Criterio de Selección para Sistemas Operativos de Tiempo Real utilizados en Nanosatélites

*William Arias, Lenin Luna
Agencia Bolivariana para Actividades Espaciales (ABAE)
Borburata, Venezuela
⋆warias@abae.gob.ve
lluna@abae.gob.ve

***Resumen*—Un sistema operativo de tiempo real (SOTR) responde de manera predecible y rápida a un conjunto de tareas, siendo el enfoque más comúnmente utilizado en proyectos espaciales grandes y complejos. Las tareas son llamadas específicamente cuando es necesario en vez de en un ciclo de ejecución serial como en los sistemas operativos embebidos basados en lazos de control. Los nanosatélites son considerados plataformas cuyo uso es factible en misiones educativas y de demostración de tecnología, sin embargo, en los últimos años se plantea que debido a los avances en la microelectrónica y la experiencia acumulada en misiones previas, su aplicación puede abarcar misiones de alto valor para la humanidad, como las misiones de percepción remota. En este artículo se realiza una revisión de los sistemas operativos en tiempo real y sus aplicaciones en el área espacial, y se hace la propuesta de un criterio de selección para Sistemas Operativos de Tiempo Real a ser usados en nanosatélites.**

***Palabras Claves*—criterio de selección, CubeSat, nanosatélites, Sistema Operativo de Tiempo Real, sistemas embebidos, SOTR.**

***Abstract*—A real-time operating system (RTOS) responds predictably and fast to a set of tasks, being the approach most commonly used in large and complex space projects. Tasks are specifically called when needed instead of a cycle of serial execution as embedded operating systems based on control loops. Nanosatellites are considered platforms whose use is feasible in educational missions and technology demonstration, however, in recent years it is suggested due to the advances in microelectronics and the experience gained in previous missions, its applications may include high value missions to human activities, such as remote sensing missions. In this paper a review of the real-time operating systems and their applications in the space area is performed, and the selection criteria for Real Time Operating Systems to be used in nanosatellites is proposed.**

***Keywords*—CubeSat, embedded systems, nanosatellites, Real-Time Operating System, RTOS, selection criteria.**

I. Introducción

Los nanosatélites son plataformas que en la actualidad resultan útiles no solo en misiones educativas y de demostración de tecnología, sino también en otros tipos de misiones de valor científico. Este tipo de plataformas, por sus dimensiones, presentan limitaciones de espacio y de potencia para realizar sus misiones, por lo que es necesario analizar con detalle todos los elementos que conforman la plataforma satelital. Dependiendo de la arquitectura de hardware a implementar, es posible determinar que sea necesario el uso de un Sistema Operativo de Tiempo Real (SOTR), lo que permite diseñar un sistema de software complejo con mayores capacidades en vuelo y opciones de configuración que ayudan a el alcance de los objetivos de la misión. Sin embargo, el incremento en la versatilidad y flexibilidad incrementa el riesgo de errores críticos. Al momento de seleccionar el SOTR es necesario tener presentes los requerimientos de la misión, de manera de tomar el sistema que se ajuste más a las necesidades; es necesario entonces el uso de un criterio de selección de SOTR que permita analizar las diferentes opciones existentes, tomando en cuenta las alternativas de hardware según la arquitectura seleccionada para la misión.

II. Los Nanosatelites

Los nanosatélites fueron primeramente desarrollados en 1955 para ser usados en comunicaciones. La Universidad Surrey fue la primera universidad en adoptar el uso de nanosatélites en 1981. Para 1999, la Universidad de Stanford y la Universidad Estatal Politécnica de California (California Polytechnic State University), también conocida como Cal Poly, crearon el estándar CubeSat. No fue hasta el 2006 que la NASA (National Aeronautics and Space Administration) se dio cuenta del potencial de los nanosatélites. Los nanosatélites pueden ser usados en varios tipos de misiones, las cuales se mencionan posteriormente.

Los nanosatélites son más pequeños, menos costosos y pueden ser producidos en periodos más cortos de tiempo que los satélites convencionales. Se han hecho populares para las universidades porque proveen un medio de gran adquisición de conocimientos y de cambio rápido. Los nanosatélites son diseñados en subsistemas pequeños. El diseño alrededor de estos subsistemas pequeños permite un desarrollo mas rápido y menos costoso. [1]

A continuación se listan las características [2] y aplicaciones [4] de los nanosatélites:

A. Características

- Comúnmente poseen una masa desde 1 Kg hasta 10 Kg y miden menos de 0.5 m en cualquier dirección lineal. Sin embargo, algunos satélites de hasta 15 Kg son considerados nanosatélites, como el satélite Emerald [3].

- Típicamente tienen ciclo de desarrollo corto (de 1 a 2 años) y bajo costo, frecuentemente utilizando componentes ya existentes en el mercado.
- El tiempo de vida útil se encuentra en un rango desde varias semanas hasta unos pocos años (menos de 5 años) dependiendo de la misión.
- Típicamente son lanzados como cargas útiles secundarias.
- Algunas misiones requieren lanzamientos y operación simultanea de varios satélites.
- Actualmente, muchos nanosatélites usan el espectro asignado a el servicio de satélite amateur y el servicio MetSat en el rango de frecuencia de 30-3000 MHz.
- Tienen características limitadas de control de órbita por lo que tienen características orbitales únicas.

B. Aplicaciones

- Percepción remota.
- Investigación de ambiente espacial.
- Investigación de la atmósfera superior.
- Astronomía.
- Comunicaciones.
- Investigación biológica.
- Demostración de tecnología y educación.
- Otras aplicaciones comerciales, como los servicios de radiocomunicaciones.

Dentro de la categoría de los nanosatélites es común el estándar CubeSat. Las Especificaciones de Diseño de CubeSat (CDS, CubeSat Design Specification) contienen los requerimientos para el diseño mecánico, eléctrico y operacionales de los CubeSat, asi como los requerimientos para los distintos tipos de pruebas. También contiene las características del sistema de despliegue de CubeSat estandarizado por Cal Poly, el P-POD (P-POD, Poly Picosatellite Orbital Deployer) [5].

III. Sistemas operativos de tiempo real

Existen sistemas computacionales fuertemente relacionados con el entorno que les rodea y con el que se encuentran en constante interacción. Ejemplos de este tipo de sistemas son los sistemas de adquisición de datos y los controladores industriales, los cuales están compuestos por diversos elementos (actuadores, sensores, unidades de cómputo, redes de interconexión, etc.) que deben trabajar de forma conjunta y coordinada. Debido a la naturaleza cambiante del entorno con el que interactúan, junto con la necesidad de coordinación entre sus componentes, los resultados obtenidos sólo podrán ser considerados válidos cuando, además de ser correctos desde el punto de vista lógico, hayan sido generados a tiempo. Resultará, por tanto, preciso imponer restricciones temporales cuyo cumplimiento garantice el correcto funcionamiento del sistema. Este tipo de sistemas, capaces de realizar tareas y responder a eventos asíncronos externos dentro de unos plazos temporales determinados son los denominados "Sistemas de Tiempo Real".

Para que sea posible la predictibilidad temporal del sistema completo, todas las partes que le componen deberán de presentar un comportamiento predecible. En consecuencia, en el caso de utilizar un sistema operativo, los servicios que éste proporcione a las aplicaciones deberán presentar tiempos de respuesta acotados, de forma que así sea capaz de garantizar los requerimientos temporales de los procesos bajo su control. Mientras que en un sistema operativo de tiempo compartido, como Unix, lo importante es proporcionar a los usuarios unos buenos tiempos de respuesta promedios, la clave en los sistemas operativos de tiempo real será garantizar los requerimientos temporales; el tiempo de respuesta promedio pasa así a un segundo plano.

