Metodología para el desarrollo de una distribución GNU/Linux para computación paralela

Huriviades Calderón-Gómez[1], Miguel Vargas-Lombardo[2*]
GISES - Universidad Tecnológica de Panamá
huriviades.calderon@utp.ac.pa, miguel.vargas@utp.ac.pa

Resumen: *En este artículo se presentan las diferentes etapas de desarrollo para una distribución Linux adaptada a las necesidades de cómputo paralelo enfocada a un entorno académico y científico. Se hace especial énfasis en las metodologías de compilación cruzada y enjaulamiento root (Chroot Jail) utilizando código fuente de los paquetes GNU, permitiendo adecuar la distribución para que realice tareas de alto rendimiento como: proyectos MPI, minería de datos y de simulación enfocados en las ciencias básicas e ingenierías. Se desarrolla la distribución SpicaxOS bajo licenciamiento MIT con el propósito de demostrar las etapas que conlleva desarrollar una distribución GNU/Linux a la medida y las ventajas de ejecutar aplicaciones bajo el paradigma de la programación paralela en base a los resultados de esta investigación.*

Palabras claves: compilación cruzada, distribuciones GNU/Linux, programación paralela, SpicaxOS, construcción de distribuciones Linux, Kernel.

Title: Methodology for the development of a GNU/Linux distribution for parallel computing

Abstract: This article presents the different stages of development for a Linux distribution, adapted to the needs of parallel computing, focused on the academic and scientific environment. Special emphasis is placed on cross-compilation methodologies and Chroot Jail by using source code of the GNU packages, allowing the distribution to run high tasks performance processes, such as: MPI projects, data mining and simulation focused on the basic sciences and engineering. The SpicaxOS distribution is developed under MIT licensing with the purpose to demonstrate the stages involved in developing of customized GNU/Linux distribution and the advantages of running applications under the paradigm of parallel programming based on the results of the research.

Key words: cross compile, GNU/Linux distributions, parallel programming, SpicaxOS, Construction of Linux distributions, Kernel.

Tipo de artículo: original
Fecha de recepción: 24 de agosto de 2016
Fecha de aceptación: 7 de junio de 2017

1. Introducción

La popularidad de los sistemas multinúcleo ha hecho que la programación paralela llegue a ser una tarea de vital importancia para explotar de forma efectiva el hardware y aumentar el rendimiento de las aplicaciones [1].

Una distribución Linux es un conjunto de paquetes GNU desarrollada por grupos de personas, investigadores o empresas para satisfacer determinadas necesidades de sus usuarios [2].

En este trabajo describimos los pasos que seguimos para la construcción una distribución Linux que nosotros denominamos SpicaxOS, basándonos en el estudio de la documentación "Linux From Scratch" [3] y el comportamiento de otras distribuciones bajo el entorno de Linux [4], [5]. Se elabora el esquema de desarrollo para SpicaxOS, cuyo aporte principal es el soporte para la computación científica en paralelo.

La distribución SpicaxOS se fundamenta en seis características las cuales son:

- Un Kernel Linux modificado de la versión 4.4.13.
- Implementación del System V para administrar el proceso de arranque de la distribución.
- Integración del modelo ACL para la gestión de control de acceso de los usuarios [6].
- Implementación *uevent* de *udev*, para la gestión de los periféricos *hotplug* y sus firmwares.
- Interfaz de escritorio denomina KDE (bajo Xorg).
- Entorno *Eclipse Parallel Tools Platform* (PTP).

La distribución SpicaxOS vincula dentro del paradigma de la programación paralela importantes implementaciones como: Open MPI [7], [8]. Esta distribución se centra en fortalecer las bases de la programación paralela en el contexto científico, permitiendo abstraerse de configuraciones e instalaciones complejas facilitando la pronta puesta en marcha de proyectos y cursos avanzados de computación científica.

