Write your name, surname, student identification number (numero di matricola), e-mail. The answers can be written in English or in Italian. Please, present the work in a legible and readable form. All the answers must be properly and clearly explained, focusing on the requested issues.

1. A system $\Sigma$ is composed of five processes as in the figure below. The source process $P_1$ generates a stream of $L = 10^5$ integers, each one transmitted either to $P_2$ or to $P_3$ with the same probability. The ideal service time of $P_1$ is $T_{id-P1} = 2.5 \times 10^3 \tau$. Processes $P_2$ and $P_3$ apply a computation with calculation time equal to $3.75 \times 10^3 \tau$ and $4.5 \times 10^3 \tau$ respectively, and they transmit their results to $P_4$ with probability 0.75 or to $P_5$ otherwise. Process $P_4$ executes for each integer a function $F$ with average calculation time equal to half its inter-arrival time, while $P_5$ executes a function $G$ with calculation time $1/4$ of its inter-arrival time. Assume that the inter-process communication latency is negligible and answer the following points by providing all the necessary explanations.

   a) Evaluate the performance analysis of the system by showing that the bottleneck is the source process. Determine an approximation of the completion time of the system.

   b) Show that the source’s effective bandwidth is equal to the effective departure rate from the system (explain what is).

   c) Study the merge between processes $P_4$ and $P_5$ into a single process. Show the LC code of the new process resulting from the fusion and describe the type of messages exchanged to respect the computation semantics of the original system. Study if this restructuring of the graph impairs performance in terms of processing bandwidth.

2. A Fat Tree interconnection network is an example of indirect network used in large-scale multiprocessors. Provide your explanations to the following points by assuming wormhole flow control and directory-based cache coherence.

   a) Describe the properties of the network in terms of bandwidth and show the most critical design issues.

   b) Explain what is a Generalized Fat Tree and show graphically how it is mapped on a 2-ary 4-fly network.

   c) Suppose that the network at point b) is used in a SMP CMP-based architecture to connect CMPs to memory macro-modules. Show at least two examples of firmware messages (only request messages) that are exchanged on the network using the Fat Tree routing strategy.

3. Verify whether the following sentences are true or false under specific conditions, and explain the answers in a concise way with the minimum necessary justifications.

   a) During the execution of a critical section protected by a lock, the memory macro-module where the protected data structure is allocated replies only to memory requests coming from the processor that started the critical section (i.e. this is the effect on the indivisibility bit mechanism).

   b) The sequential organization of a modular memory system cannot be adopted in symmetric multiprocessor architectures.

   c) Inter-processor communications can be used to design a fair locking implementation that prevents starvation among processes waiting for the lock acquisition.
Solution Outline

1. The inter-arrival times to the second and the third processes can be evaluated according to the multiple-server theorem:

\[ T_{A-P2} = T_{A-P3} = \frac{T_{id-P1}}{2} = 2 \cdot T_{id-P1} = 5 \times 10^3 \tau \]

Since the communication latency can be neglected in this exercise, the ideal service time of the second and of the third process is equal to their calculation time. Therefore, the utilization factor of the two processes is:

\[ \rho_2 = \frac{T_{id-P2}}{T_{A-P2}} = \frac{3.75 \times 10^3 \tau}{5 \times 10^3 \tau} = 0.75 < 1 \]
\[ \rho_3 = \frac{T_{id-P3}}{T_{A-P3}} = \frac{4.5 \times 10^3 \tau}{5 \times 10^3 \tau} = 0.9 < 1 \]

The two processes are not bottleneck, and their inter-departure time is equal to the inter-arrival time. The inter-arrival time to the fourth process and to the fifth process is (multiple-client theorem):

\[ T_{A-P4} = \frac{1}{2 \cdot \frac{0.75}{5 \times 10^3 \tau}} = 3.33 \times 10^3 \tau \]
\[ T_{A-P5} = \frac{1}{2 \cdot \frac{0.25}{5 \times 10^3 \tau}} = 10^4 \tau \]

The two processes are not bottleneck by definition:

\[ \rho_4 = 0.5 \]
\[ \rho_5 = 0.25 \]

The effective service time of the system is the aggregate inter-departure rate from the two last processes:

\[ T_{D-\Sigma} = \frac{1}{3.33 \times 10^3 \tau + \frac{1}{10^4 \tau}} = 2.5 \times 10^3 \tau \]

As expected, it is equal to the source’s ideal service time, i.e. the source process is the one limiting the overall performance of the system (it is the bottleneck, and its utilization factor is equal to one).