Dentro de los sistemas de tiempo real, un tipo particular son los "sistemas embebidos". En este tipo de sistemas, el computador constituye una parte más de un sistema mayor en el que se encuentra altamente integrado y en el que se dedica a realizar una función (o un pequeño conjunto de ellas). Las aplicaciones tradicionales de este tipo de sistemas incluirían sistemas de control en aviones, trenes, nudos de telecomunicaciones, motores de automóviles, procesos industriales, teléfonos móviles, etc.

Sin embargo, la utilización de estos sistemas se ha extendido paulatinamente a otras muchas aplicaciones a medida que los avances tecnológicos permiten disponer de procesadores baratos cada vez más potentes y de memorias de menor coste y mayor escala de integración. Así, ya es habitual encontrarse con computadores embebidos en productos como televisores, juguetes, reproductores de discos CD/DVD/Blue-Ray, pequeños electrodomésticos, etc. Existe una gran variedad de sistemas informáticos utilizados en entornos embebidos, sin embargo, la mayor parte de los sistemas embebidos ven limitadas sus prestaciones por razones de tamaño, peso, consumo o coste. Por las citadas razones, lo normal es que no dispongan de un terminal de propósito general para interfaz con el usuario, ni de sistema de ficheros o dispositivos de almacenamiento, y tengan un procesador y capacidad de memoria muy reducidos. Normalmente no tienen hardware de gestión de memoria o, si lo tuvieran, permanecería sin usar e inhabilitado.

Las limitaciones mencionadas eran mucho más importantes hace unos años debido al menor desarrollo tecnológico, lo que exigía reducir al máximo los tiempos de computo y el tamaño de las aplicaciones embebidas. En consecuencia, la mayor parte de ellas se escribían en lenguaje ensamblador y sin utilizar ningún sistema operativo. En la actualidad, las mejoras tecnológicas y el consiguiente incremento en la complejidad de las aplicaciones han hecho casi imprescindibles la utilización de lenguajes de alto nivel y sistemas operativos en la mayoría de los entornos embebidos.

En este avance también ha influido la necesidad constante de los fabricantes de sistemas embebidos de reducir tiempos de desarrollo y aumentar la fiabilidad de las aplicaciones generadas. Esta mejoras se han logrado en gran medida mediante la aplicación en el desarrollo de las aplicaciones embebidas de los aspectos más avanzados de la ingeniería software para sistemas de tiempo real, como son la programación concurrente, estrategias sofisticadas de planificación de tareas, programación orientada a objetos, etc. Tecnologías éstas, que

difícilmente habrían podido ser aplicadas sin la utilización de lenguajes de alto nivel y sistemas operativos.

En un principio, fueron los fabricantes de los sistemas de hardware los que gradualmente procedieron a incorporar algunos de estos conceptos en sus sistemas de desarrollo específicos.

La consecuencia de esta acción no coordinada y casi siempre competitiva, era un gran desorden conceptual, que requería de las empresas un considerable esfuerzo para transferir o adaptar el software a las diferentes plataformas de hardware de sus controladores industriales, que por su naturaleza son en general poco homogéneos. La solución a esta problemática se está buscando a nivel internacional mediante la elaboración de normas de estandarización, que permiten independizar los diferentes aspectos del diseño del sistema (arquitectura del hardware, software de base, sistema operativo, aplicación), consiguiendo con ello una mayor modularidad, reusabilidad y, en definitiva, la reducción del tiempo de diseño y de los costos.

Entre las normas internacionales de estandarización destaca el estándar POSIX (conocido en el ámbito internacional con la referencia ISO/IEC-9945), cuyo objetivo es permitir la portabilidad de aplicaciones a nivel de código fuente entre diferentes sistemas operativos. Con este fin, y basándose en la interfaz del extensamente utilizado sistema operativo Unix, el estándar POSIX (Portable Operating System Interface) define la interfaz que los sistemas operativos deben ofrecer a las aplicaciones, así como la semántica de los servicios ofrecidos por esa interfaz.

La interfaz descrita en el estándar permite escribir aplicaciones de tiempo real de forma portable . Sin embargo, debido al tamaño del estándar POSIX, resulta inabordable su implementación completa en un sistema operativo destinado a computadores embebidos pequeños. Por esta razón en el estándar POSIX.13 se definieron cuatro subconjuntos de servicios del sistema operativo, que se orientaron a cuatro tipos de plataformas comúnmente utilizadas en aplicaciones comerciales de tiempo real. El menor de dichos perfiles, el denominado "Sistema de Tiempo Real Mínimo", fue pensado para aplicaciones embebidas pequeñas. Dicho perfil incluía un pequeño subconjunto de toda la funcionalidad definida en el estándar POSIX, lo que permitió su implementación como un núcleo de sistema operativo pequeño y eficiente. A partir de 2009, POSIX es conocido como POSIX.1-2008 o IEEE Std 1003.1-2008.

El estándar POSIX goza de gran aceptación entre los fabricantes. Así, en los últimos años aparecieron numerosos sistemas operativos de tiempo real conformes (en mayor o menor medida) con el estándar POSIX. La mayoría de ellos, entre los que se encuentran los más utilizados por la industria a nivel mundial, son sistemas propietarios tales como VxWorks, pSOSystem, QNX o LynxOS. Pero también este estándar es seguido por gran cantidad de sistemas operativos distribuidos bajo políticas de libre distribución y código libre, entre ellos que cabe citar MiThOS , RTEMS, RT-Linux y S.Ha.R.K.

Otro estándar de facto en sistemas operativos lo constituye Microsoft Windows en sus diversas versiones. Sin embargo, a pesar de su extensa utilización no podemos considerarlo un estándar real, desde el momento que la especificación de su interfaz depende de una sola compañía que puede cambiarla cuando así lo desee. Por otro lado, la mayor parte de las versiones de Windows constituyen sistemas operativos de propósito general, no resultando por tanto apropiadas para aplicaciones embebidas ni de tiempo real. Existe una versión de tiempo real, denominada Windows CE, pensada para sistemas embebidos de tamaño medio (Windows CE tiene un tamaño mínimo o "footprint" de 400Kbytes) que también adolece de los problemas de falta de estandarización ya comentados.

Otro importante conjunto de estándares en el área de las aplicaciones de tiempo real está constituido por los lenguajes de programación. Así, el C++, y el lenguaje Ada, aportan mecanismos para poder aplicar programación orientada a objetos. La versión actual de ADA, conocida como ADA 2012, se define como ISO/IEC 8652:2012 [6].