Las problemáticas de diversidad de repositorios (apt-get, yum, pacman y entre otros) traen consigo muchas versiones de un mismo paquete GNU (estable, inestable y modificados), provocando corrupción de la variable de entorno (path). Además, la mayoría de las distribuciones GNU/Linux traen consigo diversos paquetes innecesarios o entornos gráficos de mucho consumo (GNOME), causando pérdidas en el rendimiento al momento de compilar y ejecutar aplicaciones MPI [5]. Se propone y desarrolla SpicaxOS para ofrecer mayor compatibilidad a las aplicaciones paralelas y realizar configuraciones avanzadas de los periféricos. Además, en el contexto científico se incluyen librerías completas para álgebra lineal, multiproceso, procesamiento de imágenes, expresiones regulares y de pruebas unitarias para GCC [9], [10].

Actualmente, existen muchos puntos a considerar sobre la construcción de una distribución a la medida de los desarrollos científicos actuales [5], [11], [12]. Debido a esto, hemos decidido delimitarlo a cuatro secciones, para así explicar la importancia del tema y las razones que justifican su estudio:

En la sección 2, se describen las etapas de desarrollo de la distribución. En la sección 3 se implementa la distribución con un entorno MPI. Luego, en la sección 4, se presentan los resultados

en base a la interacción con algoritmos en paralelo. En la última sección compartimos nuestra conclusión y contribución al realizar esta investigación.

2. Discusión sobre esta investigación

La presente investigación surge de la necesidad de desarrollar una distribución compacta, flexible y segura dentro de las líneas de investigación (sistemas distribuidos) del grupo afiliado. Sin duda la necesidad principal entre los integrantes eran las siguientes interrogantes: ¿Cómo interactúa el kernel del sistema entre sus diversas capas?, ¿Desde dónde partimos en el desarrollo de la distribución GNU/Linux a nuestra medida?

Sobre las bases de las ideas expuestas, se analizaron los problemas más significativos al ejecutar computación paralela en distribuciones enfocada al usuario final:

- Diversidad de versiones de un mismo paquete GNU entre los repositorios.
- Modificaciones a la estructura del árbol de directorios, según el estándar *Filesystem Hierarchy Standard* (FHS) [13].
- Integración de paquetes secundarios o innecesarios para el enfoque del paralelismo.
- Poco soporte al paralelismo, algunas distribuciones traen consigo versiones de paquetes MPIs obsoletos. En consecuencia, se presentan vulnerabilidades de seguridad, o falta de paquetes complementarios para un mayor rendimiento [14]-[18].

La búsqueda de evidencias en países europeos, y en especial en España en las administraciones autonómicas han fomentado la creación de sus propias distribuciones de GNU/Linux para facilitar el acceso a servicios de TIC en el terreno de las gestiones públicas como en las dependencias municipales y en el educativo, y con ello no invertir en el pago de licencias para sistemas y aplicaciones propietarias. Por ejemplo: la Comunitat Valenciana, ha desarrollado Lliurex para facilitar el acceso a las tecnologías de la información y las comunicaciones dentro del ámbito educativo [19], [20].

Como complemento a esta investigación, mencionaremos las distribuciones GNU/Linux desarrolladas para específicas necesidades enmarcada al modelo de programación paralela [21]-[24]:

- PelicanHPC
- KestrelHPC
- ABC GNU/Linux
- Rocks

De las evidencias anteriores, señalaremos las ventajas de construir distribuciones GNU/Linux:

- Integración de código fuente bajo software libre y/o open Source.
- Flexibilidad de la distribución para adaptarse a las necesidades demandas por los desarrolladores.
- Adquisición de bajo costo.
- Independencia con proveedores y/o grupo de desarrolladores de las distribuciones principales de GNU/Linux.

Si bien es cierto al contar con la independencia de proveedores, las distribuciones construidas requieren de un grupo de desarrolladores especializados para el mantenimiento y soporte de la misma. Por ello se hace necesario un mínimo de tres desarrolladores enfocado en *Shebang* bajo entorno Linux.

