Portada Interna de Tesis

Universidad Autónoma de Querétaro
Facultad de Informática
Maestría en Ciencias Computacionales

IMPLEMENTACIÓN DE UN CONTROLADOR, SENSOR Y ACTUADOR EN UNA RED ETHERNET CON RTLINUX

TESIS
Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias Computacionales

Presenta:
Pedro Chávez Barrios

Dirigido por:
Dr. Héctor Benítez Pérez

SINODALES

Dr. Héctor Benítez Pérez	_____
Presidente	Firma
Dr. Marco Antonio Aceves Fernández	_____
Secretario	Firma
Dr. Jesús Carlos Pedraza Ortega	_____
Vocal	Firma
Dr. Efrén Gorrostieta Hurtado	_____
Suplente	Firma
Dr. Juan Manuel Ramos Arreguin	_____
Suplente	Firma

_____	_____
M. C. Ruth Angélica Rico Hernández	Dr. Luis Gerardo Hernández Sandoval
Director de la Facultad	Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
25 de marzo del 2010

México

RESUMEN

Este trabajo de investigación trata sobre la implementación de un sistema distribuido en tiempo real, en el primer capítulo que es la introducción se detalla el sistema distribuido en tiempo real del que son realizadas las pruebas, se definen los objetivos que se pretenden alcanzar y la relevancia de esta investigación. En el capítulo segundo son tratados los antecedentes refiriéndose a sistemas operativos de tiempo real, sistemas distribuidos, creación y formación del sistema operativo en tiempo real elegido y algunos ejemplos y características de estos sistemas tales como planificador FIFO y Round Robin, modelo cliente servidor y protocolos. En el tercer capítulo llamado desarrollo del sistema se define el concepto de tiempo real, se explican las características de las tareas en el diseño del software de un sistema de control también hacemos hincapié en la relación entre el sistema operativo Linux y el sistema operativo en tiempo real RTLinux, se da la definición de módulos, tipos de módulos y la estructura que guardan. Se explica la arquitectura que tiene RTLinux incluyendo una gráfica que expone los componentes utilizados en cada uno de los nodos utilizados. La tarjeta de red Ethernet es tratada acerca de las funciones que manejan las interrupciones, tipos de tarjetas que podemos utilizar, aplicación que nos ayudara a utilizar la tarjeta Ethernet, forma de configurar dicha tarjeta y conclusiones del capítulo.

Para el caso de estudio en el capítulo cuatro es mostrada la conexión del hardware del sistema distribuido en tiempo real, modelo utilizado, se detalla la forma en que obtuvimos el tiempo promedio y tiempo máximo de transporte de los datos a través de la red y los problemas que se presentaron a causa del cuello de botella ocasionado por los paquetes así como la forma en que se resolvió por medio de la relación periodicidad de la tarea con el tiempo máximo de envío del paquete. Las graficas de tiempos utilizados por los paquetes son mostradas también la frecuencia de los tiempos y desviación estándar de dichos tiempos. Es presentada la forma en que los procesos utilizan el procesador y como se implementa en un sistema distribuido en tiempo real; se muestran los tiempos utilizados en cada uno de los nodos dentro de un sistema distribuido en tiempo real además de las variables que se pretenden medir en el sistema y los retardos obtenidos por las tareas principales finalizando con una conclusión del capítulo.

En el capítulo 5 de pruebas y análisis de resultados es explicada cada una de las pruebas realizadas, forma y algoritmos utilizados, figuras que muestran la forma de envío y tablas que muestran los datos obtenidos en el modelo cliente servidor; por último el análisis de los resultados. Para el capitulo 6 se dan las conclusiones generales de la investigación.

SUMMARY

This research deals about the implementation of a distributed system in real time, in the first chapter is the introduction that spells out the real-time distributed system, where the tests are done, defines the objectives to be achieved and the relevance of this research too. In the second chapter the background are treated in reference to real-time operating systems, distributed systems, creation and training of real-time operating system chosen and some examples and characteristics of these systems such as FIFO and Round Robin scheduler, client-server model and protocols. In the third chapter called development of the system defines the concept of real time, explains the features of the tasks in the design of the software in a control system, we also emphasize the relationship between the operating system Linux and the operating system in real time RTLinux, given the definition of modules, module types and structure saved also in the third chapter explains the architecture that has RTLinux including a graph that shows the components used in each of the nodes used as well as the Ethernet card is treated on the functions that handle interruptions, types of cards you can use and an application that will help us to use and configure the Ethernet card and conclusions of the chapter.

For the case study in chapter four is shown the hardware connection in a distributed system of real-time, model used, it describes the way in which we obtained the average and maximum time of transporting data over the network and problems that arose because of the bottleneck caused by the packages and how they were resolved by means of the relation frequency of the task with the maximum time of dispatch of the package. The graphics of times used by the packages are shown, the frequency of the times and standard deviation of these times. Here is presented the way in which processes use the processor and as implemented in a distributed system in real time, showing the times used in each of the nodes within a distributed real-time addition to the variables being measured in the system and the delays obtained by the main tasks ending with a conclusion of the chapter.

In Chapter 5 of testing and analysis of results is explained each of the tests, forms and algorithms used, figures that show how to send and tables showing data from client-server model, and finally the analysis of results. For Chapter 6 gives the general conclusions of the investigation.

AGRADECIMIENTOS

A mi hijo Pedro que me motivó a finalizar mi tesis.

INDICE

INDICE DE TABLAS

INDICE DE TABLAS

INDICE DE FIGURAS

Capítulo 1

1.- INTRODUCCIÓN

En este trabajo se implementa un sistema distribuido con 3 nodos trabajando con un sistema operativo en tiempo real llamado Rtlinux versión 3.2 el cual es obtenido a través de FSMLabs. Para que el sistema se encuentre en red es necesario algunas aplicaciones para la tarjeta Ethernet (RTL-lwIP), para el uso de memoria (bigphysarea-3.2) además de la instalación más completa de C, en este trabajo se eligió la versión gcc-2.95.3. Los nodos utilizados son llamados Sensor, Control y Actuador.

Se crea una aplicación en el espacio del usuario que genera un flujo de paquetes de 80 caracteres que serán manipulados por el nodo Sensor y enviados al nodo Control, en el nodo Control se reciben los paquetes se muestran en la pantalla y se envían al nodo Actuador, el nodo Actuador recibe los paquetes y envía una señal al puerto paralelo. Los eventos descritos enmarcan la definición de tiempo real “Responder a estímulos externos en un tiempo acotado”. En cada uno de los nodos se hace uso de módulos que son archivos de código que pueden ser insertados en tiempo de ejecución.

El nodo Sensor, al cual se le han insertado los módulos requeridos para el uso de los recursos de la computadora como hardware y memoria se le inserta un módulo que contiene hilos para el manejo de las tareas que tiene este nodo, siendo la más importante la obtención de los paquetes; El flujo de paquetes de 80 caracteres es obtenido por medio de una aplicación localizada en el espacio del usuario dentro de la arquitectura de RTLinux utilizada, este flujo es manejado con estructuras y un archivo secuencial fifo; En el momento de obtener cada paquete es enviado al nodo Control. El nodo Control entre sus tareas la más importante es la de recibir los paquetes desde Sensor realizar una acción con estos paquetes posteriormente enviarlos al nodo Actuador. En el nodo Actuador la tarea primordial es la de recibir los paquetes desde Control y enviar una señal al puerto paralelo. Todo esta implementación es para dar una respuesta que es la señal al puerto paralelo a un evento que aparece en Sensor que es la generación de paquetes por medio de la aplicación en el espacio del usuario en un tiempo acotado; Así este trabajo intenta implementar tiempo real (dar respuesta a un evento en un tiempo acotado) dentro de cada nodo como en la red.

Para llevar a cabo esta implementación es necesario el manejo de hilos que son procesos que consumen los recursos (tareas) por lo que será necesario contar con una planificación de tareas para la adecuada repartición de los recursos. Además el uso de técnicas de exclusión mutua serán necesarias para el exclusivo uso de los recursos por la tarea en ejecución, como habrá más tareas, la competencia por los recursos esta reglamentada, así cada tarea tendrá definidas sus características de tiempo de consumo de procesador, prioridad y periodicidad necesarias para los planificadores utilizados FIFO y Round Robin.

Implementado el sistema se muestran estadísticas de pérdidas, análisis de tiempos, latencia, tiempo de respuesta con variaciones en el tamaño de los datos en un límite máximo de tiempo, gráfica tiempos máximo, mínimo de envío de paquetes y promedio de traslado de paquetes desde un modelo cliente servidor.

El estudio de Sistemas Distribuidos en Tiempo Real fue iniciado en los años 80's y es ampliamente utilizado en nuestros días. Organizaciones dedicadas al estudio de tiempo real, latencia, planificación y control tales como Fsmlabs y QNX han realizado trabajos que están enfocados a aplicaciones de simulación, redes, servicios financieros, logística, sistemas de aerolíneas, seguridad, telecomunicaciones y educación.

Este estudio involucra conceptos de latencia, planificación y redes en un sistema distribuido en tiempo real, con nodos que representan un Controlador, Actuador y Sensor. Posteriormente se realizan estadísticas de tiempos utilizados por los paquetes, pérdidas, análisis de tiempos (máximos) tomando en cuenta las variaciones de las respuestas del Sistema Distribuido. Además este trabajo se enfoca en evitar la pérdida de datos en un Sistema Distribuido en Tiempo Real así como garantizar la entrega de n datos logrando un mejor desempeño.

El contar con el software tanto como el hardware para realizar estas estadísticas es una tarea complicada que involucra conocimientos del sistema operativo Linux, procesos, hilos, imagen, kernel así como de incontables ensayos de prueba y error en la preparación de cada uno de los nodos que involucran esta tesis. Por otra parte existe una problemática dentro de un Sistema Distribuido en Tiempo Real la cual involucra el manejo de módulos que llevarán a la solución de evitar la pérdida de datos. En uno de estos módulos esta la función de levantar a tarjeta de red que es primeramente configurada a través de RTL-lwIP-0.4.

Finalmente contando con hardware, software, módulos y habiendo realizado las pruebas necesarias se logra eliminar la pérdida de datos por medio de threads controlando el envío y la recepción de cada uno de los paquetes que viajan a través de la red.

Los Sistemas Distribuidos en Tiempo Real están presentes en nuestra vida diaria, prácticamente en todo lo que nos rodea; en aviones, trenes, automóviles, teléfonos celulares, etc., son un elemento imprescindible para garantizar la generación, transmisión y distribución de los datos sin pérdida de datos para asegurar la calidad y la seguridad de incontables procesos industriales. Este estudio se realiza por la necesidad de conocer el comportamiento de los datos en la comunicación entre procesadores que garantizan el desempeño de rendimiento y tiempos. Este estudio beneficiará a las personas, estudiantes e investigadores que requieran implementar o utilizar un Sistema distribuido en Tiempo Real y de esta manera cuenten con la información acerca del desempeño de la red y sus tiempos de respuesta.

En el aspecto teórico, los conocimientos que brinda este trabajo de investigación son de sincronización, planificación de tareas, procesos, tareas, particionamiento, administración del tiempo y finalmente sistemas embebidos. Profundiza en el concepto de Tiempo como la principal característica que distingue a los Sistemas de Tiempo Real de otros tipos de sistemas. La palabra tiempo significa que el correcto funcionamiento de un sistema depende no sólo del resultado lógico que devuelve la computadora, también depende del tiempo en que se produce ese resultado.

El aporte de esta investigación es comprobar que en la implementación de un Sistema Distribuido en Tiempo Real se garantiza el desempeño en la ejecución de un proceso en un límite máximo de tiempo sin perdida de datos.

En la figura 1.1 se tiene un ejemplo del uso de un Sistema Distribuido con Sistema Operativo en Tiempo Real donde se pueden arruinar piezas o productos si están pasando por un proceso de calentamiento, presión o cualquier otro proceso donde el tiempo sea un factor determinante. Existen 3 etapas en las cuales el producto ingresará. En cada etapa el procesador (Controladora de Unidad) tiene varias tareas que realizar y cada tarea cuenta con una Periodicidad P, Prioridad PR y uso de procesador C las cuales serán planificadas con FIFO, Round Robin u otro planificador, posteriormente se pasa el producto a la siguiente etapa que tendrá sus propias tareas con Periodicidad, Prioridad y consumo de procesador así como su planificador, en cada etapa hasta el final tosas las tareas serán planificadas para el adecuado uso del procesador de acuerdo al planificador utilizado.

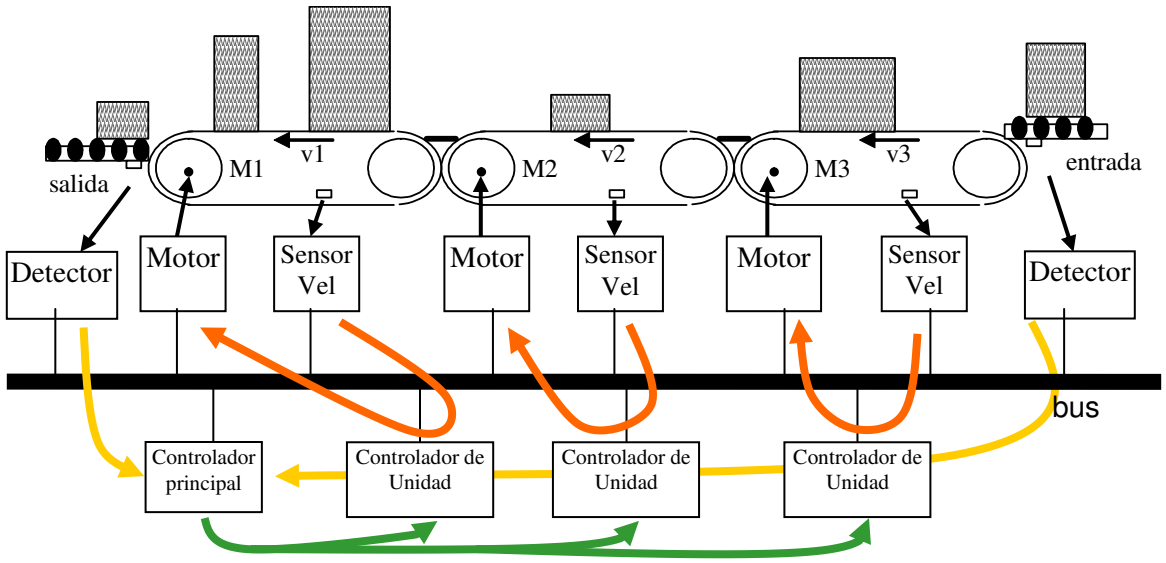


Figura 1.1 Banda transportadora en un Sistema distribuido en tiempo real.

Un Sistema Distribuido en Tiempo Real es utilizado para responder a los estímulos externos en un tiempo establecido donde la predicibilidad del tiempo de respuesta determine que el sistema es capaz de ofrecer una respuesta correcta ante la llegada de un estímulo en un tiempo acotado. Algunas de las razones y motivos por las cuales se realiza este trabajo de investigación son:
En primer lugar que lideres globales como Cisco, Delphi, General Electric, Siemens han investigado que un sistema Operativo en Tiempo Real (como FSMLabs, QNX software systems, Red Hat Embedded Linux, Ubuntu Studio, VxWorks) les da la plataforma a aplicaciones de gran performance en las telecomunicaciones, en el ramo automotriz, instrumentación medica, automatización y seguridad además la gran cantidad de Arquitecturas distribuidas existentes que también pueden migrar a campos de aplicación en Tiempo Real [1], en muchos casos con restricciones de tiempos críticos y requerimientos de seguridad como:

- ✓ Procesos de Control y automatización en fábricas (en manufactura y robots)
- ✓ Sistemas de transporte (aviones, trenes, autobuses, carros)
- ✓ Sistemas multimedia (IP videophones).

Por otra parte, es el conocimiento adquirido de los conceptos acerca del tema, tal es el caso de Sistemas Distribuidos de Tiempo Real los cuales son sistemas informáticos en los que la respuesta de la aplicación ante estímulos externos debe de realizarse en un tiempo establecido. La predicibilidad del tiempo de respuesta determina que el sistema será capaz de ofrecer una respuesta correcta ante la llegada de un estímulo en un tiempo acotado [3]. Además del manejo de algunos sistemas operativos Unix ya sea SUSE, Mandrake, Red Hat, Ubuntu y de sistemas operativos en tiempo real (QNX RTOS y Momentics, FSMLabs rtlinux) así como el conocer la compatibilidad entre sistemas operativos y sistemas operativos en Tiempo Real como del concepto de sistemas embebidos, el cual debido a su enorme crecimiento en los últimos años ha repercutido en un renovado interés por los sistemas en tiempo real.

El estudio de los Sistemas en Tiempo Real como de los Sistemas Embebidos en donde los avances en el hardware hacen posible que funciones de comunicación y procesos en componentes de bajo costo embebidos hacen de este tema de suma importancia. Sus aplicaciones están en procesos de Control en automatización, automatización de oficinas. Algunas aplicaciones tienen restricciones de tiempo real teniendo fuerte impacto en la vida humana, ejemplo sistemas de transporte, suministro de energía, químicos, etc. [2]. De esta forma estaremos entrando a un tema de tecnología avanzada lo cual es atractivo.

Finalmente, la facilidad de comprender su uso en la vida cotidiana, ejemplo: Los sistemas de tiempo real deben ser capaces de actuar ante situaciones que son detectadas por Sensores. Un caso se encuentra en los sistemas de Control de vuelo si un aeroplano experimenta un cambio brusco en la altitud de manera inmediata se realiza una acción que maneje la emergencia. En un automóvil el sistema de frenos debe de Controlarse de la mejor manera posible porque una falla que no es atendida en el menor tiempo posible puede producir la muerte de personas.

Objetivos

Objetivo General:
Implementar un Sistema Distribuido en Tiempo Real con nodos que representen un Controlador, Actuador y Sensor, realizar estadísticas de pérdidas, análisis de tiempos (máximos) tomando en cuenta las variaciones de las respuestas del Sistema Distribuido.

Objetivos Específicos:
Comparar los tiempos de respuesta para n datos que son transmitidos desde Sensor al Controlador y del Controlador al Actuador, analizando el tiempo de respuesta de los datos y sus retardos en el incremento del tamaño y número de datos.
Garantizar la ejecución de un proceso en un límite máximo de tiempo en un Sistema Distribuido en Tiempo Real.
Determinar que el sistema es capaz de responder a los estímulos externos en un tiempo acotado

Relevancia

El aporte de esta investigación es comprobar que en la implementación de un Sistema Distribuido en Tiempo Real se garantiza el desempeño en la ejecución de un proceso en un límite máximo de tiempo sin pérdida de datos.

Capítulo 2

2.-ANTECEDENTES

Se elige RTLinux por ser un sistema operativo en tiempo real con planificadores de tareas críticas para terminar dentro de los plazos determinados (deadline) con mínima latencia, maneja tiempos en nanosegundos, contiene planificadores FIFO y Round Robin, es un software fácil de conseguir y se puede implementar en un sistema distribuido con ayuda de otras aplicaciones. RTLinux nació del trabajo de Michael Baranov y Victor Yodaiken en New México Tech. Hoy en día continúan activamente con su desarrollo desde su propia empresa (FSM Labs) desde la que ofrecen soporte técnico. RTLinux se distribuye bajo la "GNU Public License" y recientemente Victor Yodaiken ha patentado la arquitectura original en que se basa RTLinux, actualmente funciona sobre arquitecturas PowerPC, i386.

A partir del código de Yodaiken, se esta desarrollando otro proyecto liderado por Mantegazza llamado "Real time Aplicación Interface" RTAI, inicialmente las versiones de RTLinux ofrecían un API muy reducido sin tener en cuenta ninguno de los estándares de tiempo real como POSIX real time, Threads, etc. A partir de la versión 2.0 Yodaiken decide reconvertir el API original a otro que fuera compatible con el API de POSIX Threads. Los sistemas distribuidos en tiempo real son utilizados en simulación utilizando planificadores de hilos con precisión en nanosegundos, desplegado de graficas en tiempo real, etc. [10]. En octubre de 2007 se anuncia "Enterprise real time Net" el cual minimiza la latencia de la red, ancho de banda de la red y ejecución de tareas de la red con el peor caso de 50 microsegundos. Real-TimeNet controla las E/S de la red y procesos del CPU con su hard real-time networking stack con soporte para ICMP, IP, UDP and TCP que mejora el ruteo, seguridad y tolerancia a fallas por medio de un software de red en tiempo real creando una "Real-Time Processes Layer" (capa de procesos en tiempo real) la cual elimina el cuello de botella de la red e incrementa el envío de datos a través de la red.

Otro sistema operativo en tiempo Real es QNX el cual fue fundado en 1980 por Dan Dodge y Gordon Bell ambos graduados de la Universidad de Waterloo en Ontario, Canadá. Inicialmente el sistema operativo fue llamado QUNIX sin embargo cambio su nombre a QNX el cual corría en un procesador 8088. QNX 2 salió en los 80's y esta corriendo aún en muchos sistemas de misión-critica hasta ahora. Por el año de 1995, el nuevo miembro de la familia QNX fue introducido su nombre es QNX Neutrino [5].

En el año 2006, la plataforma en tiempo real QNX (el cual consiste en el sistema operativo QNX Neutrino, el sistema de ventanas Photon, herramientas de desarrollo, compiladores, etc.) fue liberado sin costo para propósitos no comerciales.

El sistema operativo en Tiempo Real QNX Neutrino RTOS: Desde 1980, empresas de manufactura han confiado sus aplicaciones críticas a esta tecnología —desde instrumentos médicos, ruteadores de Internet, Call Centers, Aplicaciones de Control de Procesos, Sistemas de Control de Tráfico Aéreo. Pequeño o grande, simple o distribuido, esos sistemas comparten la reputación de operar las 24 horas del día los 365 días del año sin parar. QNX Neutrino RTOS establece un estándar para confiabilidad, tolerancia a fallas y escalabilidad [6].

En los últimos 25 años, QNX software (los Sistemas Operativos en Tiempo Real) han llegado a ser parte de la vida diaria. La gente encuentra estos sistemas e Control en el auto, en la tienda, viendo TV, en el uso de Internet, etc. Los sistemas distribuidos en tiempo real se han aplicado a diferentes

soluciones como en los servicios financieros donde se tiene garantía en la ejecución de las tareas y baja latencia de los datos, en el mercado de valores debe de entregar una confiable información de precios, así podrá maximizar los beneficios en el mercado axial como minimizar el tiempo de recepción de una nueva información para recalcular los riesgos, transacciones en cajeros automáticos, autorizaciones de tarjeta de crédito, Web sites, etc.

Las Redes en Tiempo Real han tenido avances tecnológicos de HW desde los años 70's, dichas redes han sido establecidas en automatización de oficinas, automatización de procesos de fábrica, Control de procesos, sistemas embebidos. Algunas empresas cuentan con casi 15 de años de experiencia en sistemas de tiempo real enfocados en adaptar la tecnología para aplicaciones de alto performance y tiempo de respuesta crítico. Sus aplicaciones son relacionadas a soluciones de sincronización del tiempo, sistemas de tiempo real y diseño de servicios para Linux u otro sistema base, seguridad y tolerancia a fallas, ciclo de vida del software, simulación y automatización.

Los Sistemas Distribuidos son "sistemas cuyos componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor". Características de un Sistema Distribuido:

- Concurrencia.- Esta característica de los sistemas distribuidos permite que los recursos disponibles en la red puedan ser utilizados simultáneamente por los usuarios y/o agentes que interactúan en la red.
- Carencia de reloj global.- Las coordinaciones para la transferencia de mensajes entre los diferentes componentes para la realización de una tarea, no tienen una temporización general, está más bien distribuida a los componentes.
- Fallos independientes de los componentes.- Cada componente del sistema puede fallar independientemente, con lo cual los demás pueden continuar ejecutando sus acciones. Esto permite el logro de las tareas con mayor efectividad, pues el sistema en su conjunto continua trabajando.

La Computación Cliente Servidor es un modelo, que predomina en la actualidad, permite descentralizar el procesamiento y recursos, sobre todo, de cada uno de los servicios y de la visualización de la Interfaz Gráfica de Usuario. Esto hace que ciertos servidores estén dedicados solo a una aplicación determinada y por lo tanto ejecutarla en forma eficiente.

La definición de Cliente-Servidor: es un sistema donde el cliente es una máquina que solicita un determinado servicio y se denomina servidor a la máquina que lo proporciona. Los servicios pueden ser la ejecución de un determinado programa, acceso a un determinado banco de información, acceso a un dispositivo de hardware.

Es un elemento primordial, la presencia de un medio físico de comunicación entre las máquinas, y dependerá de la naturaleza de este medio la viabilidad del sistema. Un protocolo es un conjunto bien conocido de reglas y formatos que se utilizan para la comunicación entre procesos que realizan una determinada tarea. Se requieren dos partes la especificación de la secuencia de mensajes que se han de intercambiar y la especificación del formato de los datos en los mensajes. El protocolo permite que componentes heterogéneos de sistemas distribuidos puedan desarrollarse independientemente, y por medio de módulos de software que componen el protocolo, lograrán que

exista una comunicación transparente entre ambos componentes. Es conveniente mencionar que estos componentes del protocolo deben estar tanto en el receptor como en el emisor.
Ejemplos de protocolos usados en los sistemas distribuidos:

- IP: Protocolo de Internet.- Protocolo de la capa de Red, que permite definir la unidad básica de transferencia de datos y se encarga del direccionamiento de la información, para que llegue a su destino en la red.
- TCP: Protocolo de Control de Transmisión.- Protocolo de la capa de Transporte, que permite dividir y ordenar la información a transportar en paquetes de menor tamaño para su transporte y recepción.
- HTTP: Protocolo de Transferencia de Hipertexto.- Protocolo de la capa de aplicación, que permite el servicio de transferencia de páginas de hipertexto entre el cliente WEB y los servidores.
- SMTP: Protocolo de Transferencia de Correo Simple.- Protocolo de la capa de aplicación, que permite el envío de correo electrónico por la red.
- POP3: Protocolo de Oficina de Correo.- Protocolo de la capa de aplicación, que permite la gestión de correos en Internet, es decir, le permite a una estación de trabajo recuperar los correos que están almacenados en el servidor.

