CS586: Distributed Computing
TA: Eleftherios Kosmas
Tutorial 3

Exercise 1

1. Is the Bakery Algorithm correct if all $choosing[j]$ variables ($1 \leq j \leq n$), as well as all code lines referring to them, are omitted in the code? Prove your claim.

2. Present an execution of the Bakery algorithm where the sequence of values that are stored in each $number[j]$ variable, ($1 \leq j \leq n$), is strictly increasing.

Solution:

1. The new bakery algorithm:

   Code for process i
   1: $Number[i] = 1 + max(Number[0], ..., Number[n-1])$;
   2: for $j = 0$ to $n-1$, $j \neq i$, do
   3: wait until $((Number[j] == 0)$ OR $(Number[j], j) > (Number[i], i))$
   4: critical section;
   5: $Number[i] = 0$
   6: remainder section;

   Bellow we present a bad scenario which violates mutual exclusion property. Without loss of generality we assume that only two processes exist in the system:

p_0	p_1
$tmp = max\{0, 0\} + 1$	
	$Number[1] = max\{0, 0\} + 1$
	wait until $((0 == 0)$ OR $(0, 0) > (1, 1))$
	critical section
$Number[0] = tmp$	
wait until $((1 == 0)$ OR $(1, 1) > (1, 0))$	
critical section	

2. Bellow is presented the required execution (again assuming only two processes in the system):

p_0	p_1
$Choosing[0] = TRUE$	
$Number[0] = max\{0, 0\} + 1$	
$Choosing[0] = FALSE$	
wait until $FALSE = FALSE$	
wait until $(0 == 0)$ OR $((0, 1) > (1, 0))$	
	$Choosing[1] = TRUE$
	$Number[1] = max\{1, 0\} + 1$
	$Choosing[1] = FALSE$
	wait until $FALSE = FALSE$
	wait until $(1 == 0)$ OR $((1, 0) > (2, 1))$
critical section	
$Number[0] = 0$	
remainder section	
$Choosing[0] = TRUE$	
$Number[0] = max\{0, 2\} + 1$	
$Choosing[0] = FALSE$	
wait until $FALSE = FALSE$	
wait until $(2 == 0)$ OR $((2, 1) > (3, 0))$	
	wait until $(3 == 0)$ OR $((3, 0) > (2, 1))$
	critical section
	$Number[1] = 0$
	remainder section
	$Choosing[1] = TRUE$
	$Number[1] = max\{3, 0\} + 1$
	$Choosing[1] = FALSE$
	wait until $FALSE = FALSE$
	wait until $(3 == 0)$ OR $((3, 0) > (4, 1))$
wait until $(4 == 0)$ OR $((4, 1) > (3, 0))$	
critical section	
$Number[0] = 0$	
remainder section	
$Choosing[0] = TRUE$	
$Number[0] = max\{0, 4\} + 1$	
$Choosing[0] = FALSE$	
wait until $FALSE = FALSE$	
wait until $(4 == 0)$ OR $((4, 1) > (5, 0))$	
...	...

Exercise 2

1. Prove that the order of lines 12 and 13 in the Black-White Bakery algorithm is crucial for correctness.
2. The author of the Black-White Bakery algorithm claims the following: "The color bit can be easily implemented using n single-writer atomic bits, one for each process. To change the color, process i changes the value of the i-th bit. Black corresponds to the case where the number of bits which are set to 1 is even, while white corresponds to the case where the number of bits which are set to 1 is odd." Provide code for this new version of the algorithm and explain why the new algorithm is correct.
3. Is the Black-White Bakery Algorithm correct if the third clause of the second wait statement is removed?
4. In the Black-White algorithm what would be the size of the entries of the number array if we replace the third line with the following line: $Number[i] = 1 + maximum\{Number[0], ..., Number[n-1]\}$.

Solution:

The original Black-White Bakery algorithm is the following:

Shared variables:
color: a bit of type {black,white}
$choosing[1..n]$: boolean array
$(mycolor, number)[1..n]$: array of type {black,white} × {0,...,n}
Initially, $\forall : 1 \leq i \leq n : choosing[i] =$ false and $number[i] = 0$,
the initial values of all other variables are immaterial.

Code for process i:
$choosing[i] = true$;
$mycolor[i] = color$;
$number[i] = 1 +$ max $\{number[j] | (1 \leq j \leq n) \wedge$ (mycolor[j] = mycolor[i])$\}$;
$choosing[i] = false$;
for $j = 0$ to $n - 1$ do {
    await ($choosing[j] ==$ false);
    if ($mycolor[j] == mycolor[i]$)
    then await ($number[j] == 0$) $\vee$ $(number[j], j) \geq (number[i], i)$ $\vee$
        ($mycolor[j] \neq mycolor[i]$);
    else await ($number[j] == 0$) $\vee$ ($mycolor[i] \neq$ color) $\vee$
        ($mycolor[j] == mycolor[i]$);
}
critical section;
if ($mycolor[i] == black$) then $color = white$;
else $color = black$;
$number[i] = 0$;
remainder section;

1. This solution will be given after the 1st laboratory.
2. The required code is presented bellow:

Shared variables:
color: a bit of type {black,white}
$choosing[1..n]$: boolean array
$b[1..n]$: bit array
$(mycolor, number)[1..n]$: array of type {black,white} × {0,...,n}

Initially, $\forall : 1 \leq i \leq n : choosing[i] =$ false, $b[i] =$false and $number[i] = 0$,
the initial values of all other variables are immaterial.

Local variables:
$count_i$: an integer

Code for process i:
$choosing[i] = true$;
$mycolor[i] =$ **GetColor()**;
$number[i] = 1 +$ max $\{number[j] | (1 \leq j \leq n) \wedge$ (mycolor[j] = mycolor[i])$\}$;
$choosing[i] = false$;
for $j = 0$ to $n - 1$ do {
await ($choosing[j] ==$ false);
if ($mycolor[j] == mycolor[i]$)
then await ($number[j] == 0) \vee (number[j], j) \geq (number[i], i) \vee$
($mycolor[j] \neq mycolor[i]$);
else await ($number[j] == 0) \vee (mycolor[i] \neq$ **GetColor**()) $\vee$
($mycolor[j] == mycolor[i]$);
}
critical section;
if ($mycolor[i] ==$ **GetColor**()) then $b[i] = 1 - b[i]$;
$number[i] = 0$;
remainder section;

{black,white} **GetColor()** :
$count_i = 0$
for each $j = 0$ to $n - 1$ do
$count_i = 1 + b(j)$;
return ($count_i$%2 == 0 ? $black$: $white$)

3. Mutual exclusion property is still ensured. However, deadlock may occur, as presented bellow (assume that *color* is initially black):

p_0	p_1
1-4: $number[0] = 1$ and $mycolor[0] =$ black	
	1-4: $number[1] = 2$ and $mycolor[1] =$ black
5-11: enters critical region	
	5-8: **blocks** on line 8 due to p_0
12-13: $number[0] = 0$ and $color =$ white	
1-4: $number[0] = 1$ and $mycolor[0] =$ white	
5-9: **blocks** on line 9 due to p_1	
deadlock!	

4. In this case, it is possible the $number[i]$ values to be increased illimitably, for the same reason that this can also happen to the Bakery algorithm.