3. Descripción de la distribución SpicaxOS

El kernel Linux es el componente central de una distribución GNU/Linux. Un kernel se encarga de manejar los recursos hardware como CPU, memoria, dispositivos *Hotplug*, los discos duros, y proporciona abstracciones que le dan a las aplicaciones una visión consistente de esos recursos [25]. SpicaxOS se construye en base a la versión 4.4.13 del kernel Linux (modificado) únicamente para 64 bits. Se utiliza *busybox* para iniciar la imagen de la distribución mediante RAM, cuyo paquete depende de un gestor de arranque del sistema (*Syslinux* o *Grub2*), paquetes GNU (Bash, Util-Linux, Make, GCC…) y entre otros. Todos estos componentes trabajan en conjunto desde la carga inicial de la imagen de kernel a través de *Initial Ram disk* (initrd) [26], [27] hasta la ejecución del entorno gráfico en KDE (ver figura 1).

Proponemos crear una partición dedicada bajo el formato "ext4" para la construcción de la distribución, en conjunto con los códigos fuentes de los paquetes GNU y en *scripts* desarrollados dentro de la investigación [3], [25], [28] en las siguientes subsecciones procedemos a explicar las diferentes etapas requeridas para el desarrollo de la distribución.

3.1 Construcción del sistema temporal mediante compilación cruzada (etapa 1)

El desarrollo de experimentos, bajo la arquitectura (x86-64) genera código ejecutable para una arquitectura diferente (x86, x86-64, ARM o PowerPC), cuyo concepto se denomina compilación cruzada. Mediante este método se requiere combinar elementos de recursos de hardware y de librerías que establezcan un ambiente óptimo para llevar a cabo esta tarea [29].

En el método de compilación cruzada, existen dos categorías:

- Huésped: es la distribución GNU/Linux (Gentoo, CentOS, Ubuntu, Mint, Slackware y entre otros), donde se realiza la compilación del sistema a construir (SpicaxOS). El cual utiliza sus propias librerías y enlace simbólico del directorio tools.

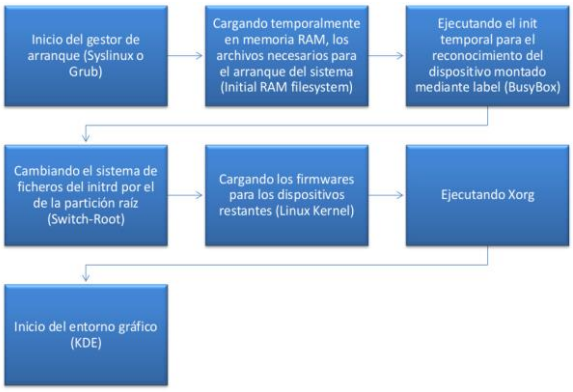

Figura 1. Diagrama del proceso "initrd" en spicaxos.

- Objetivo: es la partición de un disco (comúnmente separado del huésped), donde se ejecuta el sistema.

Durante esta etapa inicial, SpicaxOS es dependiente del sistema huésped; debido a que debe utilizar las librerías de huésped como base, y pueda compilarse a sí mismo, así puede ser adaptado a nuevas arquitecturas fácilmente (ver figura 2).

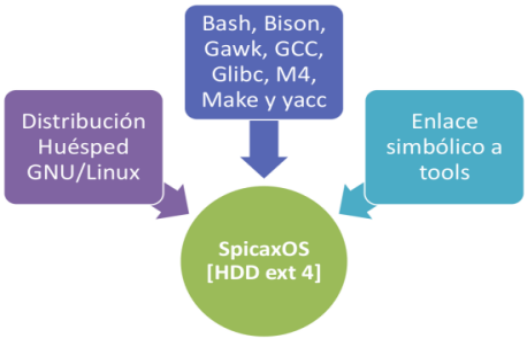

Figura 2. Diagrama de compilación cruzada para desarrollar SpicaxOS.

Además, para llevar a cabo esta metodología de compilación, se debe crear un nuevo usuario bajo un entorno controlado (definiendo reglas en el archivo de inicio del bash, ~/bashrc), y que no tenga privilegio (*non-root*); por consiguiente, sí no se cumplen estas condiciones, puede causar daños irreversibles al huésped y/o al sistema a construir.

3.2 Construcción del sistema persistente a partir del sistema temporal bajo chroot (etapa 2)

El chroot (*change root*) es un proceso utilizado en sistemas operativos tipo UNIX, cuya función es intercambiar el fichero de raíz y sus procesos actuales en ejecución (sistema huésped) a un directorio (*Chroot Jail*) determinado en el sistema temporal (SpicaxOS) [30]. Este cambio se hace comúnmente para el mantenimiento del sistema, instalación de un gestor de arranque o configuración del mismo Kernel.