Los protocolos en algunos sistemas distribuidos en tiempo real están soportados por un proyecto llamado LWIP. LWIP es un una pequeña e independiente implementación del protocolo TCP/IP que ha sido desarrollado por Adam Dunkels en “the Computer and Networks Architectures (CNA) lab at The Swedish Institute of Computer Science (SICS)” [7]. LWIP es adecuado para usar en sistemas distribuidos de tiempo real embebidos con decenas de kilobytes de memoria RAM libre. Las características de lwIP son el manejo de protocolos como IP, ICMP, UDP, TCP, DHCP.

En los años 80’s inician los trabajos en el campo de los sistemas distribuidos en tiempo real, el cual se define como programas que interactúan con el mundo exterior de una manera que implica al tiempo. Cuando aparece un estímulo, el sistema responde a éste de cierta manera y antes de cierto tiempo límite. El momento en que se produce la respuesta es tan importante como aquello que se produce.

En un sistema distribuido de tiempo real un dispositivo externo genera un estímulo para la computadora, la que entonces debe realizar varias acciones antes de un momento límite. Al terminar el trabajo solicitado el sistema queda inactivo hasta que viene el siguiente estímulo. Con frecuencia los estímulos son periódicos, de modo que un estímulo ocurre de manera regular cada ΔT segundos, como una computadora en un televisor o una video casetera que recibe un nuevo cuadro cada 1/60 segundos. Algunos estímulos son esporádicos (inesperados) como el sobre calentamiento de un dispositivo. Aún en un sistema que en gran medida sea periódico, una complicación es que pueden existir muchos tipos de eventos, como entrada de video, entrada de audio y el control de la unidad motora, cada uno con su periodo y acciones necesarias. En un sistema distribuido en tiempo real el CPU tiene que trabajar con varios flujos de eventos, no es aceptable que el CPU "diga": "es cierto que omití el evento B, pero no es mi error; yo seguía trabajando en A cuando sucedió B". Los sistemas distribuidos en tiempo real pueden estructurarse como se muestra en la figura 2.1. Aquí vemos una colección de computadoras conectadas mediante una red. Algunas de estas están conectadas a dispositivos externos que producen o aceptan datos o esperar ser controlados en tiempo real. Las computadoras pueden ser pequeños microcontroladores integrados a los dispositivos, o máquinas independientes. En ambos casos, por lo general tienen sensores para recibir señales de los dispositivos y/o actores a los cuales enviar señales. Los sensores o actores pueden ser digitales o analógicos.

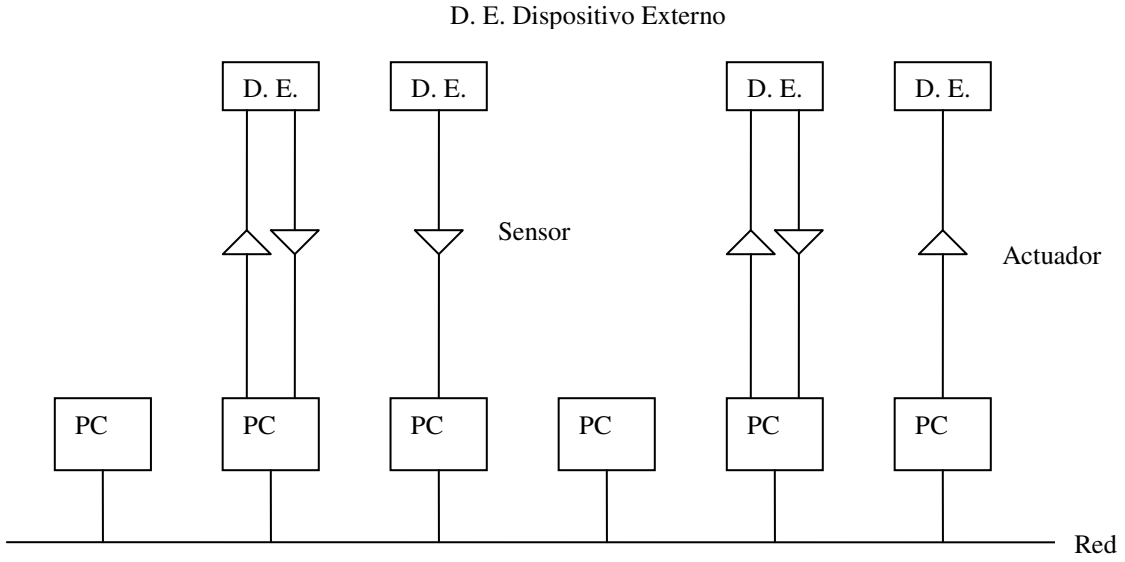


Figura 2.1 Sistema distribuido en tiempo real.

En un sistema distribuido de tiempo real aparecen nuevos problemas a resolver como el de comunicación, sincronización y planificación. El problema de la sincronización es fundamental entre varios procesadores, cada uno con su propio reloj local [12]. El punto es que la rapidez ayuda a sistemas distribuidos de tiempo real a alcanzar la respuesta requerida pero en general no soporta la predictibilidad que es uno de los objetivos de los sistemas distribuidos de tiempo real. Un sistema

distribuido de tiempo real en caso de contención de recursos, las tareas mas importantes deben ser atendidas primero que las tareas de menor importancia (fairness no es importante) [13].

Un planificador es la huella de identidad de un sistema operativo, es tan importante en los sistemas distribuidos manejan más de uno, buena parte del esfuerzo por optimizar el rendimiento de un SO se basa en la política del Scheduling según Fernando Shapachnik, Departamento de Computación, FCEyN, Universidad de Buenos Aires, Buenos Aires, Argentina, Objetivos de un planificador:

- Ecuanimidad (fairness): que cada proceso reciba una dosis justa" de CPU.
- Eficiencia: tratar de que la CPU esté ocupada todo el tiempo.
- Carga del sistema: minimizar la cantidad de procesos listos que están esperando CPU.
- Tiempo de respuesta: minimizar el tiempo de respuesta percibido por los usuarios interactivos.
- Latencia: minimizar el tiempo requerido para que un proceso empiece a dar resultados.
- Tiempo de ejecución: minimizar el tiempo total que le toma a un proceso ejecutar completamente.
- Rendimiento (throughput): maximizar el número de procesos terminados por unidad de tiempo.
- Liberación de recursos: hacer que terminen cuanto antes los procesos que tiene reservados más recursos.

Cuando actúa el planificador con preemptive:

- El scheduling puede ser cooperativo o con desalojo, si es con desalojo (también llamado scheduling apropiativo o preemptive), el scheduler se vale de la interrupción del reloj para decidir si el proceso actual debe seguir ejecutándose o le toca a otro.
- Recordemos: el reloj interrumpe 50 o 60 veces/seg.
- Si bien suele ser deseable, el scheduling con desalojo: Requiere un reloj con interrupciones (podría no estar disponible en procesadores embebidos).
- No le da garantías de continuidad a los procesos (podría ser un problema en SO de tiempo real).

Algunos de los sistemas distribuidos en tiempo real utilizan los planificadores Round Robin o FIFO

FIFO

- Un enfoque posible es FIFO, también conocido como FCFS (First Came, First Served).
- El problema es que supone que todos los procesos son iguales. Si llega un mega proceso" que requiere mucho tiempo de CPU, tapona a todos los demás. Entonces, se agregan prioridades al modelo. Como en una sala de espera.
- Posible problema: inanición (starvation). Los procesos de mayor prioridad demoran infinitamente a los de menor prioridad, que nunca se ejecutan. Una posible solución: aumentar la prioridad de los procesos a medida que van envejeciendo".
- Cualquier esquema de prioridades fijas corre riesgo de inanición.

Round Robin

- La idea es darle un quantum a cada proceso, e ir alternando entre ellos.
- Cuánto dura el quantum, si es muy largo, en SO interactivos podría parecer que el sistema no responde. Si es muy corto, el sistema pasa un porcentaje alto de su tiempo haciendo mantenimiento" en lugar de trabajo de verdad.
- Se lo suele combinar con prioridades, que pueden estar dadas por el tipo de usuario (administrativas) o pueden ser “decididas" por el propio proceso. Esto último no suele funcionar.
- Además, los procesos que hacen E/S suelen recibir crédito extra, por ser buenos compañeros

Los sistemas distribuidos en tiempo real ejecutan software que provee soporte a negocios con decisiones críticas para una organización, este software de tiempo real es desplegado en PCS estándar o comerciales, notebooks, servers, mainframes, etc. que son mantenidos por departamentos de Tecnología de Información. Para los sistemas distribuidos en tiempo real se pueden utilizar diferentes sistemas operativos como QNX, LynxOS, RedHat Embedded Linux, Ubuntu Studio (Linux), VxWorks , Windows CE y UNIX (Some). Las características de un sistema operativo en tiempo real es que usado típicamente para aplicaciones integradas, normalmente no utiliza mucha memoria, cualquier evento en el soporte físico puede hacer que se ejecute una tarea, muchos tienen tiempos de respuesta predecibles para eventos electrónicos.

Algunos sistemas operativos de tiempo real tienen un programador y diseños de Controladores que minimizan los periodos en los que las interrupciones están deshabilitadas. Muchos incluyen también formas especiales de gestión de memoria que limitan la posibilidad de fragmentación de la memoria y aseguran un límite superior mínimo para los tiempos de asignación y retirada de la memoria. Un ejemplo temprano de sistema operativo en tiempo real a gran escala fue el denominado «programa de Control» desarrollado por American Airlines e IBM.

LynxOS RTOS es un sistema operativo de tiempo real tipo Unix de LynuxWorks (anteriormente "Lynx Real-Time Systems"), la primera versión de LynxOS se creó en 1986 en Dallas Texas para un procesador Motorola 68010. En 1988-89, una versión de LynxOS que se hizo para una arquitectura Intel 80386, en 1989, la compatibilidad con SVR3 se ha añadido al sistema, y más tarde, la compatibilidad de Linux. Hoy en día, LynxOS también funciona en otras arquitecturas [11]. LynxOS es un sistema operativo en tiempo real determinístico esto significa que deben de responder dentro de un conocido periodo de tiempo. Esta respuesta predecible es asegurada aún en la presencia de un pesado hardware de I/O por lo que debe ser extremadamente rápido además ejecuta complejas series de tareas.

VxWorks es un sistema operativo de tiempo real, basado en Unix, vendido y fabricado por Wind River Systems, como la mayoría de los sistemas operativos en tiempo real, vxWorks incluye kernel multitarea con planificador preemptive (los procesos pueden tomar la CPU arbitrariamente), respuesta rápida a las interrupciones, comunicación entre procesos, sincronización y sistema de archivos. Las características distintivas de VxWorks son la compatibilidad POSIX, el tratamiento de memoria, las características de multi-procesador, una shell de interfaz de usuario, monitor de rendimiento y depuración de código fuente y simbólico. VxWorks se usa generalmente en sistemas embebidos. Al contrario que en sistemas nativos como Unix, el desarrollo de vxWorks se realiza en un "host" que ejecuta Unix o Windows.

En la actualidad, vxWorks puede ejecutarse en prácticamente todas las CPU modernas del mercado de sistemas embebidos. Esto incluye la familia de CPUs x86, MIPS, PowerPC, SH-4, ARM, StrongARM y xScale [9].

Windows CE es el sistema operativo de Microsoft incrustado modularmente de tiempo real para dispositivos móviles de 32-bits inteligentes y conectados. Windows CE combina la compatibilidad y los ping a servicios de aplicación avanzados de Windows con soporte para múltiples arquitecturas de CPU y opciones incluidas de comunicación y redes para proporcionar una fundación abierta para crear una variedad de productos. Windows CE impulsa a los dispositivos electrónicos del cliente, terminales Web, dispositivos de acceso a Internet, Controladores industriales especializados, computadoras de bolsillo, dispositivos de comunicación incrustados e incluso consolas de video juegos como fue en el caso de la Sega Dreamcast (1997 - 2001) con procesador SH4 de 128 Bits que ya con un sistema operativo propio, incluía compatibilidad con los kits para desarrollo de software de Windows CE.

Windows CE no es un subconjunto de Windows XP, o de Windows NT, sino que fue desarrollado a base de nuevas arquitecturas y una nueva plataforma de desarrollo. Aún así mantiene cierta conexión con sus hermanos. Windows CE tiene sus propias APIs para desarrollo, y necesita sus propios drivers para el hardware con el cual va a interactuar. Windows CE no es un sinónimo de Windows XP en forma pequeña, incrustada o modular.

Capítulo 3

3.- DESARROLLO DEL SISTEMA

En este capítulo se describen conceptos relevantes en el desarrollo de un sistema distribuido en tiempo real como el tiempo de respuesta, tarea, periodicidad, tiempo de ejecución también se explica la relación entre Linux y RTLinux. Se describe el significado de módulo, el uso y su forma de operar además se describe la arquitectura, el hardware y software de apoyo utilizado así como la descripción de la tarjeta de red utilizada para que se realice la implementación en un sistema distribuido.

3.1.- Tiempo Real

Cuando se habla de tiempo real se hace referencia a una acción que ocurre inmediatamente o dentro de un tiempo bien establecido. Actualmente el termino es utilizado para describir diferentes características de los sistemas de cómputo, por ejemplo existen sistemas operativos de tiempo real que permiten usar funciones y estructuras que garantizan tiempos de respuesta [8]. La característica principal de un sistema en tiempo real es la respuesta que se tiene dentro de un límite de tiempo acotado el cual puede medirse en mili, micro o nanosegundos.

En el diseño del software del sistema de Control, cada Controlador se estructura en una tarea. Una tarea (T) esta caracterizada por los siguientes parámetros funcionales: periodo (P) representa que cada tarea es ejecutada de forma regular cada intervalo de tiempo, plazo de entrega (D) de una tarea es el intervalo de tiempo máximo que puede transcurrir entre el instante en el cual debe de activarse y su finalización, desfase inicial de una tarea denota el desfase que tiene el inicio de la primera activación de la tarea con respecto al tiempo de inicio y se denota como ø, tiempo de computo (C) de una tarea es el tiempo de CPU necesario para completar su ejecución en cada una de las activaciones depende del algoritmo de Control y la velocidad del procesador, retardo de inicio una actividad de una tarea tiene un retardo de inicio que corresponde con el tiempo en el cual la actividad empieza a ser ejecutada, tiempo de finalización es el tiempo en el cual una actividad de una tarea, finaliza su ejecución.

En general un sistema esta formado por un conjunto de tareas: $\tau = \{ T1, T2, \ldots, Tn\}$. Cada tarea $Ti=(Ci, Di, Pi, øi)$ donde Ci es el tiempo de peor caso de ejecución de la tarea, Di es el plazo de entrega, Pi es el periodo, øi es el desfase inicial e i es el número de tarea.

3.2.- RTLinux y su relación con Linux

RTLinux es una extensión al kernel de Linux para Control del tiempo real. Rtlinux es un sistema operativo para diseñar tareas en tiempo real. No confundir la versión de Rtlinux con el núcleo de Linux. Parte de la distribución de RTlinux es un parche sobre el código Linux. Y otra parte son los módulos recargables

RTlinux es un sistema operativo en tiempo real estricto con planificador expulsivo por prioridades fijas, señales, sistemas de archivos POSIX (open, close, etc.) semáforos y variables de condición, acceso directo al hardware (puertos e interrupciones, eficiente gestión de tiempos, el peor caso 1 microsegundo, facilidades para incorporar nuevos componentes: relojes, dispositivos de E/S y planificadores.

Otras características relevantes: POSIX Threads, actualmente Linux Thread ha pasado a formar parte de la biblioteca estándar del lenguage C de GNU, conocida como la glibc2. Esta implementación de Pthreads es utilizada para crear procesos. La biblioteca de Pthreads no modifica el núcleo del SO., solamente implementa el API de hilos sobre las llamadas al sistema existente.

Interrupciones, cuando un proceso normal esta ejecutándose el SO (núcleo) esta detenido, pues en un sistema procesador solo existe un proceso ejecutándose. La única forma de tomar el Control del núcleo es mediante las interrupciones. Cada vez que se produce una interrupción el procesador deja lo que esta haciendo y pasa a ejecutar la rutina de interrupción. Estas rutinas (función) pertenecen al núcleo.

Módulos del núcleo

Para poner en ejecución una rt-task se tiene que utilizar el sistema de módulos cargables de Linux. Los módulos son trozos de sistema operativo que se pueden insertar y extraer en tiempo de ejecución. Un módulo es un fichero objeto obtenido a partir de un fuente en C.

Módulos del núcleo cargables

Desde las funciones de inicialización y finalización (init y cleanup) se pueden hacer uso de todos los recursos del núcleo en forma segura. (Estas funciones no son en tiempo real), el núcleo de Linux se puede considerar como otra tarea de tiempo real pero que es planificada con la mínima prioridad.

Un módulo es un fichero que se puede enlazar y desenlazar en el núcleo de Linux en tiempo de ejecución. Con los módulos ya no es necesario crear un nuevo núcleo y rearrancar la maquina cada que hacemos una prueba. Un modulo es un programa en lenguaje C sin función main() y ha de tener las funciones init_module y cleanup_module. Los módulos que el núcleo puede cargar suelen residir en /lib/modules/[úname –r] [4].

Los módulos son archivos de código que no contienen dentro de su estructura a la palabra reservada main( ) y pueden ser cargados en tiempo de ejecución. Los componentes de un módulo en tiempo real, los módulos que se utilizan para cada uno de los nodos ya sea Cliente o Servidor tienen la siguiente estructura:

- Inicialización.- llamada a un hilo que se crea y esta listo para ejecución.
- Run-time.- inicio de la ejecución del hilo que se ejecuta en intervalos de tiempo dados en nanosegundos.
- Terminación: Para salir del run time se invoca la función cleanup_module(), la cual cancela el hilo.

Los nodos tienen cargados los módulos de rtlinux que corresponden al hardware necesario para utilizar en este sistema distribuido, enseguida están los módulos de rtlinux (fifo, mem, rtl, etc.), posteriormente esta el módulo de la tarjeta, de memoria, el stack y finalmente cada uno de los nodos utilizará un módulo que se encarga de su tarea principal .

La arquitectura Rtlinux

Rtlinux no modifica el Kernel de Linux, sino que construye un microkernel o capa de software llamado Rtlinux sobre el hardware. Linux es ejecutado como la tarea de más baja prioridad de Rtlinux. Rtlinux no añade nuevas llamadas al sistema ni las modifica, tampoco es una biblioteca para el programador. Se sitúa entre el hardware y el SO creando una máquina virtual para que siga funcionando. Las tareas RT-linux se ejecutan utilizando el Run Time Support de Linux (RTS).

Las tareas de tiempo real:

Comparten el mismo espacio de memoria que el núcleo.
No pueden hacer uso de las llamadas al sistema de linux
Se ejecutan en modo supervisor

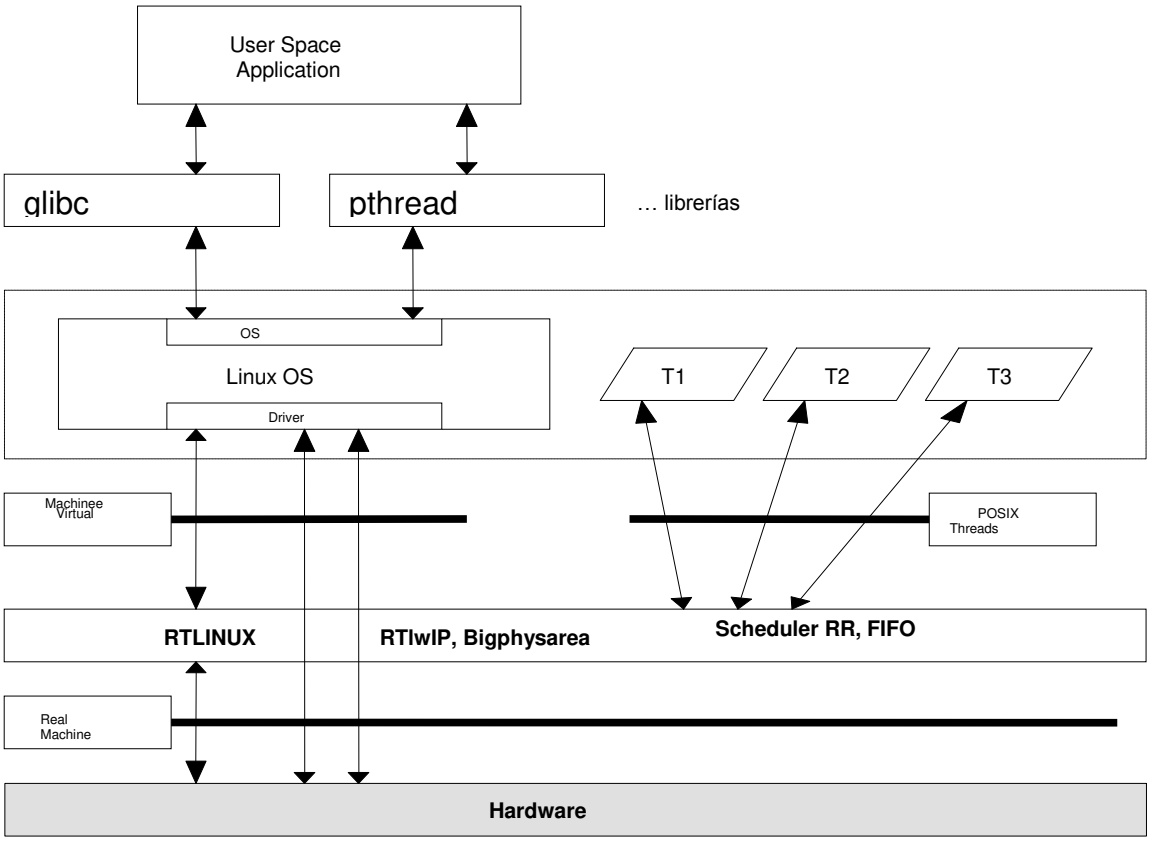


Figura 3.1 Arquitectura de Rtlinux

En la figura 3.1 la arquitectura de RTLinux es la siguiente: sobre el hardware se establece la máquina virtual con RTLinux el cual maneja procesos en tiempo real con planificadores, a RTLinux tenemos que incluirle algunas aplicaciones, una es un software de manejo de memoria (bigpgysarea), para el uso de la tarjeta Ethernet (RTL-lwIP); Estableciendo memoria, tarjeta de red y planificadores a utilizar se puede accesar al uso de algunas librerías de C para el manejo de hilos del kernel proporcionadas por Rtlinux. Los programas creados se pueden ejecutar o en el espacio del usuario o dentro de la máquina virtual creada por Rtlinux.

La configuración de bigphysarea uno de los elementos más importantes para el manejo de la memoria este código te permite reservar una larga porción de memoria física contigua en el boot time el cual puede ser alojado o desalojado por los drivers del kernel, el kernel de linux-2.4.20 debe

ser parchado con bigphysarea.diff así como agregar en lilo la cantidad de espacio físico que se ocupa (append="bigphysarea=1024") posteriormente reconstruir el kernel para cerciorarse de que la reservación de área física se hizo correctamente utiliza el comando $cat /proc/bigphysarea el cual te indica el área física que se reserva en este momento. En la configuración del nuevo kernel se debe de seleccionar la opción "Support for big physical area reservation" en el menú "Processor type and features" cuando se compile el kernel de Linux.

Los equipos utilizados en el sistema distribuido en tiempo real se especifican en la tabla inferior, se utiliza el sistema operativo rtlinux-3.2-pre2 el cual contiene los módulos (-) mbuff, (-) rtl_fifo, (-) rtl, (-) rtl_posixio, (-) rtl_sched, (-) rtl_time stack , se utilizan 3 Computadoras IBM Personal Computer 300GL con las características mostradas en la tabla 3.1.

Hardware	**PC 300GL** **Sensor**
Procesador	Intel Pentium III
Cd-rom	Creative cd533E-Cf
USB	Controlador de raíz USB Controlador de Host Universal Intel82861A4
Disco Duro	8 GB
Disco 3.5	estándar
Mouse	Puerto de mouse compatible con ps/2
Teclado	Estándar 101/102
Tarjeta de sonido	Cmedia PCI 8338/8738 (solo para Sensor)
Puerto paralelo	Lpt1
Tarjeta de red	3com 905cx PCI

Tabla 3.1.- Hardware necesario en cada PC para el sistema distribuido en tiempo real.

Construyendo un RTLinux POSIX device driver: el aspecto que es necesario conocer de RTLinux es que no tiene soporte para red, es aquí donde se utiliza la aplicación RTL-lwIP. En Linux se tiene un sistema de red que permite registrar drivers de dispositivos de red en una forma particular accediendo a I/O a través de archivos virtuales localizados en un sistema virtual de archivos con un solo directorio: /dev. El driver de la tarjeta Ethernet sería de uno de esos archivos localizados en /dev/eth0, archivo que podría ser accesado por medio de las llamadas open, close, write, read, ioctl.

Con el propósito de registrar POSIX device drivers en RTLinux se describe el manejo de interrupciones en la figura 3.2, el driver tiene que declarar una estructura llamada rtl_file_operations que será usada para acceder a las funciones implementadas por el driver open, close, write, read, ioctl. Para registrar el nuevo dispositivo se tiene que declarar una estructura llamada rtl_file_operations que mantendrá punteros a las funciones del driver, finalmente registrar el driver por la llamada de la función rtl_register_rtldev(), los argumentos son el mayor número (major number) asignado al driver, el nombre del dispositivo (/dev/eth0) y un puntero a la estructura rtl_file_operations.