The completion time of the system can be approximated as:

\[ T_c \sim L \cdot T_{D-\Sigma} = 2.5 \times 10^6 \tau \]

The last point of the exercise asks if it is possible to merge the fourth and the fifth process without impairing performance. The merged process (we use the name M) will receive two input streams from the second and the third process non-deterministically (alternative command). To determine the function to be executed, the process M needs further information in the input message: this can be a pair of two integers (op, x), where x is the original integer and op is an integer assuming values 0 (if the function F needs to be used by M) and 1 (for the function G). The value of the op field must be properly filled by the processes P_2 and P_3 whose LC code needs to be slightly modified for this purpose. This modification is quite easy to design, since processes P_2 and P_3 already chose the function to be executed when they decide (in the original system) whether to send to P_4 or to P_5. The LC code of the new process M is the following:

M:: channel in ch_in1(1); channel in ch_in2(1); channel out ch_out; int x;
{
   while true do {
      alternative {
         receive(ch_in1, <op, x>) do skip;
         receive(ch_in2, <op, x>) do skip;
         if op == 0 then
            send(chout, F(x));
      }
else
    send(chout, G(x));
}
}
}

The ideal service time of the new process is:

\[ T_{id-PM} = p \times T_F + (1 - p) \times T_G \]

where \( T_F = 1.66 \times 10^3 \tau \) and \( T_G = 2.5 \times 10^3 \tau \) while \( p \) is the probability that the input element to the merged process requests the execution of function \( F \). This probability is \( p = 0.75 \) therefore:

\[ T_{id-PM} = \frac{3}{4} \times 1.66 \times 10^3 \tau + \frac{1}{4} \times 2.5 \times 10^3 \tau = 1.88 \times 10^3 \tau \]

\[ \rho_M = \frac{1.88 \times 10^3 \tau}{2.5 \times 10^3 \tau} = 0.75 < 1 \]

In conclusion, the fusion between the last two processes does not impair performance and the two graphs have the same steady-state performance in terms of output bandwidth.

2. The properties of a Fat Tree network are described in the book in Section 18.7 and briefly summarized in the 11th slide block at slide 72. The concept of Generalized Fat Tree has been shown in the book at page 18.7.2 and at slide 73. The general construction rule of a generic 2-ary n-fly network is described in Section 18.6.2 of the textbook. The network has the following structure with dashed boxes indicating the nodes of the Fat Tree. Each internal node of the Fat Tree at level \( i=0 \) to \( n-1=3 \) is decomposed in \( 2^i \) switches of the butterfly.

A SMP CMP-based architecture using a 2-ary 4-fly as the main interconnect is characterized by 16 CMPs and 16 memory macro-modules. The network can be used as a butterfly (for main memory accesses) and as a Generalized Fat Tree in the following two examples of messages:

- A processing element \( PE_i \) (in the first CMP) executes MMI/O STORE instructions to generate an inter-processor message to a processing element \( PE_j \) in a different CMP. The message is transmitted (1 flit for the header plus additional flits for the message payload) over the network that is used as a Fat Tree (tree-based routing);
- Suppose that the architecture adopts a directory-based automatic cache coherence protocol. A processing element \( PE_i \) (in the first CMP) executes a LOAD instruction for a cache block \( b \) which is not present in its L1 and L2 caches. The firmware interpreter of the LOAD instruction sends a C2C read request message to the home node \( PE_j \). In case the home node is in a different CMP, the request message (1 flit for the header plus two flits for the block physical address according to our assumptions) is transmitted over the network that is used as a Fat Tree.
3. The exercise proposes three questions:

   a) **False**: the indivisibility bit mechanism is used to correctly implement the read-modify-write code during the lock acquisition (few assembler instructions of the lock primitive). During the critical section protected by the lock (so, once the lock has been acquired), the whole memory (all the macro-modules) can be freely accessed by any processor. In fact, all the processors trying to access the protected data structure should execute a lock primitive at the very beginning of their critical section, and since the lock is busy they will do busy-waiting until the lock is released.

   b) **False**: in the sequential organization, each macro-module stores a set of $C/\sigma$ blocks with contiguous physical addresses, where $C$ is the capacity of the macro-module. The memory system must be studied similarly to a NUMA, since the probability distribution of memory accesses may be no longer the uniform one although the distance between the PEs and the macro-modules is homogeneous.

   c) **True**: this implementation was shown during the course (14th slide block, slide 24, and pages 309-310 of the textbook). In the simplified version, the lock protects a data structure shared by exactly two processes. The identifiers of the corresponding processors (we assume exclusive mapping) is supposed to be known during the lock creation by initializing constant fields of the lock data structure that will contain the processors’ identifiers. The lock is implemented by an integer assuming values 0 (the lock is busy and no processor is waiting), 1 (the lock is busy and the other processor is waiting), 2 (the lock is free). The indivisibility bit is used to make atomic the read-modify-write pattern manipulating the integer. When a processor waits for the lock acquisition, it executes a WAITINT instruction. When the lock is released, in case the other processor is waiting it will be notified by an inter-processor message received through the interrupt mechanism. The implementation can be generalized to the case of a lock shared among $n > 2$ processes where conceptually the lock consists in a boolean flag for the lock state (free or
busy) and a FIFO buffer of processor identifiers waiting for the lock acquisition. However, the buffer and the flag must be accessed atomically during the lock/unlock sequences (through the indivisibility bit), and in modular memories this requires a special care in order to implement the lock correctly.