A. Kernel de Tiempo Real

Un proceso es una abstracción de un programa en ejecución y es la unidad lógica de trabajo programada por un sistema operativo. Es típicamente representado por una estructura de datos que contiene al menos un estado de ejecución, una identidad (tiempo real), atributos (por ejemplo, tiempo de ejecución), y los recursos asociados a el. Un hilo es un proceso liviano que comparte recursos con otros procesos e hilos. Cada hilo debe "residir", dentro de algún proceso y hacer uso de los recursos de ese proceso. Los hilos que residen dentro del mismo proceso comparten los recursos del proceso. Los sistemas operativos de tiempo real deben proveer tres funciones específicas con respecto a las tareas: programación, despacho, e intercomunicación y sincronización. El programador determina que tarea se ejecutará próximamente en un sistema multitarea, mientras que un despachador realiza el almacenamiento o restauración de las condiciones del sistema para realizar una tarea. La comunicación y sincronización entre tareas asegura que las mismas cooperen. Un nanokernel provee el manejo de hilos simples (procesos livianos). Esencialmente provee sólo uno de los tres servicios provistos por un kernel, mientras que un microkernel en adición provee programación de tareas. Un kernel también provee intercomunicación y sincronización entre tareas por medio de semáforos, cajas de correo y otros métodos. Un ejecutor de tiempo real es un kernel que incluye bloques de memoria privatizada, servicios de entrada y salida, y otras características complejas. La mayoría de los kernel de tiempo real comerciales son ejecutivos. Finalmente, un sistema operativo es un ejecutor que provee una interfaz de usuario generalizada, seguridad y un sistema de manejo de archivos. Sin importar la arquitectura usada en un sistema operativo, el objetivo es satisfacer los requerimientos de comportamiento de tiempo real y proveer un entorno multitarea que sea flexible y robusto. Para satisfacer los requerimientos mencionados para un sistema de tiempo real existen varios enfoques, los mas comunes se mencionan a continuación [7]:

- Pseudokernel: en donde los objetivos de multitarea y de tiempo real pueden ser alcanzados sin interrupciones y sin un sistema operativo.
  - Bucle sondeado (Polled Loop): estos son utilizados para respuesta rápida en dispositivos sencillos. Una instrucción sencilla y simple es utilizada para probar una bandera, la cual al activarse indica que el evento esperado ha ocurrido.
  - Bucle sondeado sincronizado: se utilizan para eventos que presentan el efecto de rebote característico de los interruptores. Entonces, por medio de un temporizador se inhibe el muestreo de la señal que activa el evento hasta que la misma se estabilice, de manera de que no se interprete la presencia de un evento mas de una vez durante la misma ocurrencia.
  - Ejecutores cíclicos: son sistemas no manejados por interrupciones que provocan la ilusión de simultaneidad en la ejecución de varias tareas debido a la rápida velocidad de ejecución de las mismas gracias a las velocidades de procesamiento de los procesadores. Estas se ejecutan en un ciclo continuo.
  - Estados manejados por código: estos sistemas utilizan sentencias condicionales anidadas en su codificación, de manera de hacer mas corta la ejecución de tareas cuya programación de manera convencional haría que el tiempo de su ejecución incremente.
  - Corrutinas: también conocido como multitarea cooperativa, son implementados unidos de los estados manejados por código por autómatas de estados finitos. En este esquema, dos o más procesos son codificados en fases haciendo uso de la forma de estados manejados por código, y luego de que cada fase es completada, se hace una llamada al despachador central. Entonces, el despachador selecciona el próximo proceso a ser ejecutado. Este proceso se ejecuta hasta que la próxima fase es completada y se realiza la llamada al despachador nuevamente.
- Sistemas manejados por interrupciones: en estos sistemas, el programa principal consta de una simple instrucción de llamado a ella misma. Las tareas en el sistema son programadas por medio de interrupciones de software y de hardware. El despacho es realizado por medio de las rutinas de manejo de interrupciones.
  - Rutinas de Servicio de Interrupciones (Interrupt Service Routine, ISR): al ocurrir una interrupción, el programa se detiene y el CPU invoca la ISR. Estas rutinas por lo general son escritas para cada tipo de interrupción. Una fotografía del sistema, llamada contexto, debe ser preservada de manera que, luego de servir la interrupción, el sistema pueda continuar con el proceso que fue interrumpido.
  - Cambio de Contexto: es el proceso de guardar y restaurar información suficiente de una tarea de tiempo real de manera que esta pueda continuar su ejecución luego de ser interrumpida. Esta información por lo general incluye el contenido de los registros generales, del contador del programa, de los registros del coprocesador (si está presente), el registro de la paginación de memoria, imágenes de localizaciones de entrada/salida mapeadas en la memoria.
- Sistemas de Prioridad Preferencial: en estos sistemas una tarea de alta prioridad puede interrumpir a una tarea de baja prioridad. La prioridad de las interrupciones se asigna según la urgencia de las tareas asociadas con la interrupción. La prioridad de las interrupciones puede ser fija o dinámica.
- Sistemas Híbridos: estos sistemas incluyen interrupciones que ocurren a intervalos fijos y esporádicos. Las interrupciones esporádicas pueden ser usadas para manejar errores críticos que requieren atención inmediata, teniendo entonces la mas alta prioridad. Este tipo de sistemas es comúnmente utilizado en aplicaciones embebidas. Otro tipo de sistema híbrido que se encuentra en sistemas operativos comerciales se basa en una combinación de sistemas de prioridad preferencial y de sistemas secuenciales. En estos sistemas, las tareas de alta prioridad siempre interrumpen a las de baja prioridad. Sin embargo, si dos o mas tareas de la misma prioridad están listas para ejecutarse simultáneamente, entonces estas se ejecutaran de manera secuencial.
  - Sistemas primer plano / segundo plano: son una mejora de los sistemas basados en interrupciones, en donde el ciclo del programa principal es reemplazado por código que realiza un procesamiento útil. Estos sistemas representan la arquitectura mas común para sistemas embebidos. Contienen un juego de procesos manejados por interrupciones, o procesos de tiempo real, llamados “primer plano” y una colección de procesos no manejados por interrupciones llamados “segundo plano”. Las tareas de primer plano se ejecutan de manera secuencial, de prioridad preferencial, o de forma híbrida. Las tareas de segundo plano pueden ser interrumpidas por cualquier tarea de primer plano y representan las tareas de menor prioridad del sistema. Todas las soluciones de tiempo real son casos especiales de sistemas primer plano / segundo plano.
  - SOTR con todas las funciones: la solución primer plano / segundo plano puede ser extendida a un sistema operativo agregando funciones adicionales como interfaces de red, manejadores de dispositivos, y herramientas de depuración complejas. Estos tipos de sistemas son disponibles como productos comerciales. Estos sistemas se basan en sistemas operativos complejos que hacen uso de ejecución de manera secuencial, de prioridad preferencial, o una combinación de ambos esquemas para proveer programación; el sistema operativo representa la tarea de prioridad mas alta, kernel o el supervisor.

Entre los recursos relevantes de los SOTR se encuentran las colas, los mutexes y los delays [8].

- Colas: son un recurso del sistema que permite el envío y/o recepción de datos entre tareas. Al crear una cola se debe definir la cantidad de elementos y el tamaño de los mismos.
- Mutex: es utilizado para proteger un recurso del sistema. Cuando una tarea desea tener acceso un recurso protegido, esta debe tomar el Mutex para garantizar que ninguna otra tarea pueda tener acceso al mismo. Cuando la primera tarea ya no requiera el recurso, debe entregar en Mutex para permitir que otras tareas lo utilicen.
- Delay: este recurso del sistema bloquea a la tarea durante el tiempo establecido, cediendo el control del procesador al sistema operativo. La tarea vuelve a estar en estado Listo al terminar este tiempo de espera.