En el caso de SpicaxOS, esta etapa se realiza los siguientes cambios:

- Creación de directorios esenciales (Bin, Boot, Etc, Usr, Tmp y entre otros directorios), según FHS.
- Creación de ficheros y enlaces simbólicos esenciales (desvinculación con el sistema huésped mediante el directorio tools) [31].
- Instalación de los paquetes esenciales de GNU (Bash, Coreutils, Ncurses, Make y entre otros), cuya compilación e instalación se realiza con el código fuente del mismo.
- Configuración, compilación e Instalación de las cabeceras del Kernel a SpicaxOS.

Antes de entrar al entorno chroot, se debe tener presente que solo estarán disponibles las librerías, enlace simbólico y ficheros del **sistema objetivo**; por consiguiente, todos los directorios dentro de Spicax OS, deben seguir el estándar FHS. Teniendo en cuenta, lo anteriormente expuesto se explicará un ejemplo de enjaulamiento:

Si en el sistema huésped se está utilizando **vim** (editor de texto) para editar ficheros de configuración de los paquetes GNU, y procedemos al enjaulamiento del sistema objetivo, no se podrá accesar al editor de texto del sistema huésped; debido a que el chroot ejecuta los procesos bajo un directorio raíz simulado; por consiguiente, de ser necesario el paquete vim, éste se deberá instalar dentro del sistema objetivo (SpicaxOS).

Con referencia al ejemplo planteado, el entorno chroot es vital en el desarrollo de una distribución GNU/Linux; debido a que el proceso ejecutado en el sistema objetivo, no pueda acceder a los archivos fuera de ese directorio (sistema huésped), esto se da para evitar daños en ambos sistemas (ver figura 3).

3.3 Instalando el kernel y sus módulos en el sistema persistente (etapa 3)

El kernel de Linux está escrito en el lenguaje de programación C, con una pequeña cantidad de lenguaje ensamblador [28]. Una de las características de este kernel, es su flexibilidad; debido a que cuenta con la integración de módulos, cuya cantidad de código puede superar a los siete millones de líneas [28].

Por otra parte, este kernel puede ser ejecutado en una gran gama de dispositivos, desde un microordenador hasta supercomputadoras.

Ninguna distribución GNU/Linux, proporciona un kernel exacto para satisfacer todas las necesidades de la mayoría de los usuarios. Por ello, se habilita la personalización del kernel (*kernel hacking*); por consiguiente, si un usuario adquiere un dispositivo nuevo, este podrá agregarlo al kernel y utilizarlo [3], [28].

Para la realización de una construcción y/o configuración del kernel, solo se requiere de tres paquetes:

- Compilador: se requiere del compilador de GCC, cuyo paquete en su mayoría son incluidos en las distribuciones GNU/Linux.
- Enlazador: no todo el trabajo es realizado por el compilador. También, es requerido un conjunto de herramientas adicionales (binutils), cuya función es de vincular y montar los códigos fuente.
- Make: es una herramienta que recorre el árbol de código fuentes del kernel para determinar los ficheros que necesitan ser compilados, y luego llama al compilador y otras herramientas de generación para hacer el trabajo en la construcción del kernel.

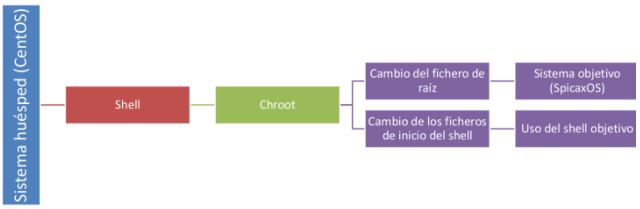

Figura 3. Diagrama del enjaulamiento root del sistema huésped para desarrollar a SpicaxOS.