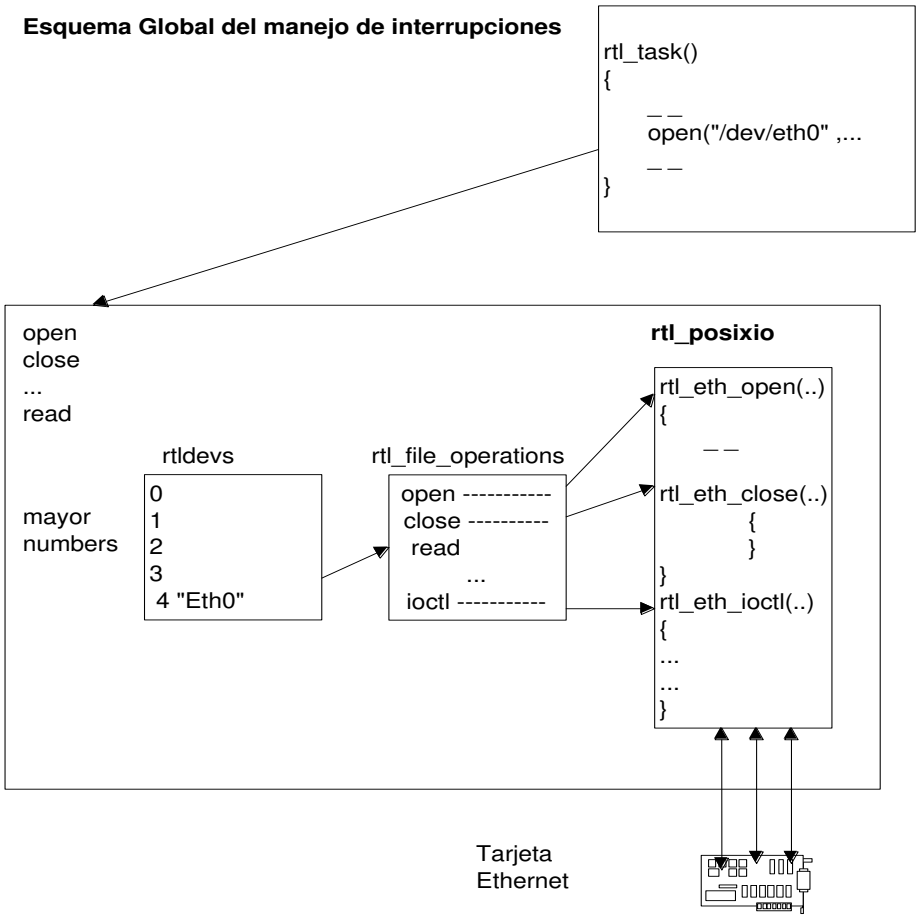


Figura 3.2 Esquema del manejo de interrupciones en la tarjeta Ethernet.

Una vez que el device driver ha sido registrado, una RT-Task que quiere usar el device driver tiene que abrirlo por medio de la llamada open(). Se envían y reciben paquetes por medio de las llamadas write() and read(). La función ioctl() dejaría al usuario configurar algunos parámetros del driver y preguntar al driver por algunas características tales como la MAC de la tarjeta Ethernet.

En RTLinux un bloqueo de la función read puede ser implementada con semáforos o mutex. La llamada read() solo tiene una función, retorna un paquete al llamador; una llamada normal read(), necesitará el usuario pasar un puntero al área reservada de memoria y el driver llenara esa área con datos, eso significa que el driver tendrá que ejecutar una copia por cada paquete. En la

implementación de RTLinux en la llamada read( ), se ahorrará esta copia solo utilizando un puntero al paquete que será mantenido en buffers internos del driver (esta técnica es llamada copia cero).

Las formas para que un modulo kernel hable con los procesos es usando device files (/dev) o usando proc file system, una de las mayores razones de escribir en el kernel es soportar algún dispositivo de hardware en este caso la tarjeta de red.

3.3.- Uso de la Tarjeta Ethernet

En la aplicación utilizada para el uso de la tarjeta Ethernet RTL-lwIP. RTL-lwIP es el porting del lwIP TCP/IP stack a RTLinux-GPL. RTL-lwIP dá a las RT-tareas la oportunidad de comunicación vía TCP/IP con otras tareas o aún con procesos de usuario en Linux.

El enfoque del RTL-lwIP stack es reducir la memoria usada y el tamaño del código, haciendo RTL-lwIP adecuado para uso en pequeños clientes con limitados recursos tales como sistemas embebidos. Con el propósito de reducir procesamiento y demanda de memoria.

RTL-lwIP incluye los protocolos IP, ICMP, UDP, and TCP. Esto provee a las RT-tareas un socket como API para comunicar sobre la red. En este paquete se encuentra el TCP/IP stack, ejemplos de programas, soporte para REDD (RTLinux Ethernet Devices Driver 3com905c-x y realtek8139) la tarjeta que se utiliza es la 3com905c-x.

RTL-lwIP es el puerto entre el stack de TCP/IP a RT-Linux sin embargo no es un protocolo de comunicación, el stack reduce la memoria usada y el tamaño del código; además da a las tareas RT la oportunidad de comunicación vía TCP/IP con otras tareas o procesos de usuarios. RTL-lwIP no es un protocolo de comunicación en tiempo real. RTL-lwIP solo da comunicación vía TCP/IP.

RTL-lwIP consiste en varios archivos los cuales son compilados y ligados dentro de un módulo: rtl_lwip_stack.o., se puede querer cambiar o seleccionar algunos parámetros tales como la tarjeta de red, buffer sizes, la dirección IP de la tarjeta Ethernet, y otros. Todos ellos son reunidos dentro de un header file (rtl_lwipopts.h). También el path del archivo rtl.mk debe estar propiamente situado en el Makefile. Una vez que esto haya sido hecho, solo tendremos que compilar (make) los módulos generados son TCP/IP stack y Dynamic Memory Allocator el cual es necesario por el stack.

Para insertar el stack es necesario iniciar rtlinux, el modulo de memoria, el módulo de la tarjeta y posteriormente el stack. Una vez insertado, el stack ofrece un socket como API para las RT-tasks que puedan se usadas para enviar y recibir paquetes sobre la red. Los sockets API están construidos sobre el secuencial API (el cual es similar al BSD socket API). El secuencial API es un API de alta ejecución por que es diseñado para tomar ventaja de los diseños internos del stack. RTL-lwip también ofrece otro API llamado raw API el cual es no solo más rápido en términos de ejecución, pero es también menos intensivo en la memoria.

Para escribir una aplicación en tiempo real usando RTL-lwIP, se utiliza las comunicaciones de API descritas arriba y al menos una vez OSEL API (Operating System Emulation Layer). OSEL API da acceso a llamadas del sistema operativo y estructuras de datos (thread managing, semaphores, interrupts, etc.) y debe ser usado al menos para crear threads de comunicación.

Para la adecuada comunicación entre el Sensor, Control y Actuador se realizan las siguientes modificaciones en la tarjeta para cada uno de ellos, estos cambios se realizan en rtl_lwipopts.h de RTL-lwIP-0.4 para cada uno de los nodos utilizados en la red.

3.4.- Conclusiones

En el desarrollo del sistema distribuido en tiempo real es necesario tener claro los conceptos y características tanto de tiempo real como de sistema distribuido, conceptos que se manejan desde el inicio hasta el fin de este trabajo, con respecto a Rtlinux y Linux es importante conocer las diferencias y relación entre ellos de esta forma no confundir que RTLinux esta sustituyendo a Linux puesto que el primero es una extensión del kernel de Linux para trabajar en tiempo real. La aplicación RTL-lwIP es necesaria en todo sistema distribuido en tiempo real la cual se encarga del manejo de la tarjeta de red, manejo de la memoria y el stack sin esta aplicación nuestra aplicación no podría estar dentro de una red, RTL-lwIP nos proporciona además una serie de ejemplos para la comunicación entre computadores bajo el modelo cliente servidor. La adecuada configuración de cada uno de nuestros nodos hace posible la comunicación entre ellos, hay que poner especial atención a los archivos que involucran esta configuración como rtl_lwip.h y makefile los cuales se encuentran dentro de RTLinux. Existen 3 tipos de versiones de RTL-lwip se elige la versión 4 por los ejemplos contenidos en ella.

Capítulo 4

4.- CASO DE ESTUDIO

El caso de estudio se centra en garantizar el desempeño en la ejecución de un proceso en un límite máximo de tiempo sin pérdida de datos en un Sistema Distribuido en Tiempo Real, realizando las pruebas necesarias para el Control de los paquetes (datos) por medio de threads (procesos) en cada uno de los nodos.

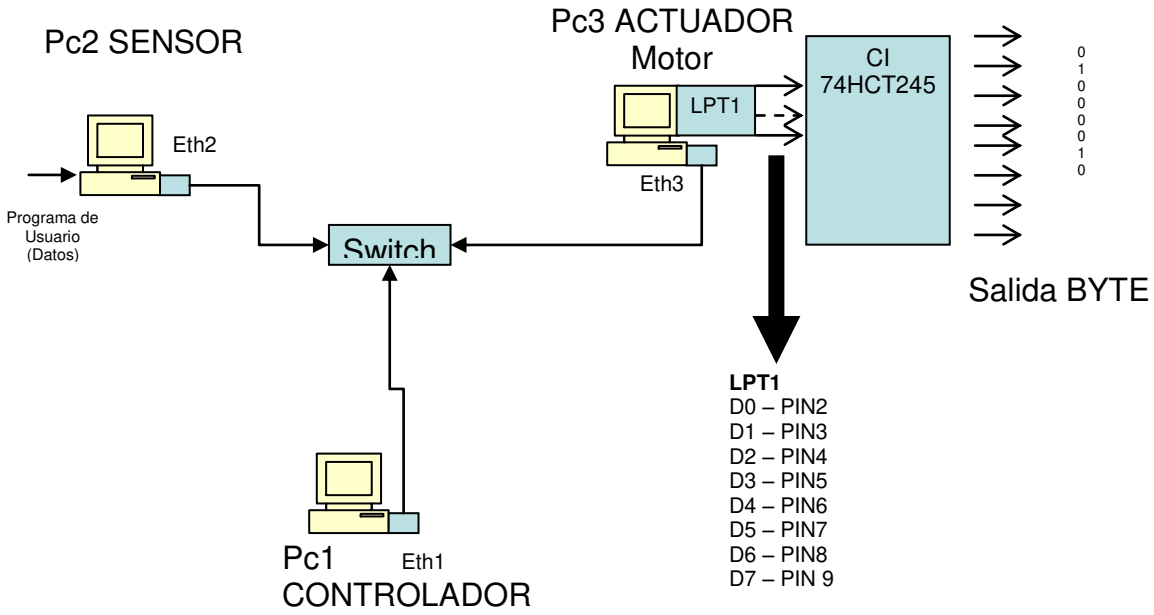


Figura 4.1 Conexión del hardware utilizado y datos de entrada y salida del sistema distribuido en tiempo real.

En la figura 4.1 se muestran los nodos necesarios en el sistema distribuido en tiempo real a implementar, el primer nodo es el de Sensor, al cual se le ingresan datos a través de un programa de usuario posteriormente Sensor envía datos a Control y Control envía a su vez datos al Actuador, Sensor contiene los datos iniciales y Actuador enviará una señal al puerto paralelo en cuanto reciba la llegada de un paquete x desde Control.

Modelo Cliente-Servidor

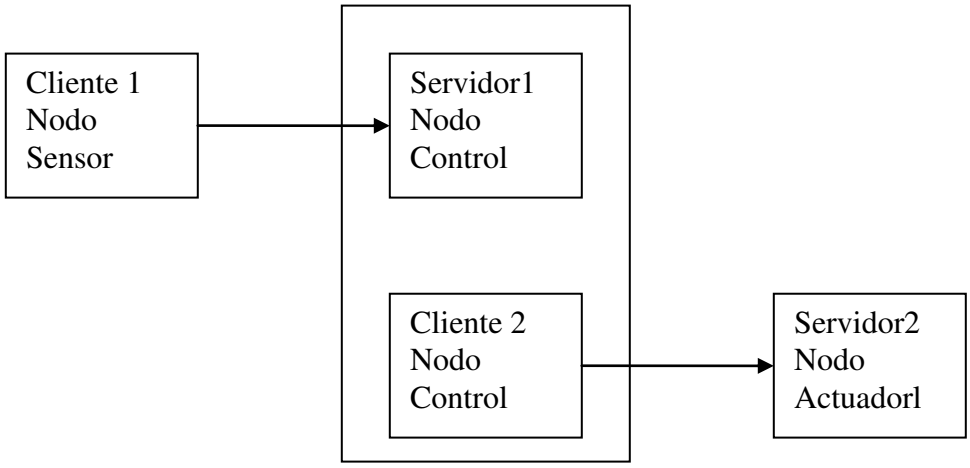


Figura 4.1.1 Modelo Cliente- Servidor en el sistema distribuido en tiempo real.

En la figura 4.1.1 indica el modelo que utilizamos en esta investigación que es el de cliente-servidor con la característica de que el nodo Control trabaja tanto como cliente como Servidor1, el nodo Sensor se desempeña como cliente y el nodo Actuador como Servidor 2. El nodo Control es necesario que en primer instancia trabaje como Servidor1 para que reciba las peticiones de conexión del cliente (Sensor) después de recibir los paquetes, el nodo Control ahora se desempeñará como cliente y realizará las peticiones al servidor (Actuador). El flujo de los datos es de cliente a servidor, primero de nodo Sensor a Control y segundo de Control a Actuador.

4.1 Tiempo promedio y máximo de transporte de los paquetes

En la medición del tiempo de traslado requerido de los paquetes a través de la red necesitamos conocer el tiempo promedio de envío de los paquetes, se utilizan hilos con prioridades, periodicidad, consumo de procesador y el uso de un planificador round robin o FIFO, para esto el tiempo promedio de envío de los paquetes así como el tiempo máximo de envío nos ayudan a establecer la periodicidad de nuestro proceso (hilo). La periodicidad del proceso encargado del envío de los paquetes debe ser mayor al tiempo máximo de envío de paquetes desde un modelo cliente servidor, en donde el cliente es el nodo Sensor y el servidor el nodo Control.

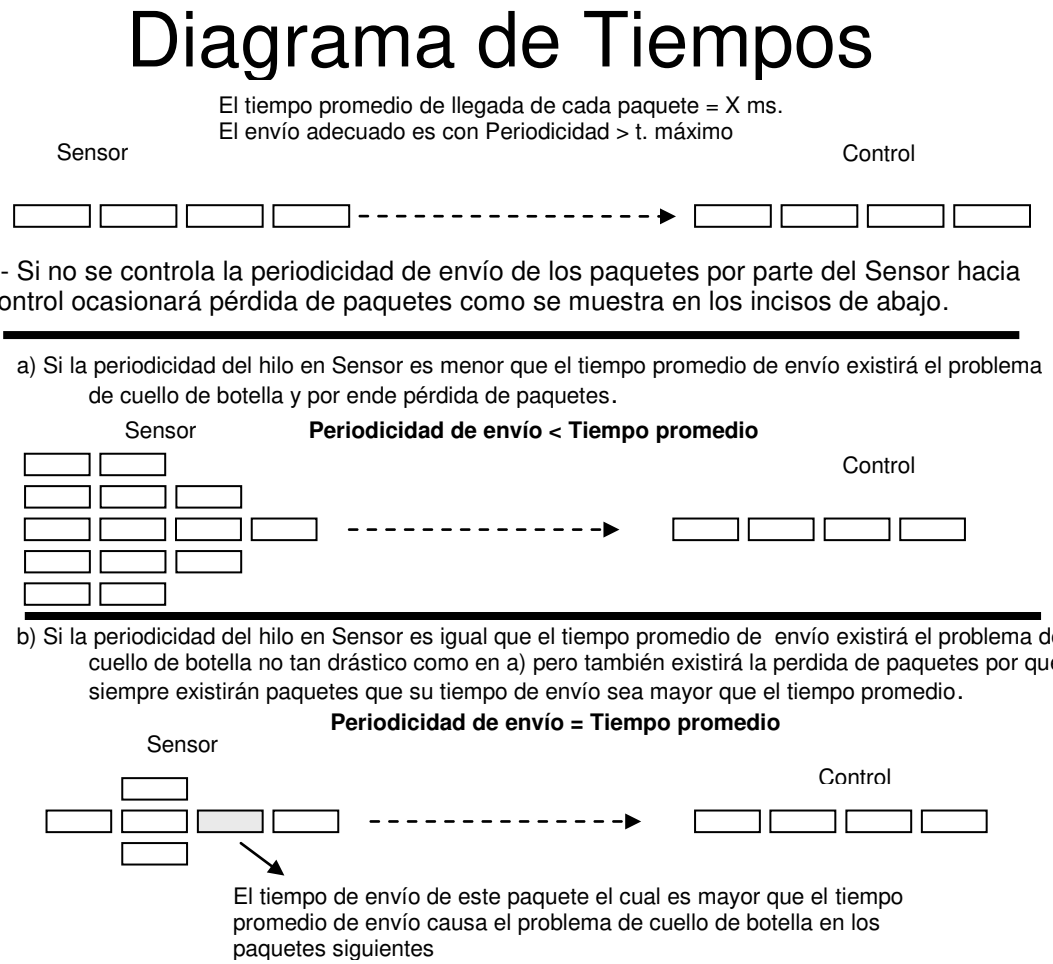


Figura 4.2.- Relación de tiempo promedio de envío de paquetes con la periodicidad de envío.

En la figura 4.2 se muestra el problema de pérdida de paquetes desde Sensor a Control si no ajustamos la periodicidad de la tarea encargada del traslado de paquetes con el tiempo promedio de traslado de cada paquete y el tiempo de traslado máximo del paquete; Si la periodicidad del hilo que contiene la tarea de traslado es menor que el tiempo promedio de envío en algún momento el tiempo de traslado de un paquete podrá superar al tiempo de periodicidad de envío de los paquetes en este momento se ocasiona la pérdida de paquetes; de una manera similar si la periodicidad del hilo que contiene la tarea de traslado es igual que el tiempo promedio de envío en algún momento el tiempo

de traslado de un paquete podrá superar al tiempo de periodicidad de envío de los paquetes en este momento se ocasiona la pérdida de paquetes. De acuerdo a este comportamiento la periodicidad del hilo que contiene la tarea de traslado debe de ser mayor que el tiempo de traslado máximo del paquete.

Para la medición del tiempo que utiliza un paquete en la red es necesario utilizar tareas con periodicidad establecida y un planificador de tareas, una de las tareas debe de estar dedicada al envío del paquete de esta forma podremos establecer un tiempo inicial de envío de cada paquete el cual esta regido por la periodicidad de la tarea.

4.2.- Tiempo utilizado por un paquete desde Sensor a Control

El objetivo en esta etapa es calcular el tiempo que requiere un paquete en llegar desde Sensor hasta Control, estableciendo una periodicidad X en la tarea 1 de Sensor que es la que envía el paquete, sabremos que se enviará un paquete cada periodo X, si después de enviar el paquete se realiza una espera de Y que me asegure que el paquete llegará a Control.

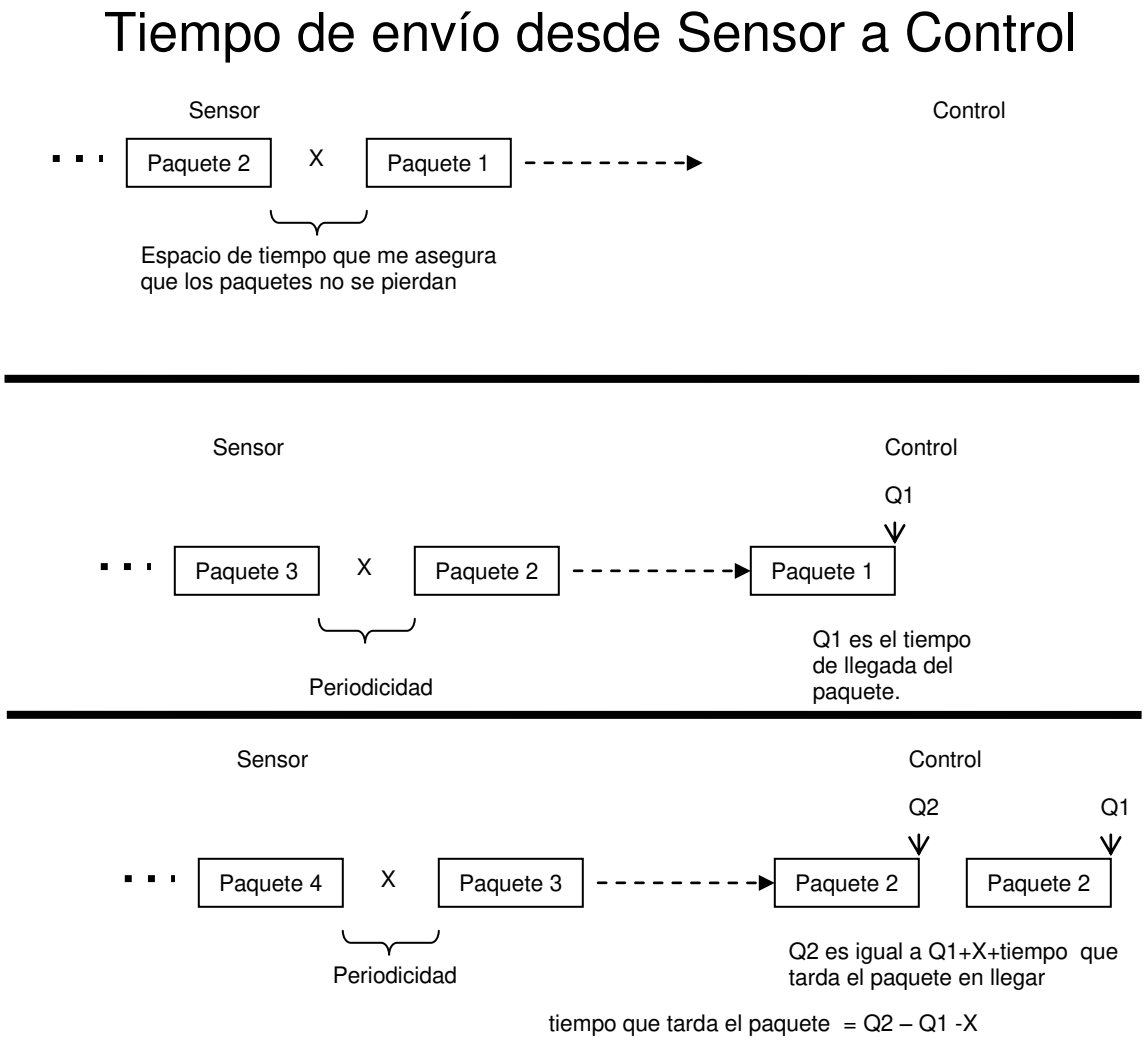


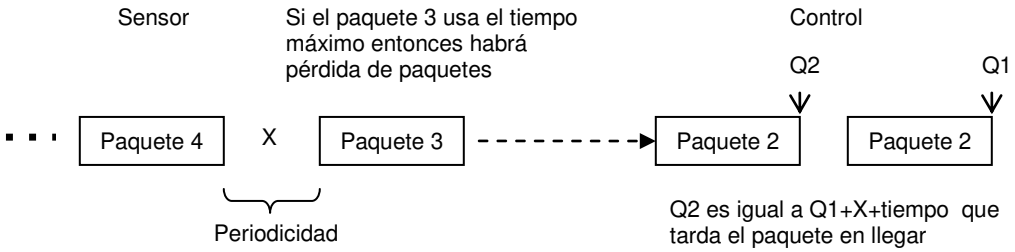
Figura 4.3.1.- Tiempo de envío desde Sensor (cliente) a Control (servidor).

En la figura 4.3.1 Control recibe el primer paquete en un tiempo Q1, el siguiente paquete lo recibirá en un tiempo final de paquete (Q2) = Q1+ X + tiempo que tarda el paquete. Como el tiempo Q1 lo podemos conocer en Control así como la periodicidad de Sensor entonces el tiempo que tarda el paquete = tiempo final (Q2) – Q1 - X con esta fórmula se puede calcular el tiempo que tarda cada paquete excepto el primer paquete.

Relación Periodicidad con Tiempo máximo

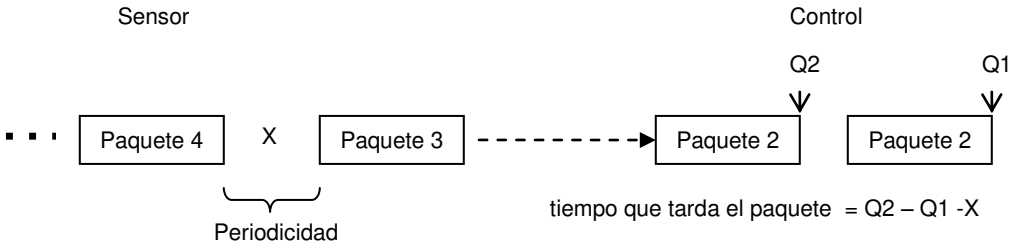
a) Si la periodicidad X del hilo en Sensor es mayor que el tiempo promedio de envío se realiza el envío y recepción de cada paquete sin embargo si algún paquete utiliza un tiempo máximo de envío habrá pérdida de paquetes.

Periodicidad de envío > Tiempo promedio



b) La periodicidad del hilo en Sensor debe ser el tiempo X de envío del paquete + espacio Y.
El espacio Y es el tiempo promedio mas una porción de tiempo que me asegura el envío y llegada de un paquete con tiempo máximo de traslado.