B. Características

Los SOTR generan una respuesta lógicamente correcta a eventos asíncronos externos dentro de los requerimientos de tiempo. Tiempo real significa que el Sistema Operativo garantiza una latencia reducida para los eventos e interrupciones, dado un diseño adecuado del software y las tareas. [9], [10]

Entre las principales características que los SOTR deben tener se encuentran:

- Organizar tareas de manera lógica, asignando una prioridad a cada una.
- Garantizar la ejecución correcta de todas las tareas críticas.
- Buen tiempo de respuesta a las tareas que no tienen plazo de culminación.
- Administrar adecuadamente el uso de los recursos compartidos.
- Recuperación ante fallos de hardware y software.
- Soportar los cambios de modo.
- Buen tiempo de respuesta a las interrupciones.
- Eficiencia en los cambios de contexto (procesos ligeros).

C. Aplicaciones

Entre las aplicaciones tenemos [6]:

- Sistemas de Adquisición de Datos.
- Controladores de Procesos Industriales.
- Control de Naves Espaciales.
- Control de Aviones.
- Trenes.
- Nodos de Telecomunicaciones.
- Motores de Automóviles.
- Teléfonos Móviles.
- Televisores.
- Reproductores de Discos.
- Etc.

D. Ejemplos De RTOS

A continuación se muestran unos ejemplos de RTOS [9]:

- FreeRTOS
- QNX
- RTlinux
- VxWorks
- Windows CE
- μC/OS-II
- DSPBios
- μClinux
- FreeRTOS

IV. Sistemas operativos de tiempo real en el área espacial

Dentro de los sistemas operativos en tiempo real utilizados en el área espacial tenemos [11]:

- RTEMS usado en Europa.
- VxWorks usado en Estados Unidos.

A. RTEMS

Es un sistema operativo de tiempo real determinístico y libre, el cual es compatible en una gran cantidad de sistemas embebidos. Fue desarrollado por la armada de los Estados Unidos y luego fue colocado en el dominio público. El proyecto invita a el soporte y uso de APIs estándares con el fin de promover la portabilidad de aplicaciones así como anexar otros paquetes a el entorno de RTEMS. El desarrollo de RTEMS utiliza un ambiente de desarrollo en el cual todos los usuarios colaboran para mejorar el sistema. El set de herramientas de desarrollo de RTEMS is basado en herramientas GNU libres y de la librería de C de código abierto newlib. Soporta una larga variedad de familias de procesadores tales como ARM, Motorola 68K, PowerPC 750, Intel 80386, SuperH (SH), MIPS y SPARC (ERC32 y LEON). [11], [12]

- Características:
  - Cumple con el estándar POSIX-1003,1b-API incluyendo manejo de hilos.
  - Soporta RTEID/ORKID-API
  - Soporta ulTRON-3.0-API
  - Soporta redes TCP/IP.
  - Disponibilidad del set de herramientas de GNU.
  - Compatibilidad con la interfaz de eliminación de bugs GDB de GNU.
- Aplicaciones:
  - Misiones espaciales alrededor de la tierra como Pleiades y Galileo.
  - Misiones en los puntos de Lagrange Tierra-Sol L2 como Hershel y Planc.
  - La misión Venus express alrededor del planeta Venus.
  - La misión Electra alrededor de Marte.
  - Entre otras [13].

B. VxWorks

Es un sistema operativo de tiempo real propietario que soporta diversas arquitecturas de procesadores (ARM, IA32, Intel 64, MIPS, PowerPC, SH-4, StrongARM, xScale) y es complementada con una juego de herramientas llamada Tornado la cual incluye depurador de código, analizador de programador, etc. [11],[14]

- Características:
  - Soporta procesador de un solo núcleo o multinúcleo.
  - CPU de 32 o 64 bits.
  - Escalabilidad, modularidad y ajuste de rendimiento del kernel.
  - Administración y protección de memoria.
  - Conectividad (Bluetooth, USB, CAN, IPv4/IPv6, entre otros).
  - Avanzadas características de Red de datos (Firewall, NAT, Wireless Security, SSL, entre otros).
  - entre otras.
- Aplicaciones:
  - Satélite PROBA (ESA) [15].
  - Laboratorio de Ciencia de Marte de la NASA [16].
  - Orbitador de reconocimiento de Marte [17].
  - Experimento Científico del Programa de Espacio Profundo [18].
  - Misión Pathfinder de Marte [19].

V. Sistemas operativos de tiempo real en los nanosatélites

La implementación de un sistema operativo de tiempo real en un proyecto de un pequeño satélite tiene una extensiva lista de ventajas y desventajas. Una estructura multihilo de un sistema operativo de tiempo real permite la habilidad de diseñar un sistema de software complejo con mayores capacidades en vuelo y opciones de configuración que ayudan a el alcance de los objetivos de la misión. El incremento en la versatilidad y flexibilidad incrementa el riesgo de errores críticos, como pérdida de memoria, latencia en la programación, y concurrencia de colas de mensajes. [20]

Algunos de los sistemas operativos utilizados en misiones con nanosatelites son:

- μClinux
- FreeRTOS
- Pumpkin Salvo
- VxWorks

A. μClinux

Originalmente fue una derivacion del Kernel 2.0 de Linux, teniendo como objetivo microcontroladores sin Unidades de Manejo de Memoria (Memory Management Units, MMUs), sin embargo, el proyecto de implementación de Linux en microcontroladores ha crecido en reconocimiento de fabricantes y en la cobertura de arquitecturas de procesadores. En la actualidad uClinux como un sistema operativo incluye entregas de kernel de Linux 2.0, 2.4 y 2.6 [22] asi como colecciones de aplicaciones de usuario, librerías y set de herramientas.

Por encima de las opciones de kernel están las opciones en la librería libc. Típicamente los sistemas uClinux usan las librerías de C, tanto uC-libc como uClibc. Usualmente la libreria uC-lib es un poco mas pequeña, con funcionalidad restringida, mientras que uClibc intenta proveer una APLI de C mas completa [23].

Desde la versión 2.6 la mayor parte de uClinux ha sido integrada con la linea principal del kernel de Linux para varias arquitecturas de procesadores [21]. El proyecto continua desarrollando parches y herramientas de soporte para usar Linux en microcontroladores [24]. μClinux soporta varios procesadores, entre los que se pueden mencionar [25]:

- Motorola DragonBall y Coldfire
- ADI Blackfin
- ETRAX
- ARM7TDMI y MC68EN302
- entre otros.

Este SOTR ha sido usado en diferentes proyectos, como su implementación en un procesador Blackfin BF537 para el desarrollo de software para CubeSat [26]. Dentro de los satélites que funcionan con una versión de uClinux se encuentran el UWE-2 (University of Wuerzberg Experimental-2) [27] y el DANDE (Drag and Atmospheric Neutral Density Explorer) [20]. UWE-2 refiere su uso para el sistema OBDH (On Board Data Handling) y DANDE para el sistema de CDHS (Command and Data Handling System).

B. FreeRTOS

Soporta microcontroladores de los siguientes fabricantes [28]:

- Atmel
- Infineon
- Luminary Micro / Texas Instruments
- Microchip
- entre otros.

Este SOTR ha sido usado en proyectos como el Diseño del subsistema de comando y manejo de datos a prueba de fallos para el ESTCube-1 [29].

