Operating Systems (Erasmus 2024-2025) (25%)

Exercise #1

(i) Show how one can achieve mutual exclusion (protect a 'critical region' / CR) for 'n' concurrent/parallel processes (Pi | i=1…n), with use of a *binary semaphore*.

(ii) Show how two concurrent/parallel processes P1 and P2 can be synchronized in order to be sure that P2 will be executed after the end of the execution of P1, with use of a *binary semaphore.*

(iii) Show how two iterative concurrent/parallel processes P1 and P2 (i.e. two processes that execute iteratively for ever) can be synchronized in order to be executed continuously in the following order

P1, P2, P1, P2, P1, P2,……

with use of two *binary semaphores.*

Exercise #2

Let's have the following arithmetic expression:

A = G + B * (C - D) / ((E + F) * (H - I))

And let's suppose that its calculation is made up of the following seven instructions (E1, E2, …., E7):

```
E1: A11 = C - D;
E2: A12 = E + F;
E3: A13 = H - I;
E4: A21 = B * A11;
E5: A22 = A12 * A13;
E6: A31 = A21 / A22;
E7: A = G + A31;
```

In the above code we use also (except the variables A, B, C, D, E, F, G, H and I that appear in the expression) the variables A11, A12, A13, A21, A22 and A31 for the calculation of intermediate results.

[i] Give the 'precedence graph' that represents the execution order of the seven instructions, (with the maximum possible parallelism with respect to the calculation of the whole expression).

[ii] Give a parallel/concurrent code (that represents the execution order of the seven instructions, with the maximum possible parallelism with respect to the calculation of the whole expression) using **'semaphores'**.

Exercise #3

Let the following two processes D1 and D2 run "in parallel" (ie run on the "cobegin D1; D2; coend" scheme). Assume that each *value assignment* is executed atomically and that each *IF* statement is executed in 2 steps, as follows: step 1: the *IF condition* is checked, and step 2: if the *condition* is true the print command is executed. The two processes use two shared integers (a and b).

shared int a, b; /* they initially have some random value */	
Process D1	**Process D2**

a = 1;
b = 2;
IF (a == b)
  print "A";
	a = 2;
	b = 2;
	IF (a == b)
	  print "B";

[i] What are the 'final' results that may occur after the concurrent execution of both processes? For each such 'final' result, specify the sequence of the execution of the instructions of the two processes that leads to this result.

[ii] Use ***semaphores*** (initializing them appropriately), and P (or wait or down) / V (or signal or up) primitives on these semaphores to ensure that after the parallel execution of processes D1 and D2 the result will always be "AB".

Exercise #4

Three processes (Process1, Process2 and Process3) are executed concurrently (in parallel) (i.e. "cobegin Process1; Process2; Process3; coend"). Each process executes iteratively (10 times) two instructions (E1.1 and E1.2, E2.1 and E2.2, E3.1 and E3.2, respectively) as follows:

Process1	**Process2**	**Process3**
for k = 1 to 10 do	for j = 1 to 10 do	for l = 1 to 10 do
begin	begin	begin
E1.1;	E2.1;	E3.1;
E1.2;	E2.2;	E3.2;
end	end	end

You are asked to complete the above code of the three processes (Process1, Process2 and Process3) with the necessary P (or wait or down) and V (or signal or up) primitives over corresponding ***semaphores*** (which you must initialize appropriately), in order to ensure the following *synchronization conditions* for the order of execution of the instructions of the three processes:

1. In the i^{th} iteration of Process2 (1 <= i <= 10), the execution of E2.1 'follows' the execution of E1.1 in the same (i^{th}) iteration of Process1 (i.e. E2.1 is executed after the end of E1.1 in each iteration).

2. In the i^{th} iteration of Process3 (1 <= i <= 10), the execution of E3.1 'follows' the execution of E2.1 in the same (i^{th}) iteration of Process2.

3. In the i^{th} iteration of Process2 i (1 <= i <= 10), the execution of E2.2 'follows' the execution of E1.2 in the same (i^{th}) iteration of Process1.

4. In the i^{th} iteration of Process 3 (1 <= i <= 10), the execution of E3.2 'follows' the execution of E2.2 in the same (i^{th}) iteration of Process2.

5. In the i^{th} iteration of Process1 (2 <= i <= 10), the execution of E1.1 'follows' the execution of E3.2 in the *previous* ($i\text{-}1^{th}$) iteration of Process3.