Periodicidad de envío > Tiempo máximo



Periodicidad de envío > X+Y
Periodicidad >Tiempo Máximo

Figura 4.3.2.- Relación del tiempo máximo con la periodicidad del hilo que envía los paquetes.

En la figura 4.3.2 se muestra la forma en que se obtiene el tiempo requerido por un paquete en trasladarse desde Sensor a Control. La periodicidad del hilo que contiene la tarea de envío en Sensor debe ser mayor que el tiempo promedio de envío de los paquetes; para evitar la pérdida de datos causada por “el cuello de botella”; En el nodo Control se recibe el primer paquete en un tiempo Q1, el siguiente paquete se recibe en el tiempo Q2 mas la periodicidad del hilo en Sensor mas el tiempo que tarda el paquete en trasladarse. La periodicidad X es el tiempo que me asegura que los paquetes no se pierdan; si algún paquete utiliza el tiempo máximo de traslado la periodicidad del hilo en Sensor que envía el paquete debe ser X+Y el cual evita la pérdida de paquetes aún cuando el paquete ocupe el tiempo máximo de traslado, siendo Y la diferencia entre el tiempo promedio de traslado de los paquetes y el tiempo máximo utilizado por un paquete en trasladarse.

Solución

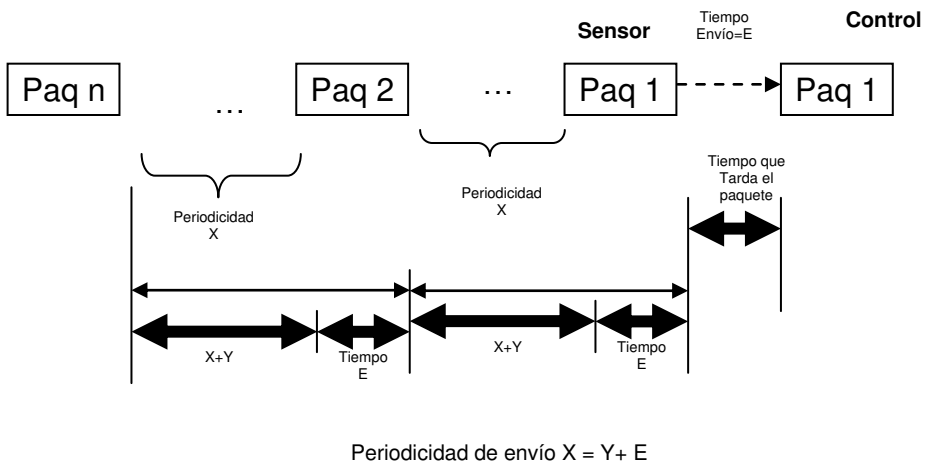


Figura 4.4.- Espacio de tiempo requerido para el envío de cada uno de los paquetes.

En la figura 4.4 se muestra la periodicidad utilizada por el hilo en Sensor encargado de enviar los paquetes, La periodicidad del hilo debe de contemplar el tiempo máximo de envío del paquete; El tiempo E es el tiempo de envío de cada paquete el cual puede variar.

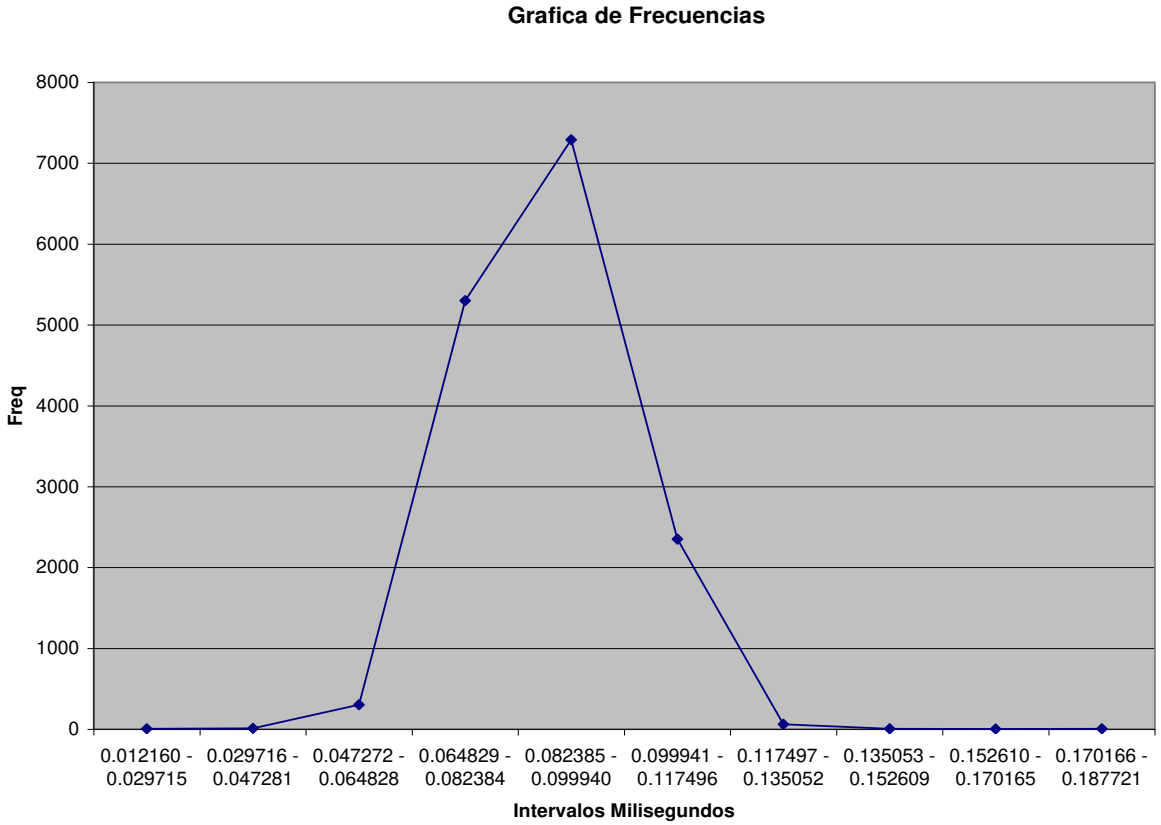


Figura 4.5.- Gráfica donde nos muestra en que intervalo está el grueso de los tiempos utilizados en el traslado de los paquetes.

En la gráfica de frecuencias (figura 4.5) se muestra que la mayoría de los paquetes enviados se encuentran en un intervalo de 0.064 ms (64,829 ns) a 0.117 ms (117,496) nanosegundos de una muestra de aproximadamente 15000 paquetes de 80 caracteres.

Diagrama de tiempos de envío de los paquetes

Para la realización de la gráfica de tiempos de los paquetes se utilizaron cerca de 15000 paquetes.

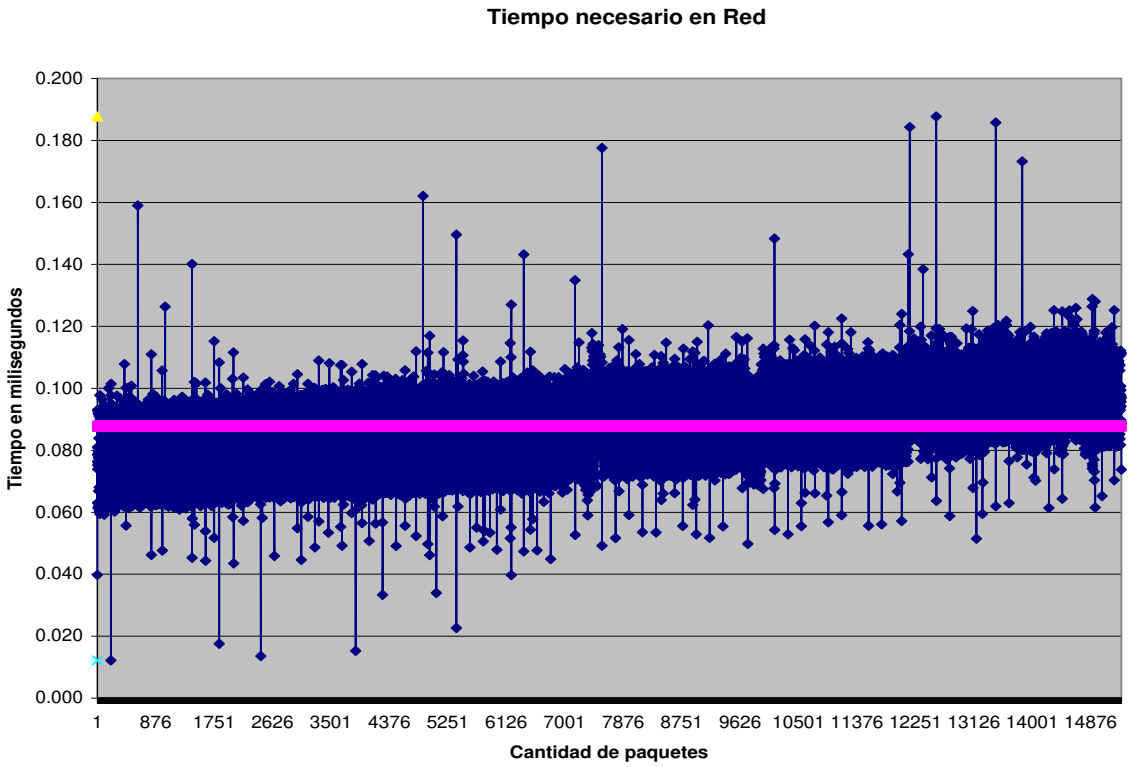


Figura 4.6.- Gráfica de los tiempos utilizados en el traslado de los paquetes en un sistema distribuido en tiempo real.

En la figura 4.6 se muestra el comportamiento en el traslado de los paquetes en el envío de datos desde Sensor hasta Control, en la grafica también observamos de algunos tiempos que alcanzan las 2 décimas de milisegundo (200,000 ns, tiempo máximo) este tiempo o uno mayor puede ser la periodicidad de envío y recepción de la tarea de Sensor, Control y Actuador. El tiempo promedio de traslado de los paquetes es de 0.087milisegundos. La periodicidad en el hilo de Sensor que envía los paquetes puede tomar valores desde 0.087 ms hasta 0.2 ms siendo el más adecuado valores arriba de 0.2 ms asegurándome que no habría pérdidas de paquetes.

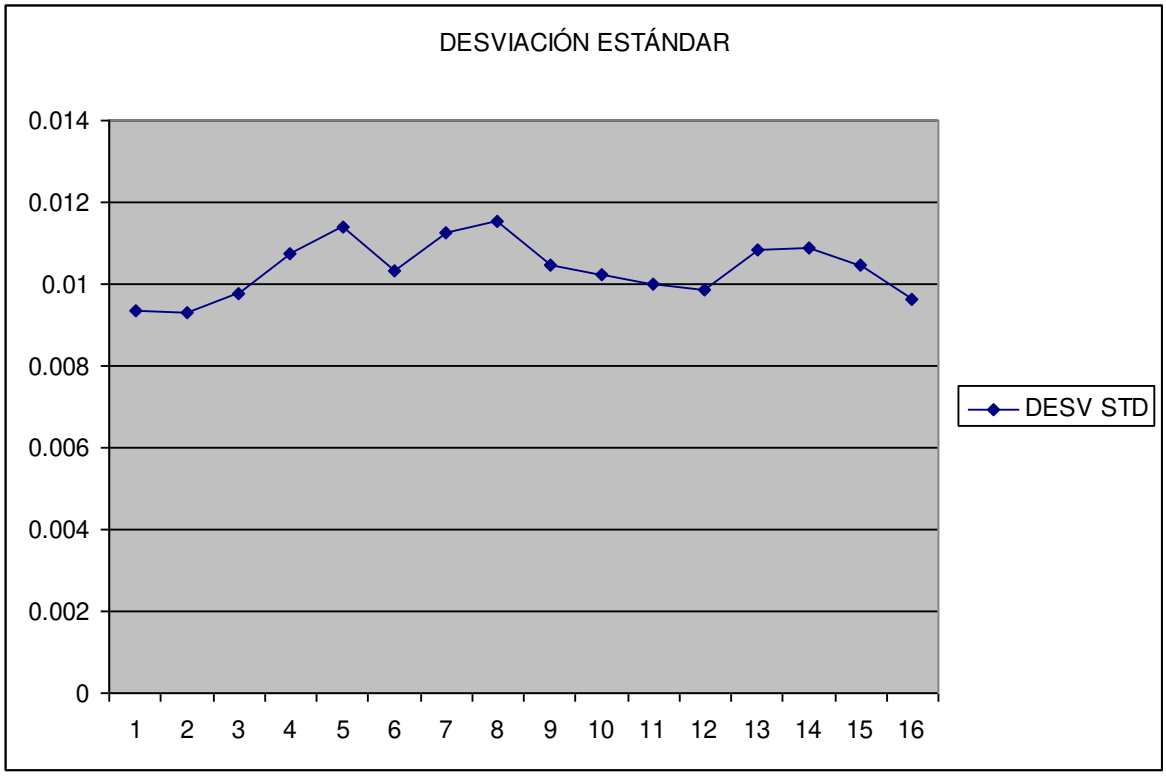
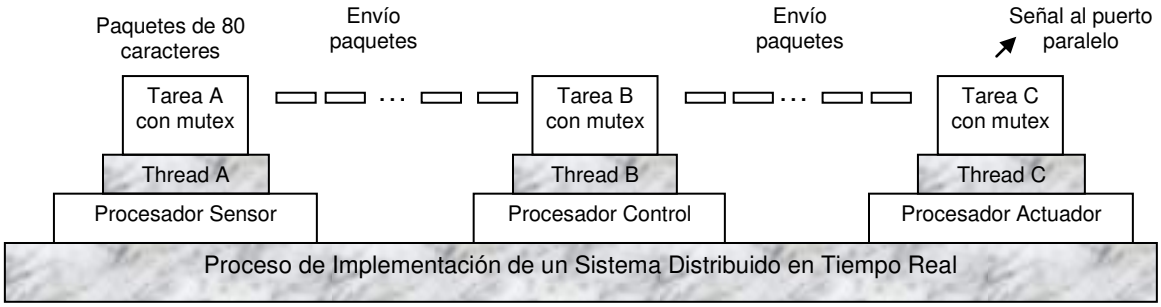


Figura 4.6.1.- Desviación estándar de los tiempos utilizados en el traslado de los paquetes en un sistema distribuido en tiempo real.

En la figura superior se muestra la variabilidad de los datos, estos pequeños cambios se encuentran en el rango de 0.009 milisegundos y 0.012 milisegundos.

Diagrama de Procesos



Los procesos en fondo gris son aquellos que consumen tiempo (Threads A, B y C) de procesador, se cuenta con 3 procesadores Sensor, Control y Actuador.

Figura 4.6.2.- Diagrama de los procesos en cada uno de los nodos dentro de un sistema distribuido en tiempo real.

En la figura 4.6.2 se presenta la forma en que las tareas utilizan el procesador, cada una de las tareas que se encuentran en los módulos son los encargados de ocupar el procesador, todas las tareas ocupan MUTEX para utilizar de forma exclusiva a su procesador, la tarea A se encarga del envío de los paquetes, la tarea B de la recepción y posteriormente de envío de paquetes a Actuador y la Tarea C de recibir paquetes y enviar una señal al puerto paralelo.

Diagrama de tiempos

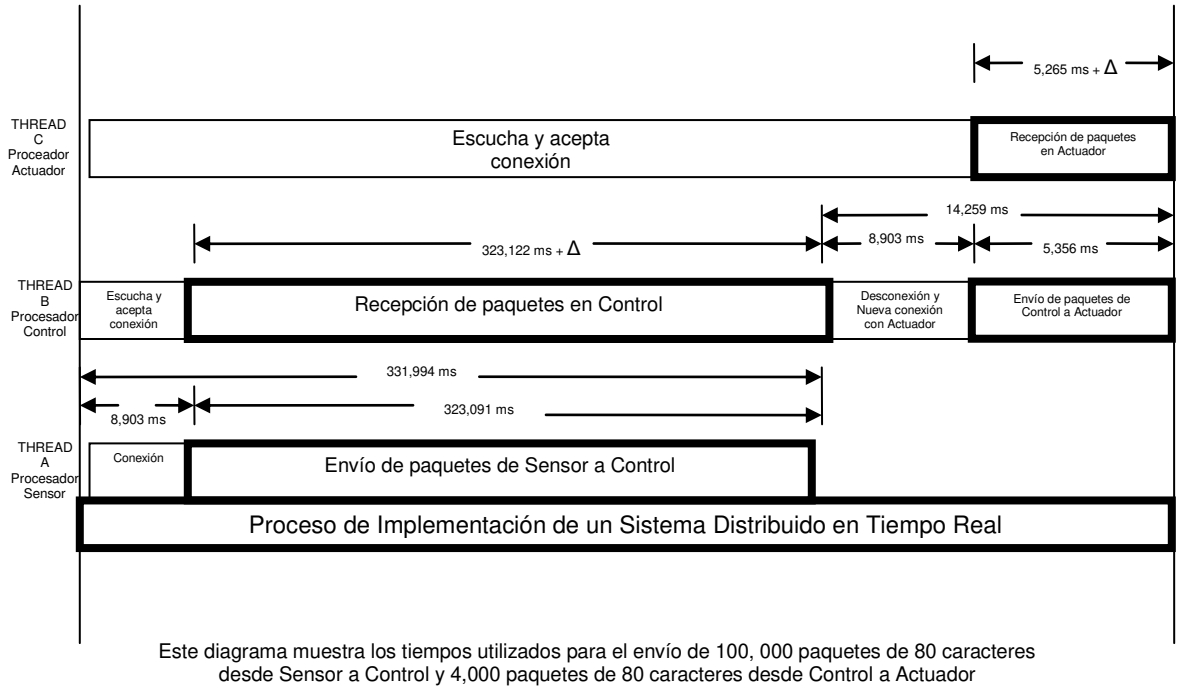


Figura 4.7.- Diagrama de los tiempos utilizados en cada uno de los nodos dentro de un sistema distribuido en tiempo real.

En la figura 4.7 se muestra los tiempos utilizados por cada uno de los nodos, el nodo de Sensor realiza su conexión y posteriormente envía los paquetes al Servidor (Control) durante aproximadamente 300 segundos. El nodo de Control inicia necesariamente primero que los demás módulos ya que se encarga de escuchar las peticiones de Sensor, es por eso que en la grafica la conexión de Control esta ligeramente antes que la de Sensor y Actuador, además el comportamiento del nodo de Control es inicialmente como servidor y posteriormente como cliente para accesar al nodo Actuador. El nodo de Actuador por ser un servidor a la espera de paquetes por parte de Control tiene que esperar (escucha) hasta que le llegue una petición de Control y realizar su tarea antes tiene que realizar su conexión.

4.3.- Variables que se utilizan en el Sistema Distribuido en Tiempo Real

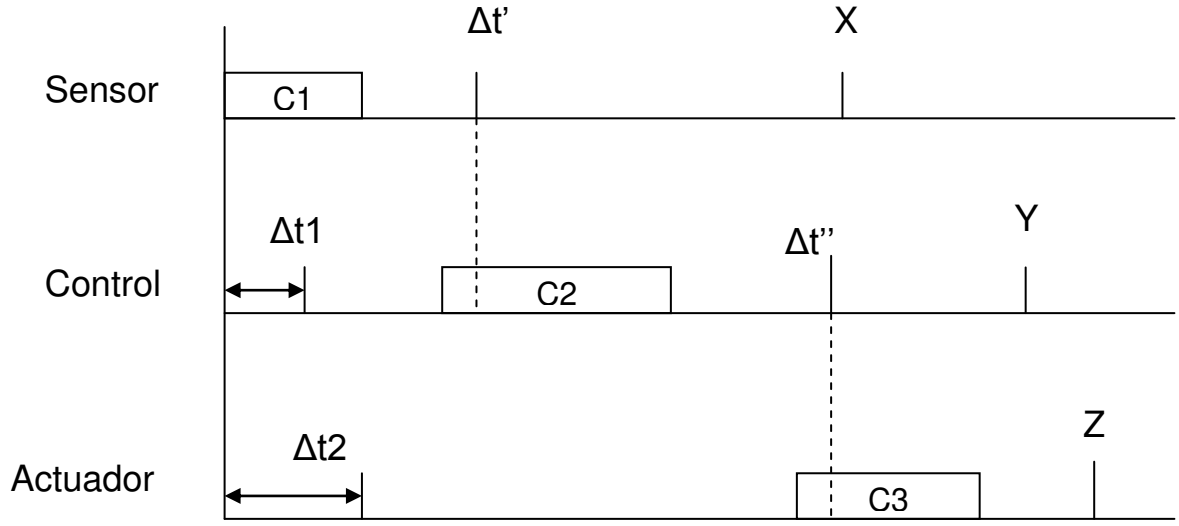


Figura 4.8.- Diagrama de tareas en un sistema distribuido en tiempo real.

En un sistema distribuido en tiempo real (figura 4.8) contamos con 3 tareas una para cada nodo Sensor, Control y Actuador, C es el tiempo de consumo en milisegundos de las tareas de cada nodo que se ejecuta con una periodicidad en milisegundos y una Prioridad PR además del uso de un planificador Round Robin, FIFO, etc. C1 es el tiempo de consumo de la tarea de Sensor (envío del paquete); C2 es el tiempo de consumo de la tarea de Control (recepción del paquete y desplegado en pantalla del paquete) y C3 es el consumo de procesador de la tarea en Actuador (recepción de paquete y envío de señal al puerto paralelo; X, Y y Z es la periodicidad de cada tarea para cada procesador y Δt n es la diferencia tiempo. Sin embargo este sistema distribuido en tiempo real no se pudo llevar a cabo, la planificación con 3 tareas solamente se pudo realizar localmente (en cada nodo) debido a que nuestro nodo Controlador no puede estar conectado simultáneamente como Servidor 1 para atender a las peticiones del nodo Sensor (cliente 1) y como cliente 2 para ser atendido por el nodo Actuador quien actúa como servidor 2 en nuestro sistema. Una de las posibles soluciones fue la de integrar una tarjeta más en el nodo Control pero Rtlinux3.2-pre2 en conjunto con RTL-lwip aún no la soportan.

Las pruebas de ejecución del sistema se realizan variando el número de tareas en cada nodo con 3 y 1 tareas, tiempo de consumo, prioridad, periodicidad y utilizando un planificador FIFO o Round Robin, en las pruebas realizadas el recorrido de los paquetes va desde el nodo Sensor al nodo Control y posteriormente desde Control a Actuador. Existe una desconexión por parte del nodo Control con el nodo Sensor antes de conectarse al nodo Actuador y terminar el proceso.

En el nodo Sensor la tarea 1 se le establece el tiempo de consumo de procesador C de ejecutar una impresión del tiempo durante 3 centésimas de milisegundo (C=0.03) y enviar el paquete de 80 caracteres al nodo Control, prioridad igual a 1 (P=1) y Periodicidad igual a 2 milisegundos (PR=2). Para la tarea 2 el consumo de procesador es de 0.005 milisegundos (C=0.005) , prioridad igual a 2 (P=2) y Periodicidad igual a 50 milisegundos. Para la tarea 3 el consumo de procesador es de 0.004 milisegundos (C=0.004), prioridad igual a 3 (P=3) y Periodicidad (PR) igual a 50 milisegundos

En el nodo Control la tarea 1 se le establece el tiempo de consumo C de procesador de ejecutar una impresión de tiempo durante 4 centésimas de milisegundo (C=4), tiene una prioridad 1 y su periodicidad es igual a 2 milisegundos (PR=2). Similar a Sensor para la tarea 2 el consumo de procesador es de 0.005 milisegundos (C=0.005), prioridad igual a 2 (P=2) y Periodicidad igual a 50 milisegundos. Para la tarea 3 el consumo de procesador es de 0.004 milisegundos (C=0.004), prioridad igual a 3 (P=3) y Periodicidad (PR) igual a 50 milisegundos, (Tabla 4.1).

Tarea	**C Tiempo consumo milisegundos**	**P Prioridad**	**PR Periodicidad milisegundos**	**Planificador**
Sensor				
1	0.03	1	2	FIFO
2	0.005	2	50	
3	0.004	3	50	
Control				
1	0.04	1	2	FIFO
2	0.005	2	50	
3	0.004	3	50	

Tabla 4.1.- Planificación en Sensor y Control.

Con una periodicidad de 2 milisegundos los retardos para la ejecución de la tarea 1 en Sensor estarán entre 0.000112 milisegundos y 91 milisegundos, puesto que no se tiene una medida exacta en el tiempo que tarda el paquete en llegar desde un Cliente (Sensor) a un servidor (Control) los retardos no son constantes o tienen un espacio muy amplio entre el tiempo menor de retardo y el tiempo mayor de retardo.

Si la periodicidad de la tarea 1 de Sensor la incrementamos (PR=100) los retardos de ejecución se vuelven más estables (0.097 milisegundos) para el envío de 10 paquetes, y el retardo para la ejecución de la tarea 1 en Control en promedio es de 0.24 milisegundos, tal como se muestra en la tabla 4.1.1.

Nodo	**C Tiempo consumo milisegundos**	**P Prioridad**	**PR en ms (periodicidad)**	**# paquetes Enviados Recibidos**	**Retardos milisegundos**
Sensor Tarea 1	0.03	1	100	10	0.097
Control Tarea 1	0.04	1	100	10	0.24

Tabla 4.1.1.- Retardos de la tarea 1 en Sensor y Control.

Con esta misma periodicidad de 100 ms. y variando el número de paquetes a 1000, enviados desde Sensor a Control, el promedio de retardos en la tarea 1 de Sensor es de 0.09780 milisegundos y para la tarea 1 de Control de 0.1999 ms. además la sincronía de envío y recepción de paquetes es adecuada debido a que el mismo numero de paquetes que envía Sensor es el mismo número que recibe Control.