C. Pumpkin Salvo

Salvo es un Sistema Operativo de Tiempo Real diseñado expresamente para sistemas embebidos de muy bajo costo, con limitaciones muy severas en memoria de datos y de programa.[30]. Pumpkin ha certificado a Salvo para usarlo en las siguientes familias de procesadores:

- La familia 8051 y sus derivados
- ARM® ARM7TDMI® y Cortex™ M3
- Atmel® AVR® y MegaAVR™
- Algunos Microcontroladores y Procesadores de Señales Digitales de TI y Microchip.
- entre otros.

Algunas de las características que menciona el fabricante son las siguientes:

- Sistema multitarea cooperativo manejado por eventos basados en prioridades.
- Diseñado para procesadores de un sólo chip de RAM limitada.
- Provee comunicación y sincronización intertareas, comunicación desde ISR a las tareas y compartimiento de recursos.
- Soporta 16 niveles separados de prioridad de tarea dinámica, las tareas de la misma prioridad se ejecutan secuencialmente.

En [31] tenemos un ejemplo de un proyecto que hace uso de este SOTR.

D. VxWorks

Un ejemplo de uso de este SOTR se encuentra en [32].

VI. Selección de un sistema operativo de tiempo real para un nanosatélite

Al momento de seleccionar el sistema operativo de tiempo real para el uso en una misión satelital con nanosatélite, al igual que para alguna otra aplicación que utiliza un sistema embebido, se puede escoger entre desarrollar un nuevo SOTR o se puede escoger alguna solución comercial existente [33]. Se debe encontrar el balance entre las características que se quieren del SOTR y las capacidades espaciales y energéticas de la plataforma que se va a utilizar [8]. Los requerimientos de la misión satelital como consumo de energía, estados de operación entre otros, los cuales son determinantes en la selección de la arquitectura de Hardware, son determinantes en la selección del SOTR [8]. Entonces se requiere encontrar las mínimas características de hardware que permitan alcanzar los objetivos de la misión.

Un criterio de selección de SOTR para nanosatélites [29], que no debe comprometer los requerimientos funcionales de la misión es el siguiente:

- Poca utilización de memoria de Flash y RAM.
- Baja sobrecarga de rendimiento.

A. ¿Seleccionar de un SOTR comercial o desarrollar uno nuevo?

Al momento de especificar los requerimientos de un sistema que demande el uso de un sistema operativo de tiempo real se puede considerar la adopción de una solución comercial o desarrollar uno nuevo. Esta selección dependerá del diseño pero, en general, un kernel comercial es preferido porque generalmente estos proveen servicios robustos, son fáciles de usar y pueden ser portátiles.

A continuación se muestran las ventajas y desventajas de los sistemas operativos comerciales [33].

Ventajas:

- Alto rendimiento y amplio rango de características.
- Soporta muchos dispositivos estándares y protocolos de red.
- Frecuentemente estos sistemas vienen equipados con herramientas útiles para desarrollo y depuración las cuales se pueden ejecutar en variedades de hardware y ambientes.
- Proveen flexibilidad en la disciplina de programación y el número de tareas soportadas.

Desventajas:

- Son usualmente más lentos que los sistemas que usan el modelo manejado por interrupciones debido al gran consumo de recursos que involucra implementar el modelo de bloque de control de tareas, que es la típica arquitectura de los SOTR comerciales.
- Pueden incluir características innecesarias, las cuales son incorporadas para que el producto tenga amplio atractivo. El tiempo de ejecución y los costos de almacenamiento de estas características puede ser excesivo.
- Los fabricantes pueden estar tentados a mostrar los mejores casos de rendimiento. Los tiempos de respuesta en los peores casos, el cual representa un parámetro más informativo, puede ser generalmente no conocido.

En los sistemas embebidos, cuando el costo por unidad de los productos comerciales es muy alto, cuando se desean características no disponibles en el mercado, o cuando el consumo de recursos es muy alto, la única alternativa es la de escribir un kernel de tiempo real. Pero esta no es una tarea trivial, por lo tanto, un SOTR debe ser considerado cuando sea posible.

B. Selección de un kernel de tiempo real

Desde una perspectiva técnica y de negocio, la selección de un SOTR comercial representa una decisión crítica. Entonces es imperativo que un criterio de selección riguroso sea usado. Las siguientes características son deseables en un sistema de tiempo real [33]:

- Puntualidad
- Diseño para la supervivencia bajo carga máxima
- Previsibilidad
- Tolerancia a fallos
- Mantenibilidad

Entonces el criterio de selección debe reflejar estos datos.

De acuerdo con [33], para determinar objetivamente si un SOTR es apto para una aplicación dada, se debe cumplir con alguno de estos items:

- Debe existir una base de experiencia exhaustiva de uso de varias alternativas comerciales de SOTR en múltiples e idénticos dominios de aplicación.
- Confiar en reportes de éxito o de fallas de terceros, los cuales se pueden conseguir en Internet.
- Comparar alternativas basadas en la información publicada por el fabricante en sus folletos, reportes técnicos y sitios web.

Otro enfoque que permite seleccionar un SOTR, consiste en hacer comparaciones en el rendimiento de varios sistemas. Se pueden realizar comparaciones entre varios SOTR en cuanto a la latencia presentada en la respuesta a las interrupciones, puesto que es el tiempo de respuesta la característica crítica en estos tipos de sistemas [34].

Se puede comparar el rendimiento de varios SOTR realizando pruebas del sistema en cuanto a tres recursos del mismo, como lo son las colas, las mutex y los retardos (delays) [8].

A continuación se describen dos criterios de selección de los SOTR, el primero basándose en la información de mercadeo provista por el proveedor [33] y el segundo en pruebas de rendimiento de uso del SOTR [34].

1) Selección de un SOTR basándose en información publicada:: A continuación se presenta una técnica para comparar SOTR basados en información publicada [33]. Esta puede ser usada con información suplementaria de la experiencia actual de reportes de terceros.

Se consideran 13 criterios de selección, $m_1 ... m_{13}$, cada uno con un rango de $m_i \in [0, 1]$, donde la unidad representa la mayor satisfacción del criterio y cero representa completa insatisfacción. A continuación se enumeran:

