third army (led by General Z), which will travel through the canyon. However, the third army is quite strong, and in order to succeed, both generals need to attack at exactly the same time (both Generals have synchronized clocks available). Thus, both generals devised the following algorithm for a coordinated attack:

- General A chooses an attack time t and sends a messenger m_1 carrying t (denoted $m_1(t)$) to General B.
- General B waits for m_1. Upon the arrival of $m_1(t)$, General B sends a messenger m_2 carrying a confirmation to General A. Then it waits until time t and attacks.
- General A waits for m_2. Upon the arrival of m_2, General A waits until time t and attacks.

Which of the following properties are safety properties, which ones are liveness properties, and which ones are a mixture of the two? Explain your answers (See [CGR11, Sec. 2.1.3]).

a) If some general attacks at time t, then the other general attacks at the same time.

b) If m_2 arrives after time t, then General A will attack after General B.

c) Eventually, General B will attack.

d) If messengers m_1 and m_2 are not intercepted, then eventually both generals attack.

e) If m_1 and m_2 are not intercepted, then eventually both generals attack at time t.

Solution. As a syntactical rule, the word *eventually* in the property **usually** indicates a liveness property. As an intuitive rule, if a system doing nothing satisfies the property, then it is probably a safety property. As a more formal guideline: If one can describe a finite point in time at which

it is possible to check whether the property is satisfied or violated, then it's probably a safety property.

a) This property is a safety property: One could instruct an observer to wait until one general attacks and at that point in time check whether the other general also attacks. If not, then there is no chance to satisfy the property again. This is a characteristic safety property.

b) This property is also a safety property: at any point in time during the execution, one may check if m_2 has arrived and A has promptly attacked, and whether B has attacked before this time.

c) This property is a classical liveness property. There is no finite point in time where this property can be violated.

d) This property is a liveness property because it does not state that both generals must attack at the same time. Any finite behavior can be made good again: If some general attacks then the other general should also attack. If no general has attacked yet, this can be made good again by letting both generals attack.

e) This property is a safety property like properties (a) and (b). Wait for the time when one general attacks and then check if the other general attacks at the same instant. If not, then one has a finite point in time where the property is violated.

2.3 Unreliable clocks (3pt)

Read more in Chapter 8 of DDIA [Kle17], specifically pp. 287–299, which address unreliable clocks.

Describe systems or protocols, where unreliable clocks as presented in DDIA may lead to safety and liveness violations. In particular, for each of the following examples, describe a property and how it is violated:

a) Find two examples, where timing issues lead to safety violations;

b) Find two examples, where timing issues lead to liveness violations.

Solution. Here are two examples of systems and their safety and liveness properties violation.

System 1: Database shared between two computers. Every write in the database includes timestamp of a computer that does it.
Safety:

- Property: Later writes to the database have bigger timestamps than entries written before.
- Violation: Computer A writes at time 0, but the local clock of computer A is broken and writes timestamp 200. Computer B writes at time 100. The violation is that the later write has a smaller timestamp.

Liveness:

- Property: Computer B eventually commits to the database.
- Violation: Use of time-of-day clocks for the timestamps can lead to the bad clock synchronization. Further this can lead to the lost of data that computer B wanted to commit into the database. See an example from the book [Kle17, Figure 8-3].

System 2: System described in the Exercise 2.2 but without the assumption that both generals have synchronized clocks.
Safety:

- Property: If some general attacks at time t, then the other general attacks at the same time.
- Violation: The clock of General A is broken and it is one hour late. Therefore, General A will attack before General B and both armies will fail to win against General Z.

Liveness:

- Property: Eventually, General A will attack.
- Violation: Again let us assume that the clock of general A is late. General A will send the message m_1 with a timestamp to general B. Because of the very old timestamp, General B ignores this message and never sends back message m_2. Therefore General A will never attack.

References

[CGR11] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming (Second Edition)*, Springer, 2011.

[Kle17] M. Kleppmann, *Designing data-intensive applications*, O'Reilly, 2017.