Variando el número de paquetes a 10,000 el promedio de los retardos de las tareas son, para Sensor 0.1020 ms. y para Control de 0.1999 ms.

Los paquetes enviados y recibidos así como los retardos en la tarea 1 de cada nodo son:

Cliente	
Paquetes Enviados	Retardos
10	0.097
1000	0.0978
2500	0.102
5000	0.102
10,000	0.102

Servidor	
Paquetes Recibidos	Retardos
10	0.24
1000	0.1999
2500	0.199
5000	0.199
10,000	0.199

Tabla 4.1.2 Retardos con diferentes cantidades de paquetes en cliente y servidor

La periodicidad de las tareas tanto de Sensor, Control y Actuador debe de ser mayor que el tiempo máximo de traslado de los paquetes en un sistema distribuido para no tener pérdida de paquetes y que los retardos de las tareas en cada nodo (Cliente y Servidor) sean uniformes como se muestra en la figura 4.9.

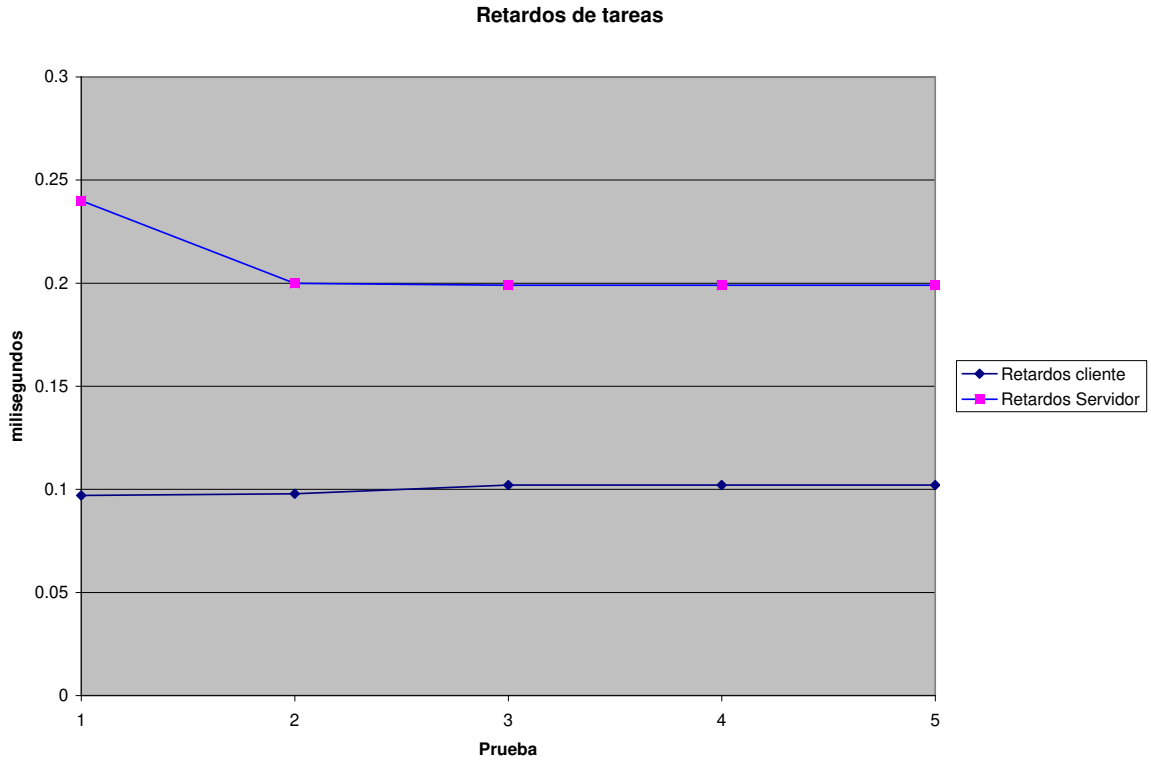


Figura 4.9.- Diagrama de retardos de tareas en un sistema distribuido en tiempo real.

Si la periodicidad de las tareas en cada uno de mis nodos es mayor a 100 ms. la sincronización en el envío y recepción de los paquetes de un cliente al servidor se dará y los retardos de sus tareas serán uniformes. Si la periodicidad entre los nodos se reduce se pierde la sincronía en el envío y recepción de los paquetes y la diferencia entre los retardos de las tareas será amplia.

Envío y recepción de paquetes (sincronía)

Figura 4.10.- Envío de paquetes (sincronía)

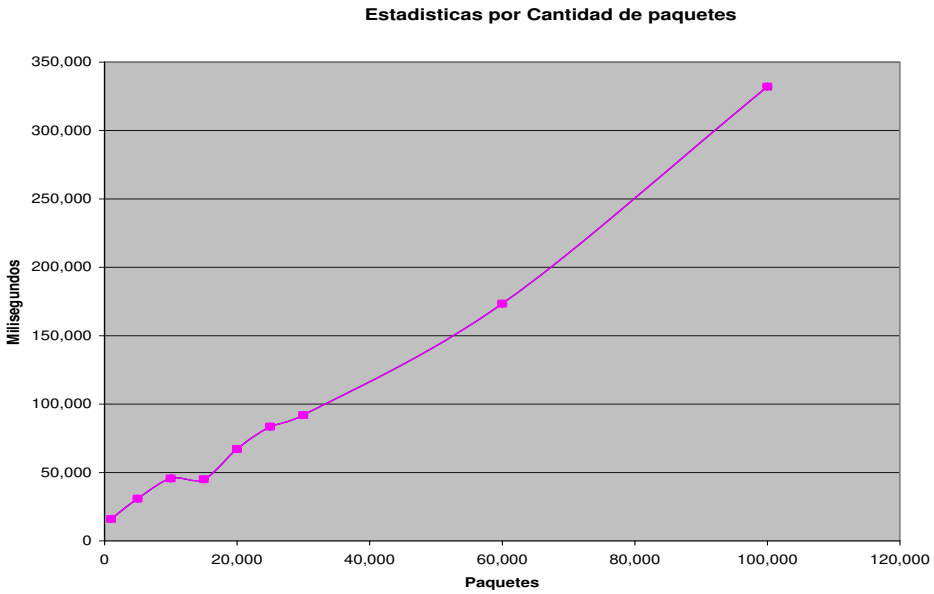
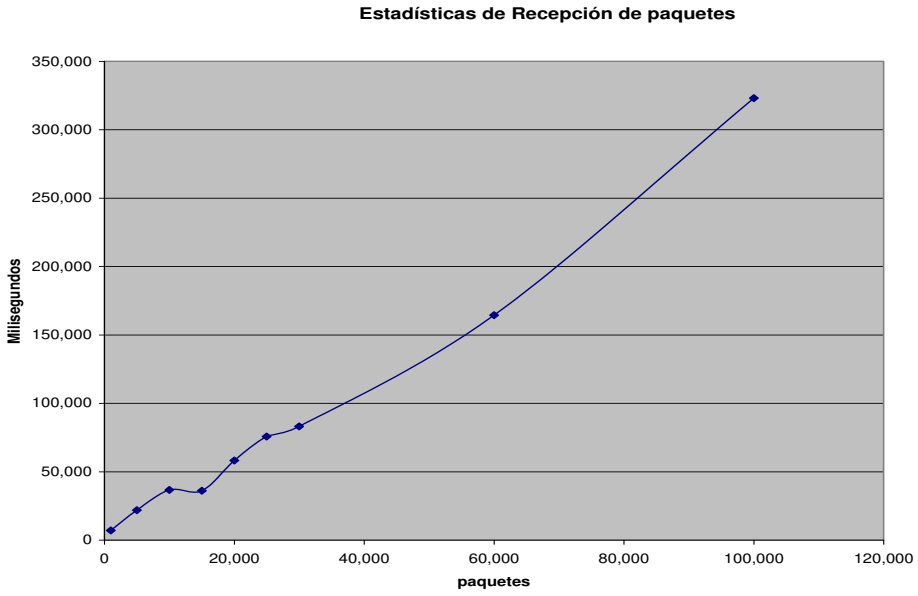


Figura 4.11.- Recepción de paquetes (sincronía)



La sincronía en el envío y recepción de paquetes es adecuada se muestra en las graficas de Control y Sensor el mismo comportamiento en el envío como en la recepción de los 100,000 paquetes.

4.4.- Conclusiones

Podemos contar con un planificador dentro de cada uno de los nodos además de manejar prioridades, periodicidad y tiempo de consumo de cada una de las tareas, se realizaron pruebas con planificadores FIFO y Round Robin, contando con 3 tareas en los nodos. Utilizando planificadores FIFO en cada uno de los nodos, si la periodicidad de la tarea de envío en el cliente es de 2 milisegundos los retardos no serán uniformes pero si la periodicidad de la tarea de envío es mayor de 100 milisegundos la sincronía en el envío y recepción de paquetes se dará además de tener retardos uniformes en las tareas.

Capítulo 5

5.- PRUEBAS Y ANALISIS DE RESULTADOS

Las pruebas realizadas se dividieron en seis incisos en los que se establecen las condiciones adecuadas para cada uno de los nodos y posteriormente se efectúan pruebas entre los nodos para realizar las graficas y mediciones de paquetes tanto enviados como recibidos así como los retardos de los paquetes y planificadores utilizados en cada uno de los nodos.

En el inciso a el objetivo ingresar una flujo de paquetes de 80 caracteres por medio de una aplicación desde el espacio del usuario hacia el cliente (nodo Sensor) con ayuda de estructuras y archivos fifo después de recibir cada paquete enviarlo desde el cliente al servidor (Sensor a Control posteriormente desde Control a Actuador), previamente establecimos el comportamiento de cada nodo como servidor o cliente de acuerdo a la necesidad de comunicación. En el caso de nuestro sistema es necesario que el nodo Sensor se comporte como cliente 1, el nodo Control como Servidor 1 y finalizando el traslado del flujo paquetes cierre la conexión que se tiene con Sensor y ahora se comporte como un cliente 2 que solicita conectarse al nodo Actuador que se comporta como un servidor 2.

Para el inciso b el objetivo es medir el tiempo requerido por un paquete en trasladarse desde un cliente a servidor (puede ser de Sensor a Control o de Control a Servidor) los paquetes son de 10 caracteres, posteriormente el número de paquetes n enviados tomará valores desde 1 hasta 2200 además mediremos la cantidad de paquetes perdidos en cada prueba ejecutada y calcular el porcentaje de paquetes recibidos; controlaremos el flujo de los paquetes para evitar el bloqueo debido al cuello de botella este manejo del envío de cliente a servidor se realizara utilizando un planificador de tareas y utilizando una periodicidad para cada tarea.

En el inciso c para el envío y la recepción del flujo de paquetes utilizamos la técnica MUTEX para sincronizar los envíos y que las mediciones de las llegadas de los paquetes sean más exactas además elevaremos la cantidad de paquetes enviados desde cliente a servidor (Sensor a Control o Control a Actuador) utilizando MUTEX, planificador FIFO, hilos y periodicidad, prioridad y tiempos de consumo de tareas. Enseguida realizaremos la medición del tiempo requerido por un paquete en trasladarse desde Sensor a Control utilizando una Periodicidad X en la tarea de envío del paquete e incrementamos la cantidad de paquetes además mediremos el tiempo utilizado por cada paquete en trasladarse desde Sensor hasta Control y poder realizar una grafica que nos muestre el comportamiento del tiempo utilizado por los paquetes

Inciso d, aquí integramos al sistema distribuido el nodo Actuador el cual recibirá datos desde Control, el nodo Control en primera instancia trabajará como servidor para recibir las peticiones de Sensor que actúa como cliente y posteriormente Control trabajará como cliente para conectarse a al nodo Actuador.

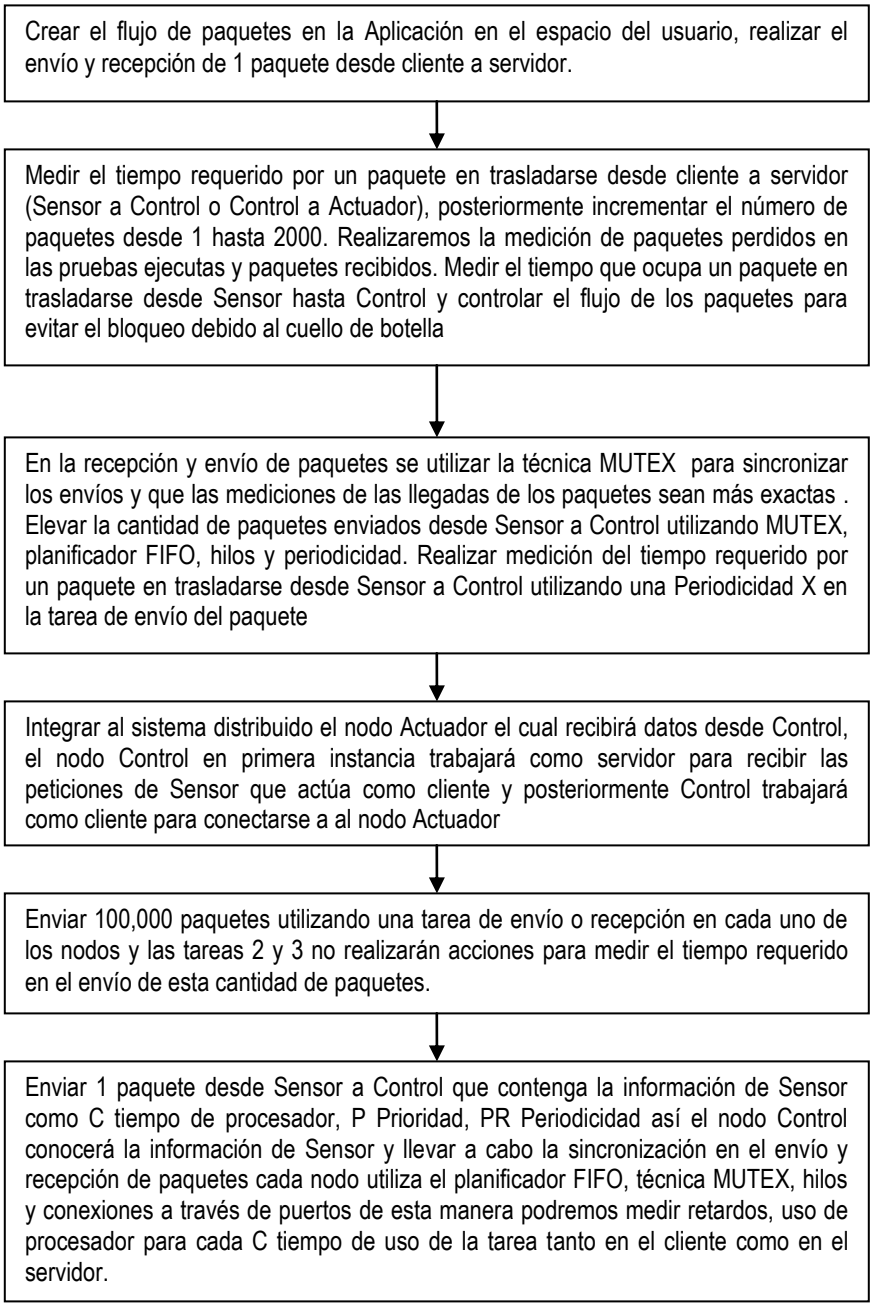
En el inciso e el objetivo es enviar 100,000 paquetes utilizando una tarea de envío o recepción en cada uno de los nodos, las tareas 2 y 3 de cada nodo no realizaran acciones para medir el tiempo requerido en el envío de esta cantidad de paquetes.

Para el inciso f el objetivo es el de enviar un paquete desde Sensor a Control que contenga la información de Sensor como C tiempo de procesador, P Prioridad, PR Periodicidad así el nodo Control conocerá la información de Sensor y llevar a cabo la sincronización en el envío y recepción

de paquetes cada nodo utiliza el planificador FIFO, técnica MUTEX, hilos y conexiones a través de puertos de esta manera podremos medir retardos, uso de procesador para cada C tiempo de uso de la tarea tanto en el cliente como en el servidor.

El tiempo de envío de paquetes es comparado con el estándar SCI (Scalable Coherent Interface) el cual fue diseñado por el comité ANSI/IEEE 1592-1992 quien diseñó una plataforma tanto de software como de hardware capaz de soportar comunicaciones en tiempo real con una baja latencia. Su arquitectura de dos o tres dimensiones ofrece redundancia en caso que se presente alguna falla. Cuando se transfiere un dato en SCI éste se copia directamente a la memoria del nodo remoto: empleando el método conocido como transferencia inteligente. En dicha técnica la tarjeta SCI crea un mapa de direcciones de memoria que corresponden a una región reservada en algún nodo remoto, de esta forma el procesador puede acceder a estos datos de la misma forma que accede a los de la memoria local.
Estas características contribuyen a que la red SCI tenga una baja latencia en la transmisión de los datos. Por ejemplo, empleado una tarjeta D330 PCI-SCI, de la empresa Dolphin Interconnect Solutions Inc., en un computador con un procesador AMD 760 MPX6 en promedio se requieren 3 microsegundos para transferir un paquete de 512 bytes y latencia de 1.46 microsegundos.

Diagrama de bloques de los objetivos de las pruebas



5.1.- Pruebas realizadas

a) Aplicación en el espacio del usuario y envío y recepción de paquetes entre nodos
Objetivos:
i) Ingresar un flujo de paquetes de 80 caracteres por medio de la aplicación en el espacio del usuario al cliente1 (nodo Sensor)

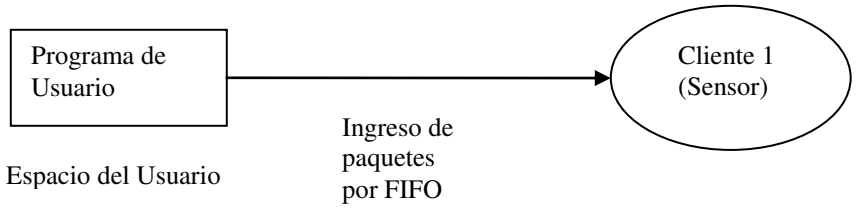


Figura 5.1.- Ingreso de paquetes a través de un programa de usuario

En la figura 5.1 se muestra el ingreso de los paquetes por medio de un programa de usuario que alimentará de paquetes al nodo Sensor por medio de FIFO.

ii) Enviar 1 paquete de Sensor a Controlador (figura 5.2) posteriormente enviar 1 paquete desde Control a Actuador y que en Actuador se envíe una señal al puerto paralelo llevando a cabo este objetivo con éxito podremos conocer como trabaja cada nodo con respecto al modelo cliente servidor.

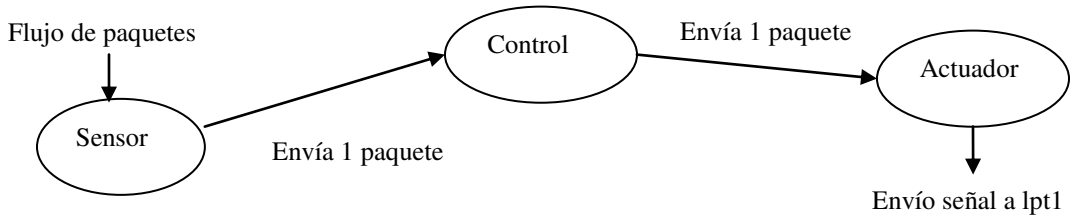


Figura 5.2.- Comunicación entre nodos, Sensor envía a Control y Control a Actuador.

En este paso nuestro objetivo es la de enviar 1 paquete desde cliente 1 (Sensor) a servidor 1 (Controlador) posteriormente enviar 1 paquete desde cliente 2 (Control) al servidor 2 (servidor) y que en Actuador se envíe una señal al puerto paralelo tal como se muestra en la figura superior.

Como hipótesis se establece la posibilidad enviar un flujo de paquetes al cliente 1 (nodo Sensor) por medio de una programa en el espacio del usuario y de enviar 1 paquete desde Sensor a Controlador y posteriormente desde Control a Actuador sin bloqueo de ninguno de los nodos además de que en Actuador se envíe una señal al puerto paralelo.

En esta prueba es necesario ingresar los paquetes al cliente 1 (nodo Sensor) a través de FIFO esto lo ponemos realizar utilizando 3 descriptores (fd0, fd1, ctl) que nos avisarán cuando haya algún dato en los 3 FIFOS utilizados (/dev/rtf1, /dev/rtf2, /dev/rtf3) y realizar tres módulos uno para cada nodo (Sensor, Control y Actuador) el modelo a seguir es el de cliente servidor, las conexiones entre nodos se harán a través del puerto 7 entre el los nodos Sensor y Control y puerto 8 entre los nodos Control y Actuador con el fin de enviar 1 paquete entre cliente y servidor.

i) Ingreso del flujo de paquetes al nodo Sensor a través de FIFO:

En la programa de usuario utilizamos 2 archivos fifos, 1 para el ingreso del flujo y el segundo para verificar cuando se inicia y termina el flujo de paquetes, para establecer la periodicidad a la que se recibirán los paquetes y para conocer el numero de tarea que en este caso es solo una tarea en el programa de usuario. El programa usuario utiliza un ciclo para ingresar la cantidad de paquetes necesarios para las pruebas en el cual nosotros podremos variar desde 1 hasta 100,000 paquetes.

Nuestro programa usuario deberá de ejecutarse como ./Frank_app solo después de que el módulo que extraerá los paquetes de fifo haya sido insertado en el cliente (nodo Sensor)

En modulo la periodicidad del hilo1 se multiplica por 500 000 que es la periodicidad en que van ingresando los paquetes a través de fifo por lo tanto si la periodicidad del hilo 1 es multiplicado por X cantidad el número de paquetes enviados a través fifo variará. Ejemplo si queremos poner la Periodicidad cada 100 000 000 nanoseg, el tiempo que trae el fifo 500 000 multiplicado por 200 entonces X=200.

Si damos valores a X de (100, 150, 2000 y 200) el número de paquetes de 80 caracteres enviados por el programa al cliente 1 (nodo Sensor) es mostrado en la tabla inferior, el paquete lo debemos de establecer en 79 caracteres para que el número de paquetes previstos por enviar sean los enviados. Con esta modificación podemos enviar por fifo la cantidad de paquetes que se requieran y el cliente 1 (nodo Sensor) recibirá exactamente dicha cantidad.

Periodo(500,000) * X	Paquetes por enviar por programa usuario	Paquetes enviados	Paquete
100	1000	500	80 char
150	1000	500	80 char
2000	1000	Uno cada segundo	80 char
200	1000	500	80 char
100	1000	1000	79 char

Tabla 5.1.- Implementación de FIFOs enviando 1000 paquetes.

Se requiere que se envíen exactamente 80 caracteres en cada paquete poner el bufer[81] y al obtener los datos del fifo será con rtf_get[80], los resultados se muestran en la tabla 5.1.1

En modulo

- poner Periodicidad en 500 000 *100 (el tiempo que trae el fifo 500 000) por (100)
- poner rtf_get en 80, quitar el while y dejar un for de 10 001
- data dejarlo en 80 caracteres

En programa usuario:

- poner el for en 10 000 paquetes para realizar la prueba
- bufer [81].

Periodo= (500,000) * X	Paquetes por enviar por programa usuario	Paquetes enviados	Paquete	Tiempo utilizado
500 000*100= 50 000 000	10 000	10 000	80 char	20 paquetes por segundo 500 seg *20=10 000 paq, 07:20 minutos

Tabla 5.1.1.- Implementación de FIFOs enviando 10,000 paquetes.

La periodicidad del hilo 1 podemos establecerla en múltiplos de 500 000 nanosegundos.

ii) Enviar 1 paquete desde cliente 1 al servidor 1:

En este paso tenemos objetivo enviar 1 paquete desde cliente 1 (nodo Sensor) al servidor 1 (nodo Control) posteriormente enviar 1 paquete desde cliente 2 (nodo Control) a servidor 2 (nodo Actuador) y que en Actuador se envíe una señal al puerto paralelo llevando a cabo este objetivo con éxito podremos conocer como trabaja cada nodo con respecto al modelo cliente servidor.

En esta implementación es necesario realizar tres módulos uno para cada nodo, el modelo a seguir es el de cliente servidor. Es necesario que el módulo de Sensor solamente tenga un hilo y que el módulo de Control maneje 1 hilo donde la función recibe el paquete de Sensor y después envíe el paquete a Actuador (tabla 5.2).

ACCION		
• Enviar 1 paquete del Sensor al Controlador después 1 paquete del Controlador a Actuador. • En el Sensor (cliente1) se crea 1 conexión conn, puerto 7 con 1 hilo. • En Controlador (servidor y cliente) conn, puerto 7, conn2, puerto 8 con 1 hilo. • Actuador (servidor2) crea conexión conn, puerto 8 con 1 hilo y salida de 0x00 (cero) por el puerto LPT1 con outb(0x00, 0x378) o outp(888,0) en decimal.		
SENSOR	**CONTROLADOR**	**ACTUADOR**
Cliente1	Servidor y cliente	Servidor2
1 conexión Conn, puerto 7	2 conexiones Conn, puerto 7 Conn2, puerto 8	1 conexión Conn2, puerto 8
1 hilo	1 hilo	1 hilo

Tabla 5.2.- Descripción del envío de 1 paquete desde Sensor a Control enseguida desde Control a Actuador.

Pseudo código para cliente 1 (Sensor):

En este módulo de Sensor solamente se maneja un hilo (con su función), se crea la conexión ligando a su dirección IP y al puerto 7, después se hace la conexión hacia Controlador y se envía el paquete por último se borra la conexión se sale del hilo y removemos el módulo.