1. La mínima latencia de interrupción, m_1, mide el tiempo entre la ocurrencia de la interrupción de hardware cuando el servicio de la rutina de interrupción comienza su ejecución. Un valor bajo representa una relativa alta latencia de interrupción, mientras que un alto valor representa una baja latencia. Este criterio es importante porque si la latencia mínima es mas grande que la requerida por el sistema embebido, un sistema operativo diferente tiene que ser elegido.
2. Este criterio, m_2, define la mayor cantidad de procesos que el sistema operativo puede soportar de manera simultanea. A pesar de que un sistema operativo puede soportar un gran numero de n tareas, esta métrica debe estar limitada a la memoria disponible. Este criterio es importante para sistemas que necesitan numerosos procesos simultáneos. Un relativo alto numero de tareas soportadas puede resultar en $m_2 = 1$, mientras que pocas tareas soportadas puede sugerir un valor bajo de m_2.
3. Criterio m_3 especifica la memoria del sistema requerida para soportar el sistema operativo. No incluye la cantidad de memoria adicional requerida para correr el software de aplicación del sistema. El criterio $m_3 = 1$ sugiere un requerimiento mínimo de memoria, mientras que $m_3 = 0$ representa un gran requerimiento de memoria.
4. El criterio del mecanismo de programación, m_4, enumera si el mecanismo usado es de prioridad preferencial, ejecutor cíclico o algún otro mecanismo de programación de tareas es usado por el sistema operativo. Si varios mecanismos son soportados, entonces un alto valor sera asignado a m_4.
5. El criterio m_5 se refiere a la disponibilidad de métodos que el sistema operativo tiene para permitir a los procesos comunicarse entre ellos. Entre las posibles elecciones están las exclusiones mutuas (mutexes), semáforos binarios y semáforos de conteo, tubos POSIX, colas de mensajes, memoria compartida, memorias intermedias (buffer) FIFO, sockets de control, señales y programación. Cada mecanismo tiene ventajas y desventajas. Con $m_5 = 1$ el SOTR provee todos los mecanismos requeridos. Un valor bajo de m5 implica que pocos mecanismos de programación están disponibles.
6. El criterio m_6 se refiere al soporte post-ventas que una compañía coloca detrás de sus productos. La mayoría de los vendedores ofrecen algún tipo de soporte técnico gratuito por un periodo corto después de la venta, con la opción de comprar soporte adicional si es necesario. Algunos incluso ofrecen consultas en sitio. Un valor alto podría ser asignado a un fuerte programa de soporte, mientras que $m_6 = 0$ si el soporte no es provisto.
7. Disponibilidad de aplicaciones, m_7, se refiere al monto de software disponible (tanto el que se entrega on el sistema operativo o esta disponible por algún medio) para desarrollar aplicaciones que se ejecutan en el sistema operativo. Por ejemplo, RTLinux es soportado por el juego de software de GNU, el cual incluye el compilador de C gcc y muchos depuradores gratuitos disponibles, y otros software de soporte, esta es una consideración importante, especialmente cuando se usa un sistema operativo en el cual no se esta familiarizado. Con $m_7 = 1$ una gran cantidad de software esta disponible, mientras que $m_7 = 0$ significaría que poco o ninguno esta disponible
8. Criterio m_8 se refiere a los diferentes procesadores que el sistema operativo soporta. Esto es importante en términos de portabilidad y compatibilidad con hardware y software comerciales disponible. Este criterio también abarca la gama de periféricos que el sistema operativo puede soportar, como el vídeo, audio, SCSI, entre otros. Un valor alto para el criterio representa un SOTR muy portátil y compatible.
9. Criterio m_9 se refiere a si el código del sistema operativo será disponible al desarrollador, para ajustar o realizar cambios. La fuente también permite hacer una vision de la arquitectura del SOTR, que es muy útil para la depuración y la integración de sistemas. Con $m_9 = 1$ sugiere código fuente abierto o el código fuente libre, mientras que un valor menor podría ser asignado en proporción al precio compra del código fuente. Con $m_9 = 0$ el código fuente no esta disponible.
10. Criterio m_{10} se refiere al tiempo que tarda el kernel para salvar el contexto cuando se va a cambiar de una tarea a otra. Un tiempo de cambio de contexto relativamente rápido resultaría en un valor más alto para m_{10}.
11. Este criterio está directamente relacionado con el costo de solo el SOTR. Esto es fundamental porque para algunos sistemas, el costo RTOS puede ser desproporcionadamente alto. En cualquier caso, un costo relativamente alto se le asigna un valor muy bajo, mientras que un bajo costo merecería un mayor valor para m_{11}.
12. Este criterio, m_{12}, lista las plataformas de desarrollo disponibles. En otras palabras, es un listado de las otras plataformas que son compatibles con el SOTR dado. Un valor alto para m_{12} representaría una amplia compatibilidad, mientras que un menor valor indicaría la compatibilidad con una sola plataforma.
13. Este criterio, m_{13}, se basa en una lista de las redes y protocolos de red que son compatibles con el SOTR dado. Esto sería útil conocerlo porque valora lo que los métodos de comunicación del software que se ejecuta en este sistema operativo sería capaz de utilizar para comunicarse con otros equipos dentro de la misma red. Un valor alto para el criterio representa un número relativamente grande de las redes compatibles.

TABLA I
VALORES DE LOS CRITERIOS DE SELECCIÓN

Criterio	Descripción	Puntuación	Comentario
m_1	Latencia mínima de interrupción		
m_2	Número de tareas soportadas		
m_3	Requerimientos de memoria		
m_4	Mecanismo de programación		
m_5	Mecanismo de sincronización entre tareas		
m_6	Tiempo de cambio de contexto		
m_7	Entorno de programación		
m_8	Portabilidad		
m_9	Disponibilidad de código fuente		
m_{10}	Soporte de Software (garantía)		
m_{11}	Costo		
m_{12}	Cuota por Compatibilidad		
m_{13}	Soporte de Redes		

Reconociendo que la importancia de los criterios individuales variará en función de la aplicación, un factor de ponderación, $w_i \in [0, 1]$, se utilizará para cada criterio m_i, en el que se asigna la unidad si el criterio es el de mayor importancia, y cero si el criterio no es importante para una aplicación particular. Entonces una métrica de ajuste, $M \in [0, 13]$, se forma comose muestra en la ecuación (1).

$$M = \sum_{i=1}^{13} w_i m_i \tag{1}$$

Claramente, un mayor valor de M significa que el SOTR es muy adecuado para la aplicación, mientras que un valor inferior significa que el SOTR no es muy adecuado para la misma.

Mientras que la selección de los valores para mi y wi será subjetiva para cualquier SOTR dado y cualquier aplicación dada, la disponibilidad de esta métrica heurística proporciona una forma para la comparación objetiva, perspectiva histórica, y otros usos. Para cada cada SOTR se debe completar los datos de la Tabla I.

Luego de llenar los valores de los criterios de selección para cada SOTR, se deben asignar los factores de ponderación wi según la aplicación, como se muestra en la Tabla II. Finalmente se pueden observar los diferentes valores para M, los que permite hacer una comparación de cual SOTR es más acorde con la aplicación.

2) Selección de un SOTR basándose en pruebas de rendimiento del sistema: Un enfoque de evaluación propuesto en [34] consiste en el uso del puerto paralelo del PC para recibir una interrupción y generar una respuesta a esta interrupción, lo que permite probar el sistema como una caja negra. Por medio del uso de un generador de señal externo y un osciloscopio, es posible obtener la latencia para manejar las interrupciones y la fluctuación de fase (una variación aleatoria de la medición de una latencia a otro). Este enfoque se basa en los siguientes criterios:

TABLA II
MATRIZ DE COMPARACIÓN DE VARIOS SOTR PARA UNA APLICACIÓN

Criterio	Descripción	Peso w_i	SOTR A	SOTR B	SOTR C
m_1	Latencia mínima de interrupción				
m_2	Número de tareas soportadas				
m_3	Requerimientos de memoria				
m_4	Mecanismo de programación				
m_5	Mecanismo de sincronización entre tareas				
m_6	Tiempo de cambio de contexto				
m_7	Entorno de programación				
m_8	Portabilidad				
m_9	Disponibilidad de código fuente				
m_{10}	Soporte de Software (garantía)				
m_{11}	Costo				
m_{12}	Cuota por Compatibilidad				
m_{13}	Soporte de Redes				
M					

- Uno de los métodos más precisos para medir el tiempo de ejecución es a través de puertos de salida.
- Las pruebas de latencia sólo pueden llevarse a cabo por medios externos.
- Las métricas más comunes de los sistemas operativos para medir la calidad es el tiempo de conmutación de tareas entre los dos procesos y la latencia hasta el inicio de una interrupción rutina de controlador.