Note: In the first iteration (i=1) the E1.1 instruction of Process1 is initially executed (it is the only one that is not subject to any execution constraint) and so on.

(i) First give a solution using five (5) ***semaphores*** (one for each synchronization condition mentioned above).

(ii) Then provide a solution by trying to use the minimum (in your judgment) possible number of ***semaphores*** that are sufficient to achieve the above-mentioned synchronization conditions.

Exercise #5

The following table gives the arrival time, the execution time and the priority value for five processes in a computer system:

Process	Arrival Time (ms)	Execution Time (ms)	Priority Value
A	0	7	2
B	3	11	3
Γ	9	5	1
Δ	10	12	5
E	12	6	4

Give the Gantt diagrams and compute the 'average turnaround time' and the 'average waiting time' for each of the following CPU scheduling algorithms:

- FCFS (First Come First Served).
- SJF (Shortest Job First).
- SRTF (Shortest Remaining Time First).
- Priority Scheduling (non-preemptive).
- RR (Round Robin) with time quantum equal to 4 ms.

Assumptions:

1. The context switch time is negligible.
2. Greater 'priority value' means greater priority in execution.

Exercise #6

The following table gives the arrival time, the execution time and the priority value for five processes in a computer system:

Process	Arrival Time (ms)	Execution Time (ms)	Priority Value
P1	0	12	3
P2	5	19	3
P3	8	21	5
P4	11	13	2
P5	15	5	3

Give the Gantt diagrams and compute the 'average turnaround time' and the 'average waiting time' for each of the following CPU scheduling algorithms:

a) FCFS (First Come First Served)

b) SJF (Shortest Job First)

c) SRTF (Shortest Remaining Time First)

d) Preemptive Priority Scheduling [with use of FCFS in case of equal priorities]

e) RR (Round Robin) with time quantum equal to 8 ms.

Assumptions:

1. The context switch time is negligible.
2. Greater 'priority value' means greater priority in execution.

Exercise #7

A computer architecture with 'paging' memory organization uses 40 bits (in total) for logical addresses, where the first (more significant) 26 bits are used for the 'page number' and the other (less significant) 14 bits are used for the 'offset'. Also, within a physical address of the system, 28 bits are used to specify the page frame number. Finally, in each page table entry 4 additional bits are used (beyond the bits that specify the corresponding page frame for each page) for auxiliary info.

1. Which is the 'page size' of the system ?
2. Which is the maximum number of pages for one process ?
3. Which is the total size of the page table in bytes;
4. The following is part of the 'page table' of a process in the above system.

Page Table	
Page	**Frame**
…	…
$03C_{16}$	$A00_{16}$
…	…
$1E3_{16}$	12345_{16}
…	…
$A11_{16}$	$C31E_{16}$
…	…
$3CDA_{16}$	56789_{16}

Let's suppose the following 'logical (virtual) addresses' are generated during the execution of the process (in hexadecimal):

(a) F1F10

(b) F3691A4

Find the 'physical addresses' that correspond to the above 'logical addresses'. Give your answer in hexadecimal too.

Exercise #8

Let's have the following 'segment table' of a process (all the numbers are in decimal):

Segment	Base	Limit
0	1024	452
1	9896	128
2	512	128
3	3912	1400
4	1536	1024
5	5688	2000

Given the above segment table, find the corresponding 'physical addresses' for the following 'logical addresses' (note that each of the logical addresses is given in the form (x, y), where x is the 'segment number' and y id the ‘offset within the segment' - all the numbers are given in the decimal system).

a. (1, 40)

b. (2, 512)

c. (5, 1536)

Exercise #9

Describe in brief their main features and explain why the following 'combined' memory organization schemes have been extensively used in modern computer systems (which are their advantages, what they offer more etc.) over/instead of the simple 'paging' and 'segmentation' schemes:

- 'paged segmentation'

- 'multi-level paging'

Exercise #10

Let's have a 'paged segmentation' memory organization system, in which each logical address is formed of 36 bits in total. These bits are separated in three fields as follows (from left to right): 's' (segment number), 'p' (page number) και 'd'' (offset). Suppose also that one process can have up to 128 segments, and the page size of the system is 2 Kbytes.

[i] Find which is the size (in bits) of each one of the three fields of the logical address ('s', 'p' and 'd'').

[ii] Let's have a process with 3 segments of 15 Kbytes each in the above system. Find how many pages are needed to host the whole of the process in physical memory, as well as how much is the internal fragmentation caused;