Inicio del modulo
    Creación del hilo 1 y función de ejecución 1
    La función de ejecución 1 asigna la dirección IP al cliente (Sensor).
        Crear la conexión conn.
        Se crea la liga entre la conexión conn, dirección IP y el puerto 7.
        Se crea la conexión al servidor (Control) a través del puerto 7.
        Se envía el paquete
        Se borra conexión.
    Salir del hilo 1
Se remueve módulo

Pseudo código en Controlador (trabajando como servidor) cierra la conexión por donde se escucha a Sensor y abre una conexión 2 utilizada para conectarme como cliente hacia Actuador.

Después de inicializar el módulo se crea la conexión, se liga al puerto 7 y a la dirección IP, ingresando en un ciclo donde el servidor (Control) escucha a los clientes (uno de estos clientes es Sensor) se acepta la conexión, se recibe el paquete (por medio de un buffer) y se imprimen los datos contenidos en el paquete, se borra la conexión. Enseguida se crea una conexión 2 en este momento Controlador actúa como cliente se liga e puerto 8 y la dirección IP a la conexión se conecta al Actuador (servidor) y se le envía el paquete.

Inicio del modulo
    Creación del hilo y función de ejecución
    La función de ejecución asigna la dirección IP al servidor(Control).
    Crear la conexión conn.
    Se crea la liga entre la conexión conn, dirección IP y el puerto 7.
    El servidor escucha por medio de la conexión conn.
    Ciclo infinito while (1)
    Se acepta cualquier conexión y su valor se asigna a newconn
    Mientras el buffer buf reciba datos
        Hacer lo siguiente
            Se copian los datos del buffer a la variable data
            Se imprime data
        Mientras el próximo buffer sea mayor o igual a 0
    Se borra buffer buf
    Se borra la conexión conn
    Crear la conexión conn2.
    Se crea la liga entre la conexión conn2, dirección IP y el puerto 8.
    Se crea la conexión al servidor (Control) a través del puerto 7.
    Se envía el paquete
    Se borra conexión.
    Salir del hilo 1
Se remueve módulo

El pseudo código en actuador es el siguiente

Inicializa el módulo se crea la conexión, se liga al puerto 8 y a la dirección IP, ingresando en un ciclo infinito donde el servidor (Actuador) escucha a los clientes (uno de estos clientes es control) se acepta la conexión, se recibe el paquete (por medio de un buffer) y se imprimen los datos contenidos en el paquete, se borra la conexión.

Inicio del modulo
- Creación del hilo y función de ejecución
- La función de ejecución asigna la dirección IP al servidor (Actuador).
- Crear la conexión conn.
- Se crea la liga entre la conexión conn, dirección IP y el puerto 8.
- El servidor escucha por medio de la conexión conn.
- Ciclo infinito while (1)
- Se acepta cualquier conexión y su valor se asigna a newconn
- Mientras el buffer buf reciba datos
  - Hacer lo siguiente
    - Se copian los datos del buffer a la variable data
    - Se imprime data
    - Se envia señal a LPT1
  - Mientras el próximo buffer sea mayor o igual a 0
- Se borra buffer buf

Se remueve módulo

Se observa que el envío del paquete funciona correctamente, los archivos se llaman tcpclient4.c, tcpserver4.c y tcpserver4paralelo.c se encuentran en el anexo, pero el envío de la señal al puerto paralelo no trabajo correctamente. Posteriormente se realizo la configuración del puerto paralelo en el kernel de rtlinux donde se agrego al hardware, después de esto funcionó la señal enviada al puerto paralelo.

Se concluye que una forma de enviar un paquete es la de utilizar Sensor como cliente, Actuador como Servidor y Controlador debe de trabajar primero como servidor y posteriormente como cliente además de no olvidar agregar en el kernel de rtlinux el hardware del puerto paralelo y verificar que funcione correctamente el puerto antes de agregarlo al kernel.

b) Medir el tiempo requerido por los paquetes en trasladarse desde un cliente a un servidor variando el número de paquetes, obtener el porcentaje de paquetes perdidos y recibidos.

Objetivos:

i) Medir el tiempo requerido en enviar n paquetes desde cliente1 a Servidor 1 después desde cliente 2 a Servidor 2 (Control a Actuador).

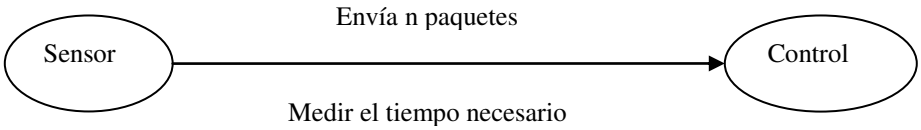


Figura 5.3.- Envío de n paquetes desde Sensor a control para medir el tiempo utilizado

En la figura 5.3 se muestra nuestro objetivo de forma grafica, donde Sensor será el cliente1 y Control el Servidor 1.

ii) Calcular el tiempo que requiere un paquete en trasladarse desde cliente 1 hasta Servidor 1 utilizando periodicidad en hilos.

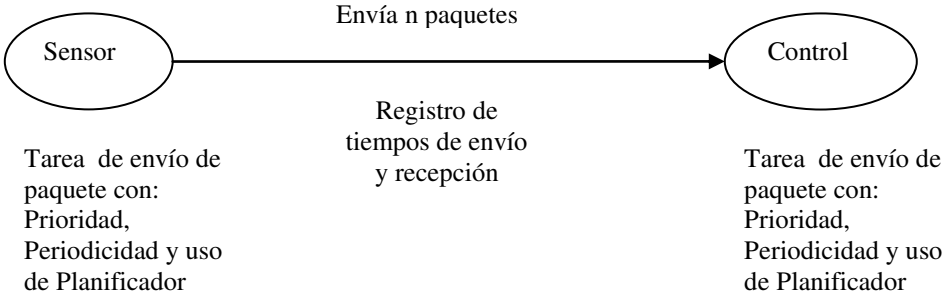


Figura 5.4.- Envío de n paquetes desde Sensor a control utilizando periodicidad en las tareas y contando con un planificador

Se maneja como hipótesis (figura 5.4) la posibilidad de enviar n paquetes desde Sensor hacia Control y realizar las mediciones del tiempo que tarda en llegar el paquete y el posible calculo del tiempo que utiliza un paquete en llegar desde un nodo hacia otro, utilizando tareas con periodicidad de ejecución.

i) Medir el tiempo requerido en enviar n paquetes desde cliente1 a Servidor 1

En este módulo se medirá el tiempo necesario para enviar uno o n paquetes desde Sensor a Control, en primer lugar se crea el hilo (con su función de ejecución), la función de ejecución liga el puerto y la dirección IP después se crea la conexión hacia Control (servidor) y creamos un ciclo desde 0 hasta n para el envío de los paquetes en este momento se registra el tiempo inicial ti de envío del paquete posteriormente se envía el paquete en este instante registramos el tiempo final de envío del paquete finalmente se imprime la diferencia.

Pseudo código para Sensor que trabaja como Cliente:

Inicio del modulo
- Creación del hilo y función de ejecución
- La función de ejecución asigna la dirección IP al cliente(Sensor).
- Crear la conexión conn. Se crea la liga entre la conexión conn, dirección IP y el puerto 7.
- Se crea la conexión al servidor(Control) a través del puerto 7.
- Inicia ciclo desde 0 hasta n-1 paquetes a enviar
  - Se asigna tiempo de inicio a ti
  - Se envía el paquete
  - Se asigna tiempo de inicio a tf
  - Se calcula el tiempo transcurrido entre el paquete enviado y el tiempo inicial Delta=tf-ti
  - Se imprime la diferencia de tiempos delta
- Fin de ciclo
- Se cierra y borra conexión. Se remueve módulo

Pseudo código para Control el cual trabaja como Servidor:

Al igual que el módulo de Sensor en este módulo se mide el tiempo necesario para enviar uno o n paquetes desde Sensor a Control, en primer lugar se crea el hilo (con su función de ejecución), la función de ejecución liga el puerto y la dirección IP después se crea la conexión en donde Control (servidor) escucha las peticiones de los clientes (Sensor) y acepta la petición creando una nueva conexión por donde se recibe el paquete a través de un buffer en este momento se registra el tiempo inicial ti de recepción del paquete posteriormente se imprime el contenido del buffer y se borra, en ese instante registramos el tiempo final tf de envío del paquete finalmente se imprime la diferencia.

Inicio del modulo
Creación del hilo y función de ejecución
La función de ejecución asigna la dirección IP a Servidor(Control).
Crear la conexión conn.
Se crea la liga entre la conexión conn, dirección IP y el puerto 7.
El servidor escucha por medio de la conexión conn.
Se acepta cualquier conexión y su valor se asigna a newconn
Inicia ciclo desde 0 hasta n-1 paquetes a enviar
Se recibe la información en un buffer buf.
Asigna tiempo de inicio a ti
Se imprime el contenido del buffer y Se borra el buffer.
Asigna tiempo de fin de envío de respuesta a tf
Se calcula el tiempo transcurrido entre el paquete enviado de respuesta y el tiempo inicial de recibido Delta=tf-ti
Se imprime la diferencia de tiempos delta
Fin de ciclo
Se cierra la nueva conexión newconn, Se remueve módulo

En las pruebas realizadas se observa lo siguiente (tabla 5.3): el nodo Sensor puede enviar solamente desde 150 hasta 2200 paquetes sin bloquear el tráfico de paquetes de los cuales el nodo Control solo puede recibir desde 50% hasta un poco más de 90% además la cantidad de paquetes enviados por Sensor no es la misma recibida por Control.

Pruebas de envío y recepción de paquetes desde SENSOR a CONTROL				
Sensor	Control	Porcentaje recibido	Control	Porcentaje recibido
Paquetes enviados	Paquetes recibidos	%	Paquetes perdidos	%
150	75	50%	75	50
150	100	66%	50	34
550	400	72%	150	28
650	500	76%	150	24
1200	1000	83%	200	17
2200	2000	90%	200	10

Tabla 5.3.- Porcentaje de paquetes recibidos desde Sensor a Control.

Si en Control se trata de recibir la misma cantidad de paquetes que envía Sensor el trafico de paquetes causa un congestionamiento que bloquea los módulos tanto de Sensor como de Control, solamente recibiendo menos paquetes que los que Sensor envía se evita que el módulo de Control no se bloquee. La función de regresar un paquete de respuesta como acknoledge no funciona aunque tenga la conexión activa, esto me dificulta la forma de calcular el tiempo que tarda un paquete en llegar desde Sensor a Controlador puesto que Sensor no conoce el tiempo de llegada del paquete enviado ni tampoco puede recibir un paquete del Controlador para realizar un promedio del tiempo de envío y el de recepción.

Los módulos que se utilizaron fueron para Sensor (tcpclient2kok.c) y Control (tcpserver2kok.c), además se registro el tiempo de inicio y tiempo final del primer envió y del tiempo final de recepción de los 2000 paquetes (tabla 5.4).

Sensor envío de paquetes Tiempo en nanosegundos.			
Envío	Tiempo inicial	Tiempo final de envío	Diferencia
1 paquete	139,202,560	139,464,094	261,534
2000 paquetes	139,202,560	1 310,359,008	1,171,156,448

Tabla 5.4.- Tiempo utilizado por Sensor en el envío de n paquetes a Control.

Como conclusión de esta etapa es la siguiente es necesario realizar varios ajustes que Controlen el flujo de los paquetes enviados ya que en esta implementación al establecer en el módulo de Sensor que el número de paquetes enviados sea igual que el numero de paquetes recibidos, en el módulo de Control sucede una desconexión, esto aparece cuando el módulo de Sensor termina de enviar los n paquetes inmediatamente cierra la conexión y el Controlador no recibe el total de paquetes, además es necesario conocer el tiempo que tarda cada paquete en llegar desde Sensor a Controlador.

ii) Calcular el tiempo que requiere un paquete en trasladarse desde cliente 1 hasta Servidor 1 utilizando periodicidad en hilos.

Se manejará la periodicidad de los envíos de paquetes desde Sensor hacia Control para que no exista pérdida de paquetes, además si medimos el tiempo que tarda un paquete en llegar de un nodo a otro, podremos establecer una periodicidad adecuada y así eliminar el cuello de botella que producen los paquetes enviados. En cada uno de los nodos contaremos con 3 tareas con sus características de periodicidad, prioridad, tiempo de ejecución y el planificador FIFO o Roun Robin.

Calcularemos el tiempo que requiere un paquete en llegar desde Sensor hasta Control, estableciendo una periodicidad X en la tarea 1 de Sensor que es la que enviará el paquete, sabremos que se enviará un paquete cada periodo X, después de enviar el paquete se realiza una espera de Y que me asegure que el paquete llegará a Control.

Por su parte Control recibe el primer paquete en un tiempo Q1, el siguiente paquete lo recibirá en un tiempo final de paquete =Q1+X+tiempo que tarda el paquete.
Como el tiempo Q yo lo puedo conocer en Control y la periodicidad de Sensor también entonces el tiempo que tarda el paquete= tiempo final –Q-X con esta fórmula se puede calcular el tiempo que tarda cada paquete excepto el primer paquete.

Tiempo de traslado de un paquete desde un nodo hacia otro.			
	Sensor		**Control**
Paquete	Tarea 1 Periodicidad de envío	Tiempo de traslado del paquete	Tiempo de recepción
1	X	¿?	Q1
2	X	Q2-Q1-X	Q2 = Q1+X+tiempo de traslado del paquete
N	X	$Qn-Q_{n-1}-X$	$Qn = Q_{n-1}+X+$tiempo de traslado del paquete

Tabla 5.4.1.- Tiempo de traslado de un paquete desde un nodo hacia otro. (Se muestran los archivos en el anexo como: Sensor: Tcpcli26abrrunok.c, Control: Tcpser26abrrunok.c)

Se observa que para 120 paquetes enviados por Sensor y recibidos en Control (Paq5 en Control):

- El tiempo promedio de envío es de 76 366 nanosegundos
- El tiempo mayor de envío es 182 944 nanosegundos

Nota: Siempre se imprime el tiempo que tarda en llegar el paquete.

Para 1000 paquetes enviados por Sensor y recibidos en Control (Paq6 en Control):

- El tiempo promedio de envío es de 2 071 118 nanosegundos
- El tiempo mayor de envío es 1 364 514 080 nanosegundos
- El tiempo inicial 705 634 432
- El tiempo final 1 675 839 424
- Tiempo transcurrido de enviar los 1000 paquetes 970.204492 milisegundos.

Conclusiones el promedio de envío se incrementa notoriamente así como el tiempo mayor de envío aunque el tiempo que tarda en llegar el paquete se visualizan similares entre 1000 a 120 paquetes. Loque tendremos que buscar es el valor adecuado de X (periodicidad de envío) en la tarea 1 y del tipo de planificador utilizado en Sensor.

Ejemplos 1 con planificador Round Robin y una tarea tanto en Sensor como en Control

Ejecutando los módulos de Sensor y Control se tienen los siguientes resultados:

PC	Objeto Activo	Periodicidad milisegundos	Planificador	Paquetes enviados	Paquetes recibidos	Tiempo transcurrido milisegundos	Promedio de datos/seg
Sensor	Hilo 1	0.1	RR	73,246		35,045.390	2147.43
Control	Hilo 1	0.1	RR		73,037	36,926.618	1977.89

Tabla 5.5.- Tiempos y promedios utilizado por Sensor en el envío de n (73000 aproximadamente) paquetes a Control.

En la tabla 5.5 el nodo Sensor tiene 1 hilo que envía el paquete con una periodicidad de 100,000 nanosegundos (0.1 milisegundos) y planificador RR, envía 73,246 datos en 35,045.390048 milisegundos con un promedio de 2147.43 paquetes/segundo por parte del nodo Controlador tiene 1 hilo que recibe el paquete cada 100,000 nanosegundos y planificador RR, recibe 73037 datos en 36,926.618304 milisegundos con promedio de 1977.89 paquetes/segundo.

Se realiza otra prueba arrojando los siguientes resultados:

PC	Objeto Activo	Periodicidad Milisegundos	Planificador	Paquetes enviados	Paquetes recibidos	Tiempo transcurrido Milisegundos	Promedio de datos/seg
Sensor	Thread 1	0.1	RR	21,702		31,411.597920	135.53
Control	Thread 1	0.1	RR		21,525	42,765.265184	233.071

Tabla 5.6.- Tiempos y promedios utilizado por Sensor en el envío de n (21000 aproximadamente) paquetes a Control.

Tabla 5.6 para la prueba de 35 segundos se observa que existe una sincronía en el envío y recepción de los paquetes en la segunda prueba los datos enviados y recibidos decrementan en un 60% pero la sincronía se mantiene buena puesto que el número de paquetes enviados y similar al número de paquetes recibidos (el archivo de prueba esta en ejerc2.txt en el anexo y los módulos en tcpc7abr.c y tcps7abr.c) además se esta utilizando un planificador Round Robin en donde solo hay una función o tarea a realizar que es el envío y recepción de datos.

Es necesario utilizar un planificador que maneje más de 1 tarea, dentro de cada nodo se crean 3 hilos, cada función realizará una tarea diferente una para cada hilo, los hilos utilizados son los siguientes:

Static void * Start routine1(void *arg),
Static void * Start routine2(void *arg) y
Static void * Start routine3(void *arg)

Se aplica la técnica de mutual exclusión al inicio de los 3 threads con el fin de que el hilo utilice al procesador de forma exclusiva (MUTEX) para terminar su tarea. Se crean 3 threads (esto en init_module):

```
Ret1=pthread_create((&threads[1], &attr, start_routine1, (void *) 1);
Ret2=pthread_create((&threads[2], &attr, start_routine2, (void *) 2);
Ret3=pthread_create((&threads[3], &attr, start_routine3, (void *) 3);
```

c) Envío y recepción de paquetes utilizando la técnica de mutual exclusión para sincronizar envíos.

Objetivos

i) Sincronizar el envío y recepción de paquetes utilizando un planificador de tareas y técnica de mutual exclusión.

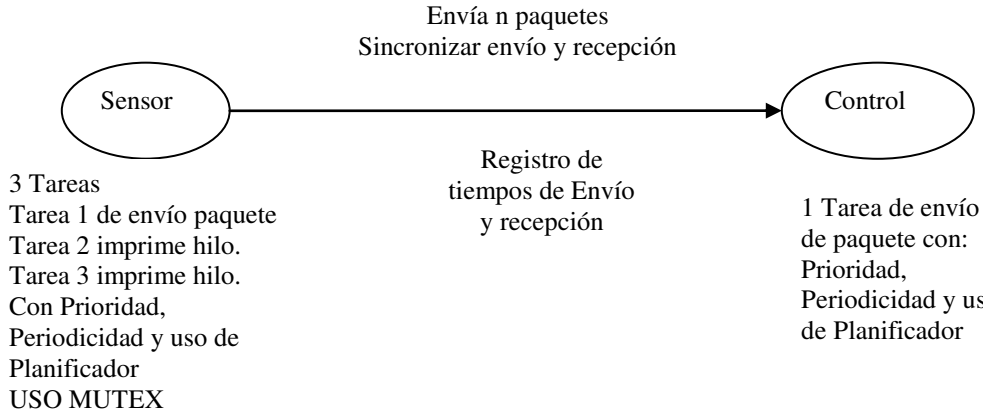


Figura 5.5.- Envío de n paquetes desde Sensor a control utilizando periodicidad en las tareas (3), contando con un planificador y MUTEX

En este paso dejaremos funcionando los tres threads con la implementación de MUTEX paquetes lo cual me asegura que el procesador es exclusivo de mi tarea hasta que termine dicho envío y así poder medir el tiempo que tarda un paquete en llegar desde Sensor a Control, en primer lugar tendremos que sincronizar el envío y recepción de los paquetes y en segundo lugar tratar de enviar y recibir el mayor numero de paquetes (figura 5.5).

ii) Incrementar la cantidad de paquetes enviados entre el nodo Sensor y el nodo Control, esto lo podemos lograr realizando cambios en la periodicidad del primer hilo que es el encargado de enviar los datos.

Nuestra hipótesis es que podemos incrementar el número de paquetes controlando la periodicidad de la tarea 1(hilo1) encargada de enviar los paquetes sin causar el problema de cuello de botella además de poder sincronizar el envío y recepción de paquetes con MUTEX.

i) Sincronizar el envío y recepción de paquetes utilizando un planificador de tareas y técnica de mutual exclusión.

El pseudo código de Sensor es el siguiente:

En este módulo registramos el tiempo inicial en la variable start_nanosec además se integra la creación de hilos (3) cada una realiza una función específica, la función 1 esta dedicada solo al envío de paquetes, la tarea tiene prioridad, periodicidad y un planificador de tareas FIFO, el envío de los paquetes se hará cada 100 milisegundos (0.1 segundos), se ingresa a un ciclo infinito de envío de paquetes donde la tarea espera hasta su siguiente turno cuando es el turno de la tarea ejecutar su acción (tiempo de procesador) el hilo aplica exclusión mutua (el procesador es exclusivo de este pequeño proceso hasta que termine su tarea). Dentro de esta exclusión mutua existen varias acciones enviar el paquete a Control, incrementar en 1 el numero de paquetes enviados y realizar una espera de 0.001 milisegundos en este momento el hilo libera la exclusión mutua en este instante se registra el tiempo en la variable now, mide el tiempo transcurrido desde el inicio del módulo y el envío del paquete finalmente se imprime este tiempo transcurrido y el número de paquetes enviados.

```
Inicio el modulo

    Tiempo se asigna a start_nanosec;
    Creo hilo 1 y su función 1;     Creo hilo 2 y su función 2;     Creo hilo 3 y su función 3
Función 1
    Realiza conexión,   Asigna prioridad 1
    Establece planificación FIFO con prioridad 1 al hilo.
    Establece al hilo, inicio inmediato, periodicidad cada 100,000,000 nanoseg (0.1 segundo)
    Ciclo infinito
        Espera hasta el siguiente turno del hilo (pthread_wait_np( ) )
        Realiza exclusión mutua
            Envía el dato a Control
            Numero de paquetes lo incremento en 1
            Espera por 1000 nanosegundos (0.000001 segundos)
        Fin de exclusión mutua
        Asigna tiempo a la variable now
        Calcula el tiempo transcurrido desde el inicio del módulo y el envío
        Elapsed_time=now-start_nanosec;
        Imprime tiempo requerido para enviar el paquete
        Imprime número e paquetes enviados
Borra los 3 threads
Función 2
    Imprime número de hilo.
Función 2
    Imprime número de hilo.
Borra modulo
```

En este módulo de Sensor en el cual envía datos (paquetes de 80 char) a Control se implementa una periodicidad de 0.1 seg para el envío de cada dato además incluyo una espera de 1 microsegundo para que el dato (paquete) llegue a su destino, imprimo el tiempo requerido de envío de cada paquete y el número de paquetes enviados.

Pseudo código para Control el cual trabaja como Servidor:

Para el módulo de Control registramos el tiempo inicial en la variable start_nanosec, se crea el hilo y su función, control escucha cualquier conexión (Sensor) inicia un ciclo infinito para aceptar los paquetes en donde se recibe el paquete se incrementa el número de paquetes recibidos en este momento se registra el tiempo en la variable now y se mide el tiempo transcurrido desde el inicio del módulo hasta la llegada de cada paquete.

Inicio del modulo
- Se asigna tiempo a Start_nanosec
- Creación del hilo y función de ejecución
- La función de ejecución asigna la dirección IP al servidor (Control).
- Crear la conexión conn.
- Se crea la liga entre la conexión conn, dirección IP y el puerto 7.
- El servidor escucha por medio de la conexión conn.
- Se acepta cualquier conexión y su valor se asigna a la nueva conexión newconn
- Si la nueva conexión es diferente nulo continúo la recepción de paquetes
- Inicia ciclo infinito
  - Recibo el paquete en el buffer y lo copio en data
  - Imprimo el paquete (data)
  - Incremento los datos recibidos en 1
  - Borro el buffer
  - Se asigna tiempo a now
  - Calcula tiempo transcurrido desde la creación del módulo hasta la llegada de cada dato elapsed_time= now-start_nanosec.
  - Se imprime le tiempo transcurrido de recepción de cada paquete o dato

Se remueve módulo

En este módulo se recibe el paquete, incremento la variable de número de paquetes e imprimo el tiempo transcurrido en que se recibe cada paquete.

Para lograr un accionar más rápido en la ejecución de los módulos se incluyen en un shell llamado “pone” los comandos que utiliza cada nodo antes de insertar el módulo, estos comandos son:

Rtlinux start
Insmod rtl_malloc.o
Insmod rtl_3c905drv.o
Insmod rtl_lwip_stack.o // los comandos se tienen en (./pone)
Y un shell llamado “quita” que como lo dice su nombre elimina todos los módulos insertados esto me permite realizar las pruebas de forma mas eficiente y rápida.

Los resultados obtenidos de varias pruebas en el envío de datos desde Sensor (utilizando tcpcli18abr.c) a Control (utilizando tcpser18abr.c) son los siguientes:

Prueba 1

Sensor							
				3 threads		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad (milisegundos) T1: 100 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	56.902918240seg	480	1 paquete cada 0.11854 seg

Tabla 5.7.- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 56 segundos utilizando mutex.

Archivo en que se guarda la corrida: p1 con (dmesg |tail -2000 >p2).
En la tabla 5.7 se muestra la cantidad de hilos utilizados en el nodo Sensor utilizando un planificador FIFO y la técnica de mutual exclusión, se envían 480 paquetes de 80 caracteres en un tiempo aproximado de 1 minuto.

Control							
				1 threads		Proceso Paquetes recibidos	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos.	Datos [80] chars	0.000000	56.902918240 seg	479	1 paquete cada 0.11854seg

Tabla 5.8.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 56 segundos.

Archivo en que se guarda la corrida: pi2.
En la tabla 5.8 se muestra la cantidad de hilos utilizados en el nodo Control sin planificador, se reciben 479 paquetes de 80 caracteres en cerca de 1 minuto.