Teniendo en cuenta lo anterior, se hacen las comparaciones teniendo cada sistema probado como una caja negra. Las pruebas se deben realizar generando estímulos externos con un generador de señales, y el análisis de la respuesta de estos estímulos con un osciloscopio. Para garantizar la fiabilidad de los resultados, todos los experimentos deben ser ejecutados en la misma plataforma y ser sometidos a varios escenarios de carga diferentes (normales y de uso sobrecargado). La Fig. 1 muestra la configuración utilizada en [34] para realizar las pruebas.

Aunque la medición de la latencia de interrupción a través de mecanismos externos es comúnmente utilizado, la Sociedad Internacional de Medición y Control (International Society for

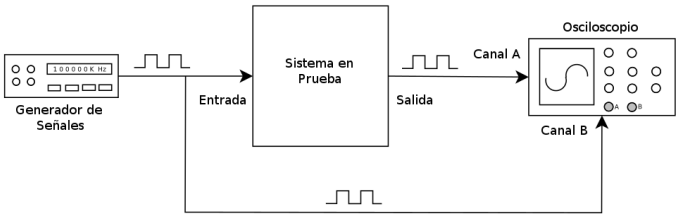

Fig. 1 Montaje experimental para realizar las pruebas de rendimiento.

Measurement and Control, ISA), a través de su comité para la automatización y control (Automation & Control Systems Committee, A&CS) no recomiendan el uso de latencia como una medida de respuesta en tiempo real. La prueba propuesta por el A&CS consiste en el establecimiento de un sistema que copia la señal de entrada directamente a un puerto de salida, y la medición de la cantidad de pulsos que se generan en la entrada y cuántos se copian a la salida. Teóricamente, mientras el sistema es estable, el número acumulado de pulsos en ambos puertos deben ser iguales.

Luego, la frecuencia de la señal de entrada debe ser incrementada poco a poco hasta que el conteo de pulsos de entrada y salida empieza a divergir. En este momento, la frecuencia debe reducirse hasta que se encuentre la frecuencia máxima de funcionamiento del sistema. Esta frecuencia es la que no causa diferencia en el contador de impulsos de la entrada al puerto de salida. Esta frecuencia obtenida representaría la inversa del tiempo de respuesta del peor de los casos.

Como consecuencia de las consideraciones presentadas anteriormente, los parámetros cuantitativos seleccionados para ser analizados en los sistemas son:

1. Latencia: se analiza tomando externamente el SOTR bajo prueba en conjunto con el hardware como una caja negra. La latencia consiste en la diferencia de tiempo entre el momento en que se genera una interrupción y el momento en que el controlador de interrupción asociado genera una respuesta externa. La latencia se debe medir en un escenario con bajo uso de la CPU y con la CPU sobrecargada. Para cada escenario se deben tomar 60 muestras independientes.
2. Fluctuación de Fase (Jitter): es una información indirecta obtenida de varias medidas de latencia, que consiste en una variación aleatoria entre cada valor de latencia. Para calcular la fluctuación, se calcula la diferencia de tiempo entre dos medidas de latencia de interrupción consecutivos. Finalmente, se selecciona la diferencia más grande encontrada como la peor fluctuación de fase de este sistema.
3. Tiempo de respuesta de mayor deficiencia: se obtiene mediante el método propuesto por ISA que se discutió anteriormente realizando el análisis de la máxima frecuencia de las interrupciones que maneja el RTOS con fiabilidad. El peor caso de tiempo de respuesta es la inversa de la frecuencia máxima obtenida. El ensayo se realiza en un escenario de bajo uso de la CPU y en un escenario de CPU sobrecargado. Para cada escenario, se deben tomar 60 muestras independientes.

TABLA III
TABLA DE COMPARACIÓN DE LOS VALORES DE PRUEBAS DE RENDIMIENTO

Parámetros	SOTR A	SOTR B	SOTR C
Latencia			
Fluctuación de Fase			
Tiempo de respuesta de mayor deficiencia			

Luego de realizar las pruebas de rendimiento, se puede hacer uso de la Tabla III para tabular los valores para luego poder hacer las comparaciones correspondientes.

VII. CRITERIO DE SELECCIÓN DE SOTR PARA MISIONES EXPERIMENTALES DE NANOSATÉLITES EN ORGANIZACIONES DE INVESTIGACIÓN ESPACIAL DE POCA EXPERIENCIA

Para la formulación de este criterio de selección se está de acuerdo que no es una tarea trivial, y que por medio de pruebas de rendimiento en diferentes escenarios de carga del sistema [34] se pueden hacer comparaciones de diferentes SOTR en función de las necesidades de la misión, que en el caso de los nanosatélites, las restricciones de espacio y de potencia limitan el rendimiento del hardware a seleccionar, influyendo directamente en los recursos disponibles para un SOTR. Pero cuando el equipo de desarrollo tiene poca experiencia, como hace notar [33], el realizar una primera selección de SOTR, a los cuales realizar posteriormente las pruebas de rendimiento, resulta subjetivo y se pudiera hacer el descarte preliminar de algún SOTR que al final podría resultar en la opción mas ajustada a las necesidades de la misión. Por lo tanto, se plantea realizar un análisis preliminar de los diferentes sistemas operativos de tiempo real existentes en el mercado compatibles con las características de hardware de los nanosatélites, haciendo uso del criterio en [33], de manera de reducir el grupo primario de SOTR a los cuales finalmente se les realice las pruebas de rendimiento.

Para la utilización de este criterio se deberían seleccionar varias alternativas de arquitectura de hardware a utilizar, según los requerimientos de la misión.

En resumen, el criterio de selección propuesto consiste en los siguientes pasos:

1. Tomar los SOTR compatibles con las arquitecturas de hardware seleccionadas.
2. Utilizar el criterio de Selección de un SOTR basándose en información publicada.
3. Analizar los datos del punto número 2 de manera que se obtenga una primera aproximación de los SOTR más ajustados a la misión.
4. Aplicar el criterio de Selección de un SOTR basándose en pruebas de rendimiento del sistema.
5. Analizar los datos obtenidos en el punto número 4 para finalmente tomar la decisión del SOTR a utilizar

VIII. Conclusiones

La selección de un SOTR para una misión implementada con un nanosatélite requiere de un análisis detallado, tomando en cuenta los requerimientos de la misión y las limitaciones que resultan del uso de estas plataformas. Para realizar este análisis se pueden utilizar lo resultados de pruebas de rendimiento de los SOTR compatibles con las arquitecturas de hardware seleccionadas. También se puede hacer uso de la información publicada por el fabricante, tanto de las características del SOTR, como de reportes de uso en aplicaciones similares. Se hace la propuesta de un criterio de selección basado en las dos ideas mencionadas anteriormente, de forma de limitar el número de SOTR a los cuales realizar pruebas de rendimiento mediante una selección primara, tomando en consideración la posibilidad de la falta de experiencia en el equipo de desarrollo que pueda limitar la experticia para realizar un descarte de opciones bien sustentado.

Se sugiere como trabajo futuro aplicar este método en una aplicación real.

