Informe:

Desarrollamos un modelo para el sistema de un procesador con dos núcleos, utilizando un monitor para garantizar la exclusión mutua y así resolver los problemas de concurrencia. El monitor hace uso de una Red de Petri para administrar así la generación y posterior procesamiento de las actividades.Se cuenta además con un objeto Política que resuelve el conflicto entre las transiciones sensibilizadas que alimentan los buffers de los núcleos, manteniendo la carga equitativa.

Entre los *criterios adoptados*, podemos mencionar que:

* El objeto Política asigna las actividades generadas a cada buffer según una asignación de probabilidades según el caso:

1. Ambos núcleos con el mismo tiempo de “service\_rate”: 50% para cada núcleo.
2. Un núcleo con el doble de tiempo de “service\_rate” que el otro: 66% para el núcleo con menor “service\_rate”.
3. Un núcleo con el triple de tiempo de “service\_rate” que el otro: 75% para el núcleo con menor “service\_rate”.
4. Para un caso general, el tiempo de “service\_rate” del núcleo menor “service\_rate” será:

* El objeto Política posee un vector en el cual se ordenan las transiciones de la red según el orden de prioridad elegido, permitiendo así decidir en caso de conflicto, que transición es preferible disparar primero. El orden de prioridad elegido es:

1. T5
2. T8
3. T4
4. T10
5. Arrival\_rate
6. Service\_rate\_1
7. Service\_rate\_2
8. T0
9. T12
10. Power\_Up\_Delay\_1
11. Power\_Up\_Delay\_2
12. T2
13. T11
14. Power\_Down\_Threshold\_1
15. Power\_Down\_Threshold\_2

* El orden elegido se debe a la decisión de darle prioridad al inicio de realizar las actividades, seguidos por las transiciones que las depositan en los buffers antes que terminar las actividades, debido a que de esta forma, puedo evitar que el núcleo se suspenda para luego iniciarse casi instantáneamente.
* Para asegurar exclusión mutua en el monitor, se implementaron semáforos en el acceso al monitor.
* Si un hilo intenta disparar una transición temporizada, primero se corrobora que este sensibilizada por tiempo, en caso de no estarlo, el hilo se duerme el tiempo que le haga falta. Antes de dormirse, libera el mutex. Transcurrido el tiempo, el hilo retorna del monitorpara luego volver a intentar dispararla transición.
* Al disparar una transición en el monitor antes de permitirle el acceso a otro hilo, primero nos fijamos si hay algún otro hilo bloqueado dentro de la cola del monitor que pueda disparar una transición. En ese caso se elige qué hilo despertar en base a las prioridades que se encuentran en Política.
* Se considera terminada la ejecución del main al finalizar el procesamiento de las 1000 actividades. Los hilos de los núcleos que sigan en ejecución, luego de procesar las 1000 actividades generadas, quedarán en las colas del monitor tras no poder salir del modo de “Stand\_by” al no generarse nuevas actividades.

*Resultados Obtenidos*:

* Debido a que los Núcleos son independientes entre sí, es posible trabajen en simultáneo si ambos tienen actividades disponibles en sus respectivos buffers.
* Al trabajar en simultáneo, el tiempo de procesamiento de actividades puede reducirse.
* Dado el paralelismo del desarrollo de las actividades por los Núcleos, el criterio de generación y procesamiento de cada núcleo y lo explicado en el punto anterior, podemos considerar que el tiempo de ejecución del programa tiene una cota inferior en la generación de actividades (). El otro tiempo a considerar es el de procesamiento. Para calcular este tiempo se debe recurrir a la siguiente fórmula: siendo el mínimo común múltiplo de los tiempos, y el tiempo de procesamiento del Núcleo 1 y el tiempo de procesamiento del Núcleo 2. Esto se va a dar considerando las probabilidades definidas anteriormente. Luego, el tiempo de ejecución del programa será aproximadamente , siendo esto el mayor de ambos tiempos. Esto puede variar, ya que como la distribución se genera a partir de una función rand, la probabilidad no garantiza que exactamente se distribuyan en la medida adoptada, si no una aproximación.
* Tiempo de ejecución obtenidos para 1000 repeticiones:
  + Prueba 1:

alfa=5; beta=10; gamma=10; corridas=10

tiempo promedio de ejecución=5,494 seg

* + Prueba 2:

alfa=10; beta=10; gamma=10; corridas=10

tiempo promedio de ejecución=10,463 seg

* + Prueba 3:

alfa=5; beta=10; gamma=20; corridas=10

tiempo promedio de ejecución=6,520 seg

* + Prueba 4:

alfa=10; beta=10; gamma=20; corridas=10

tiempo promedio de ejecución=10,579 seg

* + Prueba 5:

alfa=5; beta=10; gamma=30; corridas=10

tiempo promedio de ejecución=7,794 seg

* + Prueba 6:

alfa=10; beta=10; gamma=30; corridas=10

tiempo promedio de ejecución=10,653 seg