Prueba 2

Sensor							
				3 threads		Proceso Paquetes enviados	
Modulo	Threads	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 100 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	54.61368246seg	457	1 paquete cada 0.11950seg

Tabla 5.9- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 54 segundos utilizando mutex y un planificador (FIFO).

Archivo en que se guarda la corrida: p2

Control							
				1 threads		Proceso Paquetes recibidos	
Modulo	Threads	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	57.347749312seg	453	1 paquete cada 0.12659seg

Tabla 5.10 - Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 57 segundos.

Archivo en que se guarda la corrida: pi2

En las tablas 5.9 y 5.10 se observa que la sincronización de envío y recepción de paquetes es aceptable pues el número de paquetes enviados por Sensor es de 480 y el de recibidos por Control es de 479 aunque la cantidad de paquetes enviados es mínima 480 en 56 segundos, realizando 2 pruebas el comportamiento es similar.

Prueba 3

Ahora realizando pruebas con tiempos diferentes (180 seg) tenemos el siguiente comportamiento en Sensor, utilizando 3 hilos la tarea 1 (T1 es la encargada de enviar el paquete a Sensor), el modulo utiliza 3 hilos y un planificador FIFO además de la técnica MUTEX (Mutual Exclusión), se envían 1686 paquetes en aproximadamente 3 minutos (Tabla 5.11).

Sensor							
				3 hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 100 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	177.532438912seg	1686	1 paquete cada 0.105316seg

Tabla 5.11- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 177 segundos utilizando mutex y un planificador (FIFO).

Archivo en que se guarda la corrida: p3

Control							
				1 hilos		Proceso Paquetes recibidos	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	179.686582112seg	1684	1 paquete cada 0.106702seg

Tabla 5.12.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 179 segundos.

Archivo en que se guarda la corrida: pi3, se muestra la cantidad de hilos utilizados en el nodo Control sin planificador, se reciben 1684 paquetes de 80 caracteres en aproximadamente 3 minutos (tabla 5.12).

En una tercera prueba se ha incrementado el tiempo de ejecución a 180 segundos la sincronización en el envío y recibo de datos es correcta pero la cantidad de datos sigue siendo muy poca.

Notas: Se almacenan los archivos que funcionaron con periodicidad de 100 000 000 ns (100 ms) como tcpcli18abr.c para Sensor, tcpser18abr.c para Control, los resultados de su corrida están en p1, p2, p3, p4, pi1, pi2, pi3, pi4 (en archivos de texto).

Prueba 4

En Sensor con Periodicidad de 100 000 000 ns (100 ms) y 30 segundos en la corrida, se utilizan 3 hilos en la tarea 1 (T1 es la encargada de enviar el paquete a Sensor), el modulo utiliza 3 hilos y un planificador FIFO además de la técnica MUTEX (Mutual Exclusión), se envían 230 paquetes en aproximadamente 30 segundos (Tabla 5.13).

Sensor							
				3 hilos		Proceso Paquetes enviados	
Modulo	Threads	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 100 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	31.942472928seg	230	1 paquete cada 0.138880317seg

Tabla 5.13- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 30 segundos utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 100 milisegundos.

Archivo en que se guarda la corrida: s1,

Control							
				1 hilos		Proceso Paquetes recibidos	
Modulo	Threads	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	35.027647744seg	228	1 paquete cada 0.15363seg

Tabla 5.14.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 30 segundos.

Archivo en que se guarda la corrida: su1, se muestra la cantidad de hilos utilizados en el nodo Control sin planificador, se reciben 228 paquetes de 80 caracteres en aproximadamente 30 segundos (tabla 5.14).

Se observa que en las pruebas de 180 y 30 segundos la sincronía no se pierde tampoco con 177 segundos en el Sensor ya que envía 1686 paquetes de los cuales Control ha recibido 1684.

ii) Incrementar la cantidad de paquetes enviados entre el nodo Sensor y el nodo Control, esto lo podemos lograr realizando cambios en la periodicidad del primer hilo que es el encargado de enviar los datos.

En esta etapa nuestro problema es incrementar la cantidad de datos (paquetes) enviados y recibidos desde Sensor a Control, nuestro objetivo es claro "tratar de elevar el número de paquetes enviados", esto lo podemos lograr realizando cambios en la periodicidad del primer hilo que es el encargado de enviar los datos. En las pruebas 1, 2 y 3 de los ejemplos anteriores la periodicidad de la tarea es de 0.1 segundos (100,000,000 nanosegundos). Realizando dichos cambios en la periodicidad los resultados fueron los siguientes:

Prueba 1

Con una Periodicidad de 10, 000,000 nanosegundos (0.01 milisegundos) en Sensor y duración de 50 segundos aproximadamente, T2: 4 000,000 (4 milisegundos), T3: 3 000,000 (3 milisegundos) estamos decrementando la periodicidad para aumentar el número de datos enviados y recibidos. Los resultados se muestran en la tabla 5.15.

Para 53 segundos

Sensor							
				3 Hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 10 T2: 4 T3: 3	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	53.02910780seg	630	1 paquete cada 0.084173seg

Tabla 5.15- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 50 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 10 milisegundos.

Archivo en que se guarda la corrida: s2

Control							
				1 Hilos		Proceso Paquetes recibidos	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	56.010444992seg	621	1 paquete cada 0.090193seg

Tabla 5.16.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 56 segundos.

Archivo en que se guarda la corrida: su2, la recepción de los paquetes no se da con sincronía y se pierden algunos de ellos, solo se reciben 621 de los 630 que se enviaron además no se incremento el número de paquetes enviados (tabla 5.16).

Prueba 2

Con una Periodicidad de 20 000,000 ns (20 ms) en Sensor y duración de 167 segundos, T2: 400 000 000 ns (400 ms), T3: 300 000 000 ns (300 ms).

Sensor							
				3 Hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 20 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	167.358428512seg	1165	1 paquete cada 0.143655seg

Tabla 5.17- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 160 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 20 milisegundos.

Archivo en que se guarda la corrida: s3

Control							
				1 Hilos		Proceso	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	170.152 548096 seg	1161	1 paquete cada 0.146556 888 seg

Tabla 5.18- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 170 segundos aproximadamente.

Archivo en que se guarda la corrida: su3.

En las tablas 5.17 para el nodo Sensor y 5.18 para el nodo Control el número de paquetes se incrementa teniendo la periodicidad de la tarea 1 en 20 ms con planificador FIFO y técnica MUTEX, el promedio de recepción es de 1 paquete cada 0.14 segundos.

Prueba 3

Periodicidad de 50 ms en T1 en Sensor y duración de 181 segundos, T2: 400 ms, T3: 300 ms

Sensor							
				3 Hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 50 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	181.8032420 seg	3458	1 paquete cada 0.052574 seg

Tabla 5.19- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 180 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 50 milisegundos.

Archivo en que se guarda la corrida: s4

Control							
				1 Hilos		Proceso	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	184.34063264 seg	3457	1 paquete cada 0.053323874 seg

Tabla 5.20.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 180 segundos aproximadamente.

Archivo en que se guarda la corrida: su4, en las tablas 5.19 para el nodo Sensor y 5.20 para el nodo Control el número de paquetes se incrementa teniendo la periodicidad de la tarea 1 en 50 ms con planificador FIFO y técnica MUTEX, el promedio de recepción es de 1 paquete cada 0.053 segundos.

Prueba 4

Con Periodicidad de 50 ms en T1 en Sensor y duración de 326 segundos, T2: 400 ms, T3: 300 ms

Sensor							
				3 Hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 50 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	323.572567680 seg	6274	1 paquete cada 0.051573 seg

Tabla 5.21- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 320 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 50 milisegundos, la cantidad de paquetes enviados es de 6274.

Archivo en que se guarda la corrida: s5

Control							
				1 Hilos		Proceso	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	326.928541056seg	6271	1 paquete cada 0.052133seg

Tabla 5.22.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 326 segundos aproximadamente, recibiendo 6271 paquetes.

Archivo en que se guarda la corrida: su5, en las tablas 5.21 para el nodo Sensor y 5.22 para el nodo Control el número de paquetes se incrementa teniendo la periodicidad de la tarea 1 en 50 ms con planificador FIFO y técnica MUTEX, el promedio de recepción es de 1 paquete cada 0.0521 con una duración de 326 segundos en la corrida.

Prueba 5

Con Periodicidad de 45 ms en T1 en Sensor y duración de 316 segundos, T2: 400 ms, T3: 300 ms

Sensor							
				3 Hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad T1: 45 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	316.433144064seg	6834	1 paquete cada 0.046302seg

Tabla 5.23- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 316 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 45 milisegundos, la cantidad de paquetes enviados es de 6834.

Archivo en que se guarda la corrida: s6

Control							
				1 Hilos		Proceso	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	317.582864352 seg	6831	1 paquete cada 0.046491416 seg

Tabla 5.24.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 317 segundos aproximadamente, recibiendo 6831 paquetes.

Archivo en que se guarda la corrida: su6.

En las tablas 5.23 para el nodo Sensor y 5.24 para el nodo Control el número de paquetes se incrementa teniendo la periodicidad de la tarea 1 en 45 ms con planificador FIFO y técnica MUTEX, el promedio de recepción es de 1 paquete cada 0.046 con una duración de 317 segundos en la corrida.

Prueba 6

Con Periodicidad de 40 ms en T1 en Sensor y duración de 296 segundos, T2: 400 ms, T3: 300 ms

Hay pérdida de datos, se envían 2529 strings de 109 chars

Sensor							
				3 Hilos		Proceso Paquetes enviados	
Modulo	Hilos	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad T1: 40 ms T2: 400 ms T3: 300 ms	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [109] chars	0.000000	296.678598496seg	2529	1 paquete cada 0.117310seg

Tabla 5.25- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 296 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 40 milisegundos, la cantidad de paquetes enviados es de 2529.

Archivo en que se guarda la corrida: s7

Control							
				1 Hilos		Proceso	
Modulo	Hilos	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [109] chars	0.000000	298.506691136seg	2374	1 paquete cada 0.125739971seg

Tabla 5.26.- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 298 segundos aproximadamente, recibiendo 2374 paquetes.

Archivo en que se guarda la corrida: su7.

En las tablas 5.25 para el nodo Sensor y 5.26 para el nodo Control el número de paquetes se decrementa teniendo la periodicidad de la tarea 1 en 40 ms con planificador FIFO y técnica MUTEX, el promedio de recepción es de 1 paquete cada 0.1257 con una duración de 298 segundos en la corrida.

La prueba con Periodicidad de 45 ms en Sensor y duración de 316 segundos, T2: 400 ms, T3: 300 ms en Sensor fue la que envío la mayor cantidad de paquetes con 6834.

d) Integración del nodo Actuador en el sistema distribuido

Objetivo
Integrar el nodo Actuador en nuestro sistema distribuido.

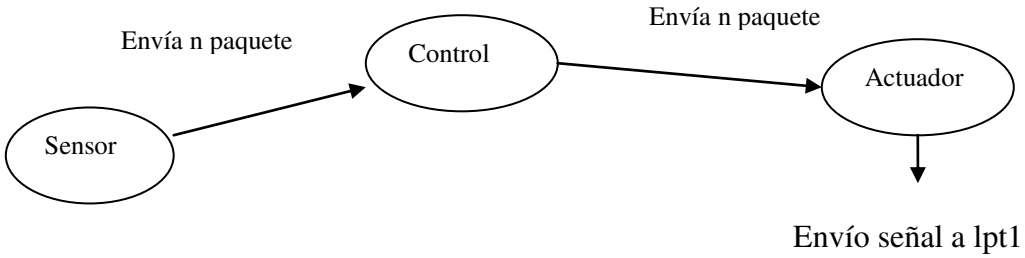


Figura 5.6.- Envío de n paquetes desde Sensor a Control posteriormente desde control a Actuador utilizando periodicidad en las tareas (3), contando con un planificador y MUTEX

Aunque nuestras pruebas aún no envían y reciben más de 10,000 paqutes entre nodos (cliente 1 – servidor 1) en un tiempo de 300 segundos; en esta etapa nos enfocaremos al objetivo de realizar el envío de datos desde Sensor a Control y posteriormente desde Control a Actuador (figura 5.6), Sensor trabajará como cliente 1, Control lo hará como servidor 1 y posteriormente como cliente 2 para conectarse a Actuador que trabajará como Servidor 2 además en Actuador se envía la señal al puerto paralelo posteriormente nos dedicaremos al problema en la cantidad de paquetes enviados en el inciso i.

El pseudo código de Sensor es el siguiente:
En este módulo de Sensor el ciclo de envío será de 1000 paquetes nuestra periodicidad del hilo de envío es de 45 milisegundos utilizando mutual exclusión, al final se imprimen el tiempo requerido de envío de cada paquete y el número de paquete que se envía.

```
Inicio el modulo
    Tiempo se asigna a start_nanosec;
    Creo hilo 1 y su función 1
    Creo hilo 2 y su función 2
    Creo hilo 3 y su función 3
Función 1
    Realiza conexión
    Asigna prioridad 1
    Establece planificación FIFO con prioridad 1 al hilo.
    Establece al hilo, inicio inmediato, periodicidad cada 45,000,000 nanoseg (0.045 segundos)
    Ciclo 1000
        Espera hasta el siguiente turno del hilo (pthread_wait_np( ) )
        Realiza exclusión mutua MUTEX
            Envía el dato a Control
            Numero de paquetes lo incremento en 1
            Espera por 1000 nanosegundos (0.000001 segundos)
        Fin de exclusión mutua MUTEX
        Asigna tiempo a la variable now
        Calcula el tiempo transcurrido desde el inicio del módulo y el envío
```

Elapsed_time=now-start_nanosec;
Imprime tiempo requerido para enviar el paquete
Imprime número e paquetes enviados
Borra conexión
Función 2
Imprime número de hilo cada 400 000 000 nanoseg.
Función 3
Imprime número de hilo cada 300 000 000 nanoseg
Borra los 3 threads

Pseudo código para Control el cual trabaja como Servidor y Cliente:

En este módulo de Control (como servidor) al aceptar la conexión del cliente ingresará en un ciclo de recibido será de 1000 paquetes al recibir cada paquete se registra el tiempo en la variable now con la cual mediremos el tiempo requerido de transporte en la red de cada paquete después de recibir los 1000 paquetes Control ahora cierra la conexión y abre otra realizando funciones de cliente conectándose al servidor (Actuador) posteriormente ingresa a un ciclo donde envía los paquetes al Actuador por último cierra la conexión.

Inicio del modulo
Se asigna tiempo a Start_nanosec
Creación del hilo y función de ejecución
La función de ejecución asigna la dirección IP al servidor (Control).
Crear la conexión conn.
El servidor escucha por medio de la conexión conn.
Se acepta cualquier conexión y su valor se asigna a la nueva conexión newconn
Si la nueva conexión es diferente nulo continúo la recepción de paquetes
Inicia ciclo 1000
Recibo el paquete en el buffer y lo copio en data
Imprimo el paquete (data)
Incremento los datos recibidos en 1
Borro el buffer
Se asigna tiempo a now
Calcula tiempo transcurrido desde la creación del módulo hasta la llegada de cada dato
elapsed_time= now-start_nanosec.
Se imprime le tiempo transcurrido de recepción de cada paquete o dato
Cierra conexión
Asigna la dirección IP al Cliente (Control).
Crear la conexión conn.
Se conecta al Servidor (Actuador)
Inicia ciclo 100
Envío el paquete
Espero 1000 nanosegundos
Incremento la variable del ciclo en 1
Borro conexión
Se remueve módulo

Pseudo código para Actuador el cual trabaja como Servidor:

Aquí el nodo Actuador actúa como cliente, aceptando la conexión de control y al recibir el primer paquete envía las señales al puerto paralelo.

Inicio del modulo
- Se envía cero al puerto paralelo
- Creación del hilo y función de ejecución
- La función de ejecución asigna la dirección IP al servidor (Actuador).
- Crear la conexión conn.
- El servidor escucha por medio de la conexión conn.
- Se acepta cualquier conexión y su valor se asigna a la nueva conexión newconn
- Si la nueva conexión es diferente nulo continúo la recepción de paquetes
- Inicia ciclo infinito
  - Incremento los datos recibidos en 1
  - Envío 0 al puerto paralelo
  - Si el contador es 1
    - Envío señales de 0 a 7 al puerto paralelo
  - Recibo el paquete en el buffer y lo copio en data
  - Imprimo el paquete (data)
  - Borro el buffer
  - Se imprime le tiempo transcurrido de recepción de cada paquete o dato
- Borro conexión

Se remueve módulo

Los resultados de las pruebas con los tres nodos son los siguientes:

Prueba 1

Sensor							
				3 threads		Proceso Paquetes enviados	
Modulo	Threads	Técnica	Envío	Tiempo Inicial	Tiempo final	total	Promedio
Insmod tcpclient.o	3 Periodicidad milisegundos T1: 45 T2: 400 T3: 300	Mutual Exclusión en Envío de datos, FIFO como planificador	Datos [80] chars	0.000000	53.903164128 seg	1000	1 paquete cada 0.053903 seg

Tabla 5.27- Tiempos y promedios utilizado por Sensor en el envío de n paquetes a Control por un tiempo de 53 segundos aproximadamente utilizando mutex, planificador (FIFO) y periodicidad en la tarea 1 de 45 milisegundos, la cantidad de paquetes enviados es de 1000.

Archivo en que se guarda la corrida: r4

Control							
				1 threads		Proceso Paquetes recibidos	
Modulo	Threads	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	63.897229760 seg	1000	1 paquete cada 0.0638972 seg

Tabla 5.28- Tiempos y promedios utilizado por Control en la recepción de n paquetes por un tiempo de 63 segundos aproximadamente, recibiendo 63 paquetes.

Archivo en que se guarda la corrida: ri4

Actuador							
				1 threads		Proceso	
Modulo	Threads	Técnica	Recepción	Tiempo Inicial	Tiempo final	total	Promedio
Tcpserver.o	1	Acepta datos	Datos [80] chars	0.000000	22.680396832 seg		

Tabla 5.29- Tiempos y promedios utilizado por Actuador en la recepción de n paquetes por un tiempo de 22 segundos aproximadamente.

Archivo en que se guarda la corrida: rit4.

Se observo que el Actuador acepta datos y envía señales al Puerto Paralelo, además la sincronización en el envío y recepción de paquetes (1000) entre Sensor y Control se realiza correctamente por aproximadamente 60 segundos (Tablas 5.27, 5.28 y 5.29). Se utilizaron los programas tcpcli21abr.c, tcpser21abrnoche.c en Actuador el archivo es tcpser22abr.c la cantidad de datos sigue siendo mínima.

e) Cambios en cantidad de paquetes de envío desde Sensor a Control

Objetivo
Enviar 100 000 paquetes utilizando una tarea de envío en el nodo Sensor y una tarea de recepción en el nodo Control.

En esta etapa el objetivo es lograr enviar 100 000 paquetes de 80 char en un tiempo de 300 segundos aproximadamente se utiliza threads con MUTEX y una espera en Sensor después de enviar el paquete que me asegure que el siguiente paquete no causará tráfico en la red.

Se realizan pruebas dando una periodicidad a la tarea de envío mayor que el tiempo mayor ó promedio que se tarda en llegar cada paquete y se le da un tiempo de espera que me asegure que no enviaré otro paquete, de esta forma me asegurare que el paquete en cuestión ya fue recibido por Control. El archivo utilizado en Sensor es tcpcli20abr.c.

Para la sincronización en el envío y recepción de paquetes entre los nodos (Sensor, Control y Actuador) se utilizan threads en cada uno de los nodos como se muestra en las tabla 5.30.

SENSOR			
	Prioridad	Periodicidad milisegundos	Planificador
Hilo 1	1	1	FIFO
Hilo 2	2	400	FIFO
Hilo 3	3	300	FIFO
CONTROL			
	Prioridad	Periodicidad milisegundos	Planificador
Hilo 1	1	0.01	FIFO
Hilo 2	2	400	FIFO
Hilo 3	3	300	FIFO
ACTUADOR			
	Prioridad	Periodicidad milisegundos	Planificador
Hilo 1	1	0.01	FIFO
Hilo 2	2	400	FIFO
Hilo 3	3	300	FIFO

Tabla 5.30.- Características de cada uno de los nodos del Sistema Distribuido en tiempo real.

Las pruebas de envío y recepción de paquetes incluyendo al nodo Actuador, sus resultados fueron los siguientes:

SENSOR		
	TAREA	**Cantidad de paquetes enviados**
Hilo 1	Conexión a nodo Control Envío de paquetes de 80 caracteres a Control	1000, 5000, 10000, 15000, 20000, 25000, 30000, 60000, 100000
Hilo 2	Impresión de identificador de thread 2	
Hilo 3	Impresión de identificador de thread 3	

Tabla 5.31.- Descripción de las funciones de los hilos utilizados en Sensor y paquetes enviados.

CONTROL		
	TAREA	**Cantidad de paquetes recibidos de Sensor.** **Cantidad de paquetes enviados a Actuador**
Hilo 1	Escucha conexiones Acepta conexión Recepción de paquetes de 80 caracteres a Control Desconexión Conexión a nodo Actuador Envío de paquetes de 80 caracteres a Control	**Recibidos** 1000, 5000, 10000, 15000, 20000, 25000, 30000, 60000, 100000. **Enviados** 100, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000.
Hilo 2	Impresión de identificador de thread 2	
Hilo 3	Impresión de identificador de thread 3	

Tabla 5.32.- Descripción de las funciones de los hilos utilizados en Control y paquetes recibidos.

ACTUADOR		
	TAREA	**Cantidad de paquetes recibidos**
Thread 1	Escucha conexiones Acepta conexión Recepción de paquetes de 80 caracteres a Control Envío señal a puerto paralelo	100, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000.
Thread 2	Impresión de identificador de thread 2	
Thread 3	Impresión de identificador de thread 3	

Tabla 5.33.- Descripción de las funciones de los hilos utilizados en Actuador y paquetes recibidos.

En las tablas anteriores (5.31, 5.32 y 5.33) se muestra que el cuello de botella sigue afectando el transporte de los datos, en cuanto a la sincronización esta es la adecuada dado que el mismo número de paquetes que se envían es el que se reciben, además el puerto paralelo en Actuador funciona correctamente. Con las modificaciones realizadas en la periodicidad se alcanza un mejor desempeño en el envío de los paquetes desde Sensor a Control.

Los resultados de envío de paquetes y tiempos desde Sensor a Control son los siguientes:

Num Test	**Sensor**					**Control**		
	Espera ms	Periodicidad ms	now – Start seg	dif	File	Periodicidad ns	paquetes	File
1	1	2	1 613.564 128 448 1 607.934 943 552	5.629184 896	Fin1	2 000 000	1 000	Fin01
2	0.9	1	4175.388 341 888 4152.239 105 696	23.149.236 192	Fin2	900 000	10 000	Fin02
3	0.9	1	2 413.624 866 720 2 279.424 016 320	134.200 850 400	Fin3	10 000	50 000	Fin03
4	0.9	1	3 861.378 512 384 3 599.065 076 096	262.313 436 288	Fin4	10 000	100 000	Fin04

Tabla 5.34.- Descripción de los tiempos utilizados en el envío y recepción de paquetes.

Por las pruebas realizadas se concluye que la mejor manera de enviar 100 000 paquetes de 80 caracteres es utilizando MUTEX, Threads, periodicidad y prioridad en cada una de las tareas como se muestra en la tabla 5.34 en el test numero 4 donde se reciben 100,000 paquetes en 262 segundos.

Un ejemplo extra en esta etapa es el envío de paquetes sin imprimir nada en los threads 2 y 3 solo realizará acciones el hilo 1 que es envío del paquete además se realizan registros de tiempos de uso del hilo que será el uso del procesador.

Los archivos antes de iniciar la las estadísticas del procesador son:

- Sensor: tcpcli9may.c
- Control: tcpser9may.c
- Actuador tcp9mayact.c

Modificación 1: Las modificaciones que se le hicieron a los módulos fue de quitar de los hilos 2 y 3 la impresión.
Modificación 2: se observo que el excesivo tráfico empieza a bloquear los nodos y si aplico una espera 2 cada 25000 paquetes el flujo de paquetes continúa sin problema.

Las estadísticas del envío (comunicación) de datos son los mostrados en la tabla 5.35.

Envío/paquetes	Tiempo milisegundos	Espera1	Espera 2	T Recep	file
100,000	294.418461504	1,000,000	900,000	296043971552	May1001
100,000	226.126535936	1,000,000	950,000	227486918496	May1002
100,000	294.727780928	1,000,000	950,000	298603340800	May1003

Tabla 5.35.- Descripción de las esperas utilizadas en el envío de 100 000 paquetes.

f) Registro de retardos de las tareas de envío y recepción y envío del tiempo del cliente hacia el servidor

Objetivo

Enviar un paquete con la información de tiempo y características de la tarea 1 de Sensor hacia Control y registrar los retardos de la tarea 1 tanto de Sensor como Control.

Tiempo	Separador	C uso de procesador	P prioridad	PR periodicidad
14 chars	1 char	1 char	1 char	1 char

Tabla 5.36.- Información del paquete en Sensor, este paquete es llamado timecar.