Referencias

[1] S. Harrison, P. Scott, and V. Zapien. (2012, Jun. 29) Nanosatellite Fabrication and Analysis. [Online]. Available: https://scholarcommons.scu.edu/bitstream/handle/11123/13/Nanosatellite%20Fabrication%20and%20Analysis_Thesis.pdf

[2] The World Radiocommunication Conference (Geneva). (2012) RESOLUTION 757 (WRC-12) Regulatory Aspects For Nanosatellites and Picosatellites. [Online]. Available: https://www.itu.int/dms_pub/itu-r/oth/0c/0a/R0C0A00000A0025PDFE.pdf

[3] J. Townsend, B. Palmintier, and E. Allison. (2000, Aug.) Effects of a Distributed Computing Architecture on the Emerald Nanosatellite Development Process. [Online]. Available: http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2085&context=smallsat

[4] J. Bouwmeester and J. Guo, "Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology," *Acta Astronautica 67*, vol. 67, no. 7-8, p. 854–862, 2010.

[5] (2014) CubeSat Design Specification. The CubeSat Program, Cal Poly SLO. San Luis Obispo, Cal. [Online]. Available: http://www.cubesat.org/images/developers/cds_rev13_final.pdf

[6] M. Aldea, "Planificación de tareas en sistemas operativos de tiempo real estricto para aplicaciones empotradas," Master's thesis, Univ. de Cantabria, Depto. de Electrónica y Comp., Santander, 2002.

[7] P. A. Laplante, "Real-Time Operating Systems," *in Real-Time Systems Design and Analysis*, 3rd ed. New Jersey: IEEE PRESS, 2004.

[8] D. C. C. Villamil, J. A. A. Mayorga, and F. A. D. González. (2013, Aug. 14) "Análisis de Rendimiento de dos Sistemas Operativos en Tiempo Real implementados en dos Arquitecturas de Hardware para la selección del Sistema Embebido que soportará el Software Command And Data Handling de la Misión Satelital Libertad 2," in 11th LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2013). [Online]. Available: http://www.laccei.org/LACCEI2013-Cancun/StudentPapers/SP031.pdf

[9] C. Becker. (2009) Sistemas operativos. [Online]. Available: http://ieee.eie.fceia.unr.edu.ar/PDF_RTOS.pdf

[10] F. Pastor. (2004) Sistemas operativos de tiempo real. [Online]. Available: http://ieee.eie.fceia.unr.edu.ar/PDF_RTOS.pdf

[11] J. Eickhoff, "Historic Introduction to Onboard Computers," *in Onboard Computers, Onboard Software and Satellite Operations*, 1st ed. Berlin Heidelberg: Springer, 2012.

[12] (2014) The RTEMS website. [Online]. Available: http://www.rtems.org/

[13] (2014) The RTEMS applications wiki. [Online]. Available: http://www.rtems.org/wiki/index.php/RTEMSApplications

[14] (2014) Wind River Products: VxWorks. [Online]. Available: http://www.rtems.org/wiki/index.php/RTEMSApplications

[15] (2002, Jul. 23) Wind River Newsroom: European Space Agency Deploys PROBA Satellite Using Wind River Technology. [Online]. Available: http://www.windriver.com/news/press/pr.html?ID=291

[16] (2012, Aug. 6) Wind River's VxWorks Powers Mars Science Laboratory Rover, Curiosity. *Virtual Strategy Magazine*. [Online]. Available: http://www.virtual-strategy.com/2012/08/06/wind-river%E2%80%99s-vxworks-powers-mars-science-laboratory-rover-curiosity#axzz3C6a63ryH

[17] S. Anthony. (2012, Aug. 6) Inside NASA's Curiosity: It's an Apple Airport Extreme... with wheels. [Online]. Available: http://www.extremetech.com/extreme/134041-inside-nasas-curiosity-its-an-apple-airport-extreme-with-wheels

[18] D. R. Williams. (2011, Nov. 7) Clementine project information. [Online]. Available: http://nssdc.gsfc.nasa.gov/planetary/clementine.html

[19] (2003, Jun. 8) Wind river newsroom: Wind river powers mars exploration rovers–continues legacy as technology provider for nasa's space exploration. [Online]. Available: http://www.windriver.com/news/press/pr.html?ID=314

[20] C. M. Cooke. (2012, Apr. 21) Implementation of a Real-Time Operating System on a Small Satellite Platform. presented at Space Grant Undergraduate Research Symposium. [Online]. Available: http://spacegrant.colorado.edu/COSGC_Projects/symposium/2012/08_Implementation%20of%20a%20Real-Time%20Operating%20System%20on%20a%20Small%20Satellite%20Platform.pdf

[21] (2014) The uClinux website. [Online]. Available: http://www.uclinux.org

[22] A. R. Deshpande. (2004, Mar. 26) Linux Kernel 2.6: the Future of Embedded Computing. [Online]. Available: http://www.linuxjournal.com/article/7477?page=0,0

[23] D. McCullough. (2003, May. 9) Deeply Embedded Linux. [Online]. Available: https://web.archive.org/web/20120322210901/http://www.ucdot.org/archive/deep/deeply-embedded-linux.html

[24] (2009, Sep.) AN3012 Application note: Getting started with uClinuxTM for STM32F10x high-density devices. [Online]. Available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00242717.pdf

[25] (2009, Sep.) AN3012 Application note: Getting started with uClinuxTM for STM32F10x high-density devices. [Online]. Available: http://www.uclinux.org/ports/

[26] J. Bhatti. (2009, Apr. 9) The University of Texas experience with μClinux, the TCM-BF537, and C++. presented at Calpoly Cubesat Workshop Developers 2008. [Online]. Available: http://mstl.atl.calpoly.edu/~bklofas/Presentations/DevelopersWorkshop2008/session2/5-uClinux-Jashan_Bhatti.pdf

[27] M. Schmidt, K. Ravandoor, O. Kurz, S. Busch, and K. Schilling. (2008, Jul. 6) Attitude Determination for the Pico-Satellite UWE-2, in Proc. 17th World Congr. The Int. Federation of Automatic Control. [Online]. Available: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/4264.pdf

[28] (2014) The freeRTOS website. [Online]. Available: http://www.freertos.org/

[29] K. Laizans, I. Sünter, K. Zalite, H. Kuuste, M. Valgur, K. Tarbe, V. Allik, G. Olentšenko, P. Laes, S. Lätt, and M. Noorma. (2014, May. 23) "Design of the fault tolerant command and data handling subsystem for ESTCube-1," in Proc. the Estonian Academy of Sciences 2014. [Online]. Available: http://www.eap.ee/public/proceedings_pdf/2014/issue_2S/Proc-2014-2S-222-231.pdf

[30] (2014) The PUMPKIN Salvo RTOS website. [Online]. Available: http://www.pumpkininc.com/

[31] B. Palmintier, C. Kitts, P. Stang, , and M. Swartwout. (2002) A Distributed Computing Architecture for Small Satellite and Multi-Spacecraft Missions, in Proc. 16th Annual AIAA/USU Conference on Small Satellites. [Online]. Available: http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1916&context=smallsat

[32] A. Barjatya, J. Nelsen, C. Swenson, and C. Fish. (2002) "A Low power Command and Control Module for Small Satellites," in Proc. 16th Annual AIAA/USU Conference on Small Satellites. [Online]. Available: http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1920&context=smallsat

[33] P. A. Laplante. Criteria and an objective approach to selecting commercial real-time operating systems based on published information. [Online]. Available: http://itech.fgcu.edu/faculty/zalewski/cop4931/files/laplantekernels.doc

[34] R. V. Aroca and G. Caurin. (2009, Jul. 22) "A Real Time Operating Systems (RTOS) Comparison," in Workshop de Sistemas Operacionais 2009. [Online]. Available: http://www.lisha.ufsc.br/wso/wso2009/papers/st04_03.pdf