En esta etapa es enviar en un paquete: C el tiempo de procesador, P Prioridad y PR periodicidad (tabla 5.36) desde Sensor a Control además de enviar el tiempo que maneja Sensor hasta Control, de esta forma el nodo Control conocerá la información necesaria para llevar a cabo una futura planificación,

Pseudo código para Sensor:

En este módulo después de iniciarlo se registra el tiempo en una variable y se concatena a un paquete llamado time_car, además se concatenan las variables C tiempo de procesador, P Prioridad y PR periodicidad (el paquete contiene 4 datos: tiempo, C, P, PR) después de conectarse al servidor (Control) ingresa en un ciclo de envío de paquetes donde registramos el tiempo inicial del envío y el tiempo final del ciclo de los n paquetes enviados, el primer paquete enviado es time_car que será recibido por Control para conocer el tiempo y las características de la tarea de Sensor.

```
Inicia modulo
    Crea periodicidad de la tarea cada 2 ms. (2 000 000 ns)
    Registra nowac y nowacu
    Imprime tiempo antes del ciclo nowac
    Ciclo infinito
        Pthread_wait_np
        Ret=ptrhead_mutex_trylock(mutex)
        Registra nowet y noweft tiempos de ejecucion y fin de la tarea C
        For (de 1 a 22)
            Imprime la diferencia de tiempos por 30 000 ns
        Registra noweft
        Mide el tiempo de C, diferencia de nowetf-nowet
        Imprime el tiempo inicial del ciclo
        Si es el primer paquete envía timecar e imprime el tiempo de ejecución C
        Sino envía el paquete, imprime tiempo de ejecución de C y retardo de ejecución de C. (nowet-
nowacu)
        Numpaquetes=numpaquetes+1
        Nowacu=nowacu+2 000 000
        Ret=pthread_mutex_unlock(mutex)
        Mide el tiempo actual now
        Calcula el tiempo que se ha llevado desde el inicio now-start_nanosec
        Imprime esta diferencia, Imprime el número de paquetes
    Fin de ciclo
```

Pseudo código para Control:

Para la recepción de cada uno de los paquetes en el nodo Control al aceptar la conexión de registra el tiempo en ti y tini posteriormente se ingresa a un ciclo de recepción de n paquetes, tomar en cuenta el primer paquete el cual contiene la información del tiempo, C, P, PR del nodo Sensor, al recibir el paquete se registra el tiempo a tf y se calcula delta que será el tiempo inicial antes del ciclo y el tiempo al recibir el paquete. Al recibir el primer paquete el tiempo mayor es el de este primer paquete su calculo es delta- 1ms (1ms es la espera que realiza Sensor) además se agrega ese tiempo a la variable suma, para el(los) siguiente(s) paquete se calcula el delta y se agrega a suma, si el paquete que se recibe no es el primero entonces se compara cual de los paquetes tiene el tiempo mayor al final se calcula el tiempo promedio de envío de los paquetes y se imprime el tiempo mayor de envío.

```
Inicia modulo
    Crea hilo y función
    Enlaza ip y puerto
    Escucha y acepta conexión
    Si la nueva conexión  es diferente de null
        Asigna tiempo a ti y tini
        For (i=0; 1>20000; i++)
            Se recibe buffer
                Si es el primer paquete
                    obtener el tiempo, C, P, y PR de Sensor
                    Ajustar el tiempo de Control tomando en cuenta el promedio de envío del paquete.
                    Establecer en control Periodicidad menor que en Sensor.
            No es el primer paquete
                Copiar el buffer a una variable
                Imprime la variable
            Asigna tiempo a tf, se imprime tiempo final de ciclo
            Calcula diferencia entre el tiempo inicial antes del ciclo y el tiempo final de recibo del dato
            Delta=tf-ti
            si (i<=1) el tiempo mayor =delta-1000000;
            otro, si mayor < delta-1000000 entonces mayor=delta-100000;
            suma=suma+(delta-1000000)
            si i==1000
                imprime promedio
                imprime el tiempo mayor de Envío
                imprime el tiempo final
            Asigna ti a tf
Borra hilo
```

Prueba 1

En Sensor se realiza el siguiente paquete:

Paquete		
Segundos	Nanosegundos	Características de la tarea Consumo C en centésimas de milisegundos Prioridad P, Periodicidad PR en milisegundos
7031	381 746 240	312

Tabla 5.37.- Ejemplo de la información que contendrá el paquete a enviarse.

En la tabla 5.37 la periodicidad en Sensor se establece en 2 milisegundos (2 000,000 nanosegundos) puesto que el promedio de envío es de 0.087 milisegundos (87000 nanosegundos) pero el tiempo de envío mayor es de 0.2 milisegundos (200 000 nanosegundos), el uso de procesador C de 0.03 milisegundos (30 000 nanosegundos).

La forma en que Control recibe el paquete:

Recibe	Segundos	Nanosegundos	Diferencia	
			segundos	nanoseg
381746240,312	7038	664 325 952	7	282 579 712

Tabla 5.38.- Ejemplo de la información (paquete) que recibe Control.

Para la tabla 5.38 la tarea de Sensor es convertir el tiempo que esta en long long a tiempo en caracteres para poder enviarlo en un paquete, se debe de establecer una cantidad de tiempo fija en C la cual será 3 centésimas de milisegundo (30 000 nanosegundos) cada instrucción se realiza aproximadamente en 350 nanosegundos pero en un ciclo de 10 instrucciones tarda aprox. 28000 nanoseg, Prioridad 1 y Periodicidad de 2.

Se realiza un ciclo de envío de 10 paquetes desde Sensor a Control, para comprobar que control este recibiendo el paquete con las características de la tarea, también verificar que Sensor este realizando las medidas de retardo de C.

Sensor, creación del primer paquete.

Paquete		
Segundos	Nanosegundos	Características de la tarea Consumo C en centésimas de milisegundos, 30 000 nanosegundos Prioridad P, Periodicidad PR en milisegundos
1076	955 489 696	312

Tabla 5.39.- Ejemplo de la información que contendrá el paquete a enviarse.

Tiempos de Sensor de los 10 paquetes (arch. p3007_01s)

	Tiempos		
paquete	Tarea 1 Ejecución de C	Retardos	Requerido por el hilo
1	31 968	1 987 684	8 905 058 016
2	31 392	1 966 560	8 907 024 576
3	31 296	2 012 224	8 909 029 216
4	31 424	2 013 344	8 911 028 544
5	31 232	92 768 640	9 003 797 184
6	29 216	90 870 976	9 003 879 520
7	29 152	88 952 320	9 003 956 256
8	29 216	85 401 728	9 004 709 096

Tabla 5.40.- Retardos y tiempos requeridos para el envío de 10 paquetes.

Los retardos en los 10 paquetes (tabla 5.40) no son uniformes, en los primeros paquetes es de 2 milisegundos y posteriormente se elevan los retardos hasta 85 milisegundos (tabla 5.42), el consumo de 30 000 nanosegundos por la tarea no es precisa y algunas ocasiones variará. En esta prueba el tiempo requerido para crear el primer paquete es de 17440 nanosegundos y el tiempo inicial del ciclo de los 10 paquetes es de 1 085 858 427 136 nanosegundos

Control, tiempos de Control (arch. p3007_01 en nanosegundos), la información obtenida para el primer paquete:

paq	Al recibir Paquete	Paquete Recibido De Sensor	Paquete Recibido De Sensor	Caract Sensor	Tiempo Final Recep c/paq	Inicial Modulo
1	1013 054 688 288	1076	955 489 696	312	1013 054 816 672	1001 991 947 840

Tabla 5.41.- Retardos y tiempos requeridos para la recepción de 1 paquete.

Para los siguientes 9 paquetes el tiempo final de recepción es:

Paquete	Tiempo Final de Recepción de cada paquete	Diferencia
2	1013152784896	97968224
3	1013152979968	195072
4	1013153153248	173280
5	1013153373248	220000
6	1013153880544	507296
7	1013154160704	280160
8	1015668739200	2514578496

Tabla 5.42.- Retardos y tiempos requeridos para la recepción del resto de los paquetes.

En Control se esta recibiendo el tiempo de Sensor así como las características de la tarea en el primer paquete, en Control se desglosa y se obtiene la información necesaria tiempo en segundos y nanosegundos, C uso de procesador, P prioridad y PR periodicidad de Sensor (tabla 5.41 y 5.42).

Prueba 3

Mediremos los retardos de la tarea de Sensor con ayuda de 9 paquetes que enviaran el tiempo del Sensor y sus características en el primer paquete y paquetes de 80 chars en los siguientes, los valores de Sensor son C=3, P=1, PR=2; En Control la periodicidad es de 2

C Uso de procesador	P Prioridad	PR Periodicidad
3 30 000 ns 0.03 ms	1	2 2000 000 ns 2 ms

Sensor

Paq	Tiempo		Envío	Retardo Tarea 1 milisegundos
	Segundos	nanoseg		
1	415 antes ejec 415 ejec	299091712 301104128	Tiempo y caract	1.212 416
2	415	303107008	Paq[80]	0.002880
3	415	305108128	Paq[80]	0.001120
4	415	307106752	Paq[80]	0.001376
5	415	399356480	Paq[80]	90.249 728
6	415	399963200	Paq[80]	1.393 280
7	415	399994000	Paq[80]	1.3345
8	415	400286688	Paq[80]	1.757 184
9	415	400913632	Paq[80]	1.373 056

Tabla 5.43.- Retardos y tiempos requeridos para el envío de 10 paquetes y Periodicidad de 2 ms.

Archivo p0108s01. Se envía el tiempo en Sensor el cual es 406, 395238400 y las características de la tarea 312, el retardo mayor esta en 91 milisegundos (90 249 278) el retardo menor en 0.00112 milisegundos (Tabla 5.43), por lo que no hay un tiempo establecido en el retardo de la ejecución de la tarea en Sensor.

Prueba 4

Mediremos los retardos de la tarea de Sensor con ayuda de 9 paquetes que enviaran el tiempo del Sensor y sus características en el primer paquete y paquetes de 80 chars en los siguientes, los valores de Sensor son C=3, P=1, PR=100; En Control la periodicidad es de 100, C=4, prioridad 1; así podremos calcular los retardos también en la tarea de Control.

Sensor

C Uso de procesador 3, 30 000 ns 0.03 ms	**P Prioridad** **1**	**PR Periodicidad** 100 100 000 000 ns 100 ms

Paq	**Tiempo**		**Envío**	**Retardo milisegundos**
	Segundos	**nanoseg**		
1	357	087967200	Tiempo y caract	0.100010528
2	357	187966944	Paq[80]	0.097999744
3	357	287985824	Paq[80]	0.098018880
4	357	387967008	Paq[80]	0.097981184
5	357	487966848	Paq[80]	0.097999840
6	357	587966976	Paq[80]	0.098000096
7	357	687967552	Paq[80]	0.098000576
8	357	787967648	Paq[80]	0.098000096
9	357	887966912	Paq[80]	0.097999264

Tabla 5.44.- Retardos y tiempos requeridos para el envío de 10 paquetes y Periodicidad de 100 ms.

Archivo p0108s02 y03.

En la figura 5.44 se muestra el envío de los 9 paquetes respetando la periodicidad y el promedio de retardo es de 0.087111079 ns (0.087111 ms), el uso del procesador por la tarea de Sensor esta en un promedio de 0.023 ms siendo un poco menor que el tiempo de ejecución que debería ejecutarse la tarea. El uso de procesador para la tarea 1 en Sensor es complicado utilizar los 30 000 ns de ejecución de la tarea, la tabla inferior muestra la ejecución real en nanosegundos.

C Uso de procesador 3, 30 000 ns, 0.03 ms
0.024096
0.023424
0.023072
0.023072
0.023200
0.023296
0.023360
0.023104
0.022944

Control, las características de la tarea 1 en control son:

C Uso de procesador 4 40 000 ns 0.04 ms	**P Prioridad 2**	**PR Periodicidad** 100 100 000 000 ns 100 ms

Paq	**Tiempo**		**Recepción**	**Retardo milisegundos**
	Segundos	**nanoseg**		
1	1465	394389440	Tiempo y caract	0.183168
2	1465	594362656	Paq[80]	0.299984
3	1465	794411136	Paq[80]	0.300061
4	1465	794717920	Paq[80]	0.300058
5	1465	794917928	Paq[80]	0.299995
6	1465	894344704	Paq[80]	0.199392
7	1465	994357568	Paq[80]	0.200008
8	1466	194559136	Paq[80]	0.3000260
9	1466	194867776	Paq[80]	0.3000258

Tabla 5.45.- Retardos y tiempos requeridos para la recepción de 10 paquetes y Periodicidad de 100 ms.

Archivo p0108c02 y03

Se reciben los 9 paquetes pero no se respeta la periodicidad en la llegada de algunos paquetes, el promedio de retardo es de 0.240079 ns (0.24 ms), el uso del procesador por la tarea de control esta en un promedio de 0.044 ms (44670 ns) siendo un poco mayor que el tiempo de ejecución que debería ejecutarse la tarea (Tabla 5.45).

Prueba 5

Mediremos los retardos de la tarea de Sensor y Control enviando 1 000 paquetes que enviaran el tiempo del Sensor y sus características en el primer paquete y paquetes de 80 chars en los siguientes, los valores de Sensor son C=3, P=1, PR=100; En Control la periodicidad es de 100, C=4, prioridad 1;

El promedio de los retardos en Control es 0.199599 ms, el promedio del los retardos en Sensor 0.097803998 ms. (archivos p0208s01, p0208c01), La sincronía de envío y recepción es la adecuada sin perdida de paquetes.

Prueba 6

Mediremos los retardos de la tarea de Sensor y Control enviando 10 000 paquetes que enviaran el tiempo del Sensor y sus características en el primer paquete y paquetes de 80 chars en los siguientes, los valores de Sensor son C=3, P=1, PR=100; En Control la periodicidad es de 100, C=4, prioridad 1;

El promedio de los retardos en Control es 0.199959 ms , el promedio del los retardos en Sensor 0.102000 ms. El tiempo requerido para enviar los 10 000 paquetes fue el esperado 16.66 seg (999.9 seg), (archivos p0208s02, p0208c02). Sin pérdida de paquetes. Los retardos tanto en 1000 como en 10 000 paquetes se mantiene uniforme.

Prueba 7

Bajaremos la periodicidad a 10 tanto en nodo Sensor como en control después mediremos los retardos de la tarea de Sensor y Control enviando 10 000 paquetes que enviaran el tiempo del Sensor y sus características en el primer paquete y paquetes de 80 chars en los siguientes, los valores de Sensor son C=3, P=1, PR=10; En Control la periodicidad es de 10, C=4, prioridad 1;

El promedio de los retardos en Control es 0.202039 ms, el promedio del los retardos en Sensor 0.181774 ms. (archivos p0208s03, p0208c03). Sin pérdida de paquetes. Los retardos tanto en 1000, 10 000 como en esta prueba de 10 000 paquetes y periodicidad 0.01 ms (10 000 000 ns) se mantiene uniforme. No existe pérdida de paquetes pero no se termina en el tiempo esperado 100 segundos o 1 minuto 40 seg. (Sino 191.77 seg tiempo exagerado de terminar el envío y recepción de los paquetes)

5.2.- Análisis de resultados

Conexión entre Sensor con Control y Control con Actuador

Antes del envío y recepción de los paquetes entre Sensor y Control es necesario realizar la conexión entre los nodos, el Sensor se comporta como un cliente en un Sistema Distribuido y el nodo Control se comporta inicialmente como servidor, posteriormente el nodo Control se comporta como cliente y el nodo Actuador se comporta como servidor. El promedio de tiempo necesario para realizar dicha conexiones esta entre los 8,903 milisegundos.

Envío y recepción de paquetes entre Sensor y Control tanto de Control y Actuador

El tiempo de envío de paquetes es comparado con el estándar SCI (Scalable Coherent Interface) el cual fue diseñado por el comité ANSI/IEEE 1592-1992. En esta red SCI se tiene una baja latencia en la transmisión de los datos. Con una tarjeta D330 PCI-SCI, de la empresa Dolphin Interconnect Solutions Inc., con un procesador AMD 760 MPX6 en promedio se requieren 3 microsegundos para transferir un paquete de 512 bytes y latencia de 1.46 microsegundos. Para lograr la sincronización en el envío y recepción de paquetes se realiza la medición del tiempo que tarda un paquete en transportarse a través de la red y después de obtener el promedio del tiempo que tardan n paquetes en llegar desde Sensor a Control. Esta medida se realiza enviando aproximadamente 15000 paquetes de 80 caracteres, el promedio de envío es de 0.08775 milisegundos, el tiempo mínimo esta en 0.02 milisegundos y el tiempo máximo de envío es 0.18 milisegundos. Tomando en cuenta el tiempo promedio de 0.08775 ms es 30 veces más lento que la transferencia de 1 paquete de 512 bytes del estándar SCI, aún tomando el tiempo mínimo de envío sería 7 veces más lento aún teniendo un paquete de 512 bytes por un paquete de 80 bytes de las pruebas realizadas en esta implementación utilizando tarjetas Ethernet PCI 3C905.
La ejecución de procesos que comparten el procesador deben de sincronizarse para evitar la condición de carrera, para evitar este problema se utiliza la exclusión mutua MUTEX tanto en el thread que envía los paquetes como en el thread que recibe.

Tiempo transcurrido de los paquetes entre Sensor y Control y desde Control a Actuador.

El tiempo necesario para que se envíe un paquete de 80 caracteres de Sensor a Control es en promedio 0.08775 milisegundos (prueba realizada con 15,000 paquetes de 80 caracteres). En Control la periodicidad del hilo 1 (tarea de recibir paquetes desde Sensor) es de 10,000 nanosegundos (0.010 milisegundos) para que siempre este listo de recibir datos, además de que el hilo 1 tiene la técnica MUTEX. La planificación me garantiza la ejecución de cada una de mis tareas dentro de cada uno de mis nodos, asignando los recursos de acuerdo a las características de dadas a las tareas tales como prioridad, tiempo de ejecución, periodicidad. Se determina que cada uno de mis nodos esta en tiempo real porque responde a estímulos con retardos en sus tareas de 0.1 milisegundos en el cliente y de 0.19 en el servidor, pero no podemos determinar tiempo real en el transporte de los paquetes en la red por que no se pudo implementar un planificador entre los procesos de cada uno de los nodos, solamente se realiza la comunicación entre nodos y transporte de paquetes entre ellos. De acuerdo a los tiempos de latencia del estándar SCI (Scalable Coherent Interface) el cual es de 1.46 microsegundos (0.00000146 segundos) el tiempo de latencia por SCI es por mucho más eficiente que el utilizado en RTLinux. .

6.- CONCLUSIONES GENERALES

Con respecto a la configuración de cada uno de los nodos del Sistema Distribuido en Tiempo Real a implementar se elige el sistema operativo Mandriva pero el tipo de sistema operativo es indistinto pues se puede realizar con RedHat, Debian y Suse, la versión del kernel utilizada fue la 2.4.20 de la cual se tienen parches de rtlinux para poder trabajar con dicho kernel.
El cambio o modificación en la configuración de rtlinux de cada uno de los nodos en la red es realizada de forma separada a través de Rtl-lwIP.x, se utiliza la versión 0.4 que contiene los módulos para las tarjeta 3com95x con la que contamos en nuestros equipos pero también podemos utilizar la tarjeta ee100 y rt_8139.

6.1.- Garantizar tiempo real

Para garantizar el tiempo real en cada uno de mis nodos se implementa rtlinux y rtl_lwip para la utilización de la tarjeta de red y un stack, además de utilizar un planificador de tareas para la asignación de los recursos, utilizamos hilos que ejecutan mis tareas y la técnica de exclusión mutua para asegurar que mis tareas terminen su ejecución antes de ser sustituidas por otro proceso. No podemos garantizar tiempo real en la red debido a que no pudo llevar a cabo la planificación entre procesos de diferentes nodos, solo se realiza la comunicación entre los nodos Sensor, control y Actuador.

Al contener los nodos la técnica de exclusión mutua la entrega y recepción de paquetes se realiza sin pérdida de datos aunque los retardos en las tareas no son uniformes en algunos casos. Para que los retardos sean uniformes y la sincronía en la entrega y recepción de paquetes sea dada es necesario que la periodicidad de la tarea tanto en el cliente como en el servidor sea mayor que el tiempo mayor que ocupa un paquete en transportarse de un nodo a otro.

6.2.- Tiempo promedio de paquetes

Se verifica que el tiempo promedio de los paquetes (datos) es de 0.087milisegundos y si no se tiene el Control de envío y recepción a través de hilos, planificadores y MUTEX, la pérdida de paquetes o peor aún el estancamiento de los paquetes podrá suceder.
Se garantiza la ejecución de un proceso de con duración de 300 segundos sin pérdida de datos que responde a estímulos que pueden originarse de sonido, video, imagen, etc en el nodo Sensor y que por medio de el nodo Control responde a los estímulos generados en Sensor a través del nodo Actuador el cual responde con una señal enviada al puerto paralelo, esta garantía se dará con la restricción de que la periodicidad de mis tareas tanto en el servidor como en el Actuador sean mayores que el tiempo mayor de traslado de los paquetes en la red.

6.3.- Aportación

La aportación de este trabajo esta en describir la forma de responder a estímulos que son generados en tiempo real en donde los paquetes pueden ser enviados en el orden de milisegundos, mostrar la forma de trabajar en tiempo real en cada nodo en un sistema distribuido así como definir el tiempo de latencia dentro mis nodos y dar el tiempo promedio de traslado de paquetes en un sistema distribuido utilizando el modelo cliente servidor.

TRABAJO FUTURO

El trabajo futuro se relaciona con implementar un planificador que me permita garantizar la entrega de paquetes en tiempo real, utilizar un sistema operativo en tiempo real diferente. La adaptación de wireless en tiempo real para que el sistema distribuido en tiempo real responda en este tipo de red. Utilizar el trabajo realizado en aplicaciones de tiempo real para que este pueda ser visualizado en simulaciones, juegos, etc.

REFERENCIAS

[1] (Almeida, Pres 2003) Luis Almeida; University of Aveiro, Portugal; “Real Time Networks”; Congreso, presentación llevada a cabo en la UNAM, México City, October2003;

[2] (Almeida, Doc 2003) Luis Almeida, University of Aveiro, “Real-Time Networks”, Congreso, documento presentado en la UNAM, México City, October 2003

[3] (Crespo, 2006) Alfonso Crespo, Alejandro Alonso; Revista iberoamericana de automática e informática industrial (RIAI), ISSN 1697-7912, Vol. 3, Nº. 2, 2006 , páginas. 7-18, “Una Panorámica de los Sistemas de Tiempo Real”;Universidad Politécnica de Valencia 2006.

[4] (RATSincRel), Fernando L. Romero, Walter Aroztegui, Fernando G. Tinetti, Publicación de sincronización de Relojes en ambientes Distribuidos, UNLP, Argentina, En la parte de Líneas de investigación y desarrollo.

[5] (QNX N-RTOS), Rob Krten updates by QNX Software Systems, Libro: “QNX Neutrino RTOS Getting Started”, en la parte de Prefacio a la primera edición.

[6] (QNX N-RTOS2), Acerca de QNX, Página de la compañia QNX Software Systems en la dirección http://www.qnx.com/company. Ultimo acceso Febrero, 2009.

[7] (ADNKL), Adam Dunkel, Publicación acerca de Design and implementation of the LWIPTCP/IP Stack desarrollado en “the Computer and Networks Architectures (CNA) lab at The Swedish Institute of Computer Science (SICS).

[8](RTcorbaHS) Honorato Saavedra Hernández, Propuesta de un algoritmo genético para la reconfiguración en línea de un sistema de tiempo real basado en rt-CORBA. Tesis de Maestría en computación en la Universidad Nacional Autónoma de México, 30 de mayo de 2005.

[9] (VxWWce) http://www.windriver.com/products/vxworks/ Pagina de la compañía Wind River, último acceso 6 de junio de 2009.

[10] (FSMLabs-sol, 2008) Pagina de la compañía FSMLabs; en FSMLabs, http://www.fsmlabs.com/solutions/linux-bsd-solutions/; ultimo acceso Diciembre, 2008.

[11] (LynxOS) Pagina de la compañía; en LinuxWorks, http://www.lynuxworks.com/rtos/; ultimo acceso Abril, 2009.

[12] (STANK88) Jhon A. Stankovic, Libro: misconceptions about real-time computing: A serious problem for the next generation system computer, v.21 n.10, p. 10-19, 1988.

[13] (STANK-SCHED) Jhon A. Stankovic, Marco Spuri, Krithi Ramamritham, Giorgio C. Buttazzo, Libro: Deadline scheduling for real time systems,.pag. 4.

APENDICE

Comprobación de Bigphysarea
Debes ver algo como esto (es real en la pc tan):
Big physical area, size 4096B
Free list: used list:
Numbers of blocks: 1 0
Size of largest block: 4096 kB 0Kb
Total: 4096 kB 0 kB

O ejecutando: $cat /proc/ksysms | grep bigphys
Veras (en pc tan no salio nada):
C012e8c8 bigphysarea_alloc
C012e898 bigphysarea_free
C012asc8 bigphysarea_alloc_pages
C012esd8 bigphysarea_free_pages

Módulos cliente servidor y archivos de prueba (módulos y archivos de prueba se integran en un CD)
Sensor (Cliente), Módulos
tcpclient3.c, tcpclient4.c, loperacli, Tcpcabr7.c, tcpcli2kok.c, Tcpcli15abr.c, tcpcli18abr.c, tcpcli21abr.c, tcpcli21abrnoche.c, tcpcli26abrtunok.c
Archivos de Prueba
Fin1, …, fin7, May1001, … , may1006, P1,…, p4, S1,…, s7.
p3007_01s, p0108s01, p0108s02 , p0108s03, p0208s02

Control (Servidor-Cliente), Módulos
tcpserver3.c, tcpserver4.c , loperaservact., Tcpsabr7.c, tcpser2kok.c, Tcpcli15abr.c, tcpser14abr.c, tcpser18abr.c, tcpser21abr.c, tcpser21abrnoche.c, tcpser26abrtunok.c
Archivos de Prueba
Ejerc2, Fin01, …, fin07, lista, Pi1,…, pi4, S1,…, s7, pn01, …, pn10, pnino01,…,pnino21, su1,…,su7, tpaq1,…, tpaq7.
p3007_01, p0108c01, p0108c02 , p0108c03, p0208c02

Actuador (Servidor), Módulos
tcpserver3paralelo.c, tcpserver4paralelo.c