Para llevar a cabo, la instalación del kernel y sus módulos en conjunto con los paquetes mencionados (GCC, Binutils y Make) dentro de la distribución, es necesario realizar las configuraciones de las listas de opciones del kernel mediante un gestor de menú [32]. Existen diferentes menús para realizar la configuración, teniendo en cuenta sí la distribución en su momento cuenta con una interfaz gráfica (GNOME, KDE o XFCE):

- Menú basado en texto: no requiere interfaz gráfica, pero se necesita responder a todas las opciones y no es posible acceder a las opciones en un orden distinto al que se muestran [32].
- Menú pseudográfico basado en Ncurses: mediante el modo texto se ejecuta un menú pseudográfico navegable, solamente por teclado, para seleccionar las opciones deseadas.
- Menú gráfico basado en Qt4 o GTK+: mediante el uso de una interfaz gráfica se invoca un formulario de opciones, utilizando el teclado o ratón, para seleccionar las opciones deseadas.

Antes de entrar en consideración, existen tres formas de configurar las secciones del kernel (seguridad, dispositivos, kernel hacking y entre otros), bajo las siguientes categorías:

- Built it (construido)
- *Module* (módulo)
- *Excluded* (excluido)

Por ejemplo, para habilitar automáticamente la gestión de los periféricos *hotplug*, requerido para soportar LiveUSB, es necesario "construir" a *udev* dentro del kernel. Otro ejemplo, los dispositivos de puerto paralelo (impresora) se puede habilitar manualmente por el usuario como "módulo" o simplemente "excluirlo" dentro del kernel.

3.4 Integrando el gestor de arranque en el sistema persistente (etapa 4)

Este es un programa sencillo, encargado exclusivamente de preparar todo lo que necesita el sistema operativo para iniciar [33]. Esta integración se ha dejado a decisión del usuario, cual gestor de arranque requiera habilitar en el sistema (Grub y Syslinux), bajo la categoría de etiquetas de disco mediante initramfs. Esto se utiliza para indicarle el directorio raíz estable al fichero **fstab**, cuya función es determinar dónde se encuentran los sistemas de archivos para ser montados por defecto, en qué orden, y que deben ser controladas (por errores de integridad) antes del montaje en el sistema [31].

3.5 Instalando la interfaz de escritorio de KDE en el sistema persistente (etapa 5)

Antes de habilitar KDE en la distribución, se requiere de la configuración del entorno del sistema X Windows (Xorg), cuya función es proporcionar una interfaz de Hardware-Desktop Environment para la manipulación y configuración de la infraestructura de ventanas gráficas de entrada y video [31], [34].

KDE es un entorno de escritorio basado en el framework Qt, cuya ventaja es la flexibilidad de configuración gráfica. Debido a esta ventaja, se elige KDE para ser instalado dentro de SpicaxOS en base a otras investigaciones [35], [36].

4. Implementación el entorno adecuado para programación en paralela

MPI (Message Passing Interface) es una interfaz de biblioteca de paso de mensajes especificado, se ocupa primordialmente de paso de mensajes de los datos para ser movido desde el espacio de direcciones de un proceso a otro, mediante el uso de operaciones de cooperativas en cada proceso [37].

Si bien es cierto, MPI no exige una determinada implementación del mismo; por consiguiente, es importante dar al desarrollador una colección de funciones para que éste diseñe su aplicación, sin que tenga necesariamente que conocer el hardware sobre el que se va a ejecutar, ni la forma en la que se han implementado las funciones que emplea [5], [37]. A causa de esto, hay una amplia variedad de implementaciones MPI. Algunas de las implementaciones más populares son: Open MPI, Open MP y entre otras.

Por otra parte, en la distribución SpicaxOS se integra a Open MPI, como compilador MPI estándar. Este proyecto, tiene la capacidad de combinar los conocimientos, tecnologías y recursos de toda la comunidad de computación de alto rendimiento (académica, de investigación y de la industria) con el fin de crear una mejor biblioteca MPI [38]. Para instalar Open MPI (v.1.8.4) en una distribución GNU/Linux se debe contar con algunas dependencias (ver tabla 1).

Tabla 1. Requisito de paquetes para Open MPI

Nombre	Versión	Dependencias
GCC	5.3.0	si
Boost	1.60.0	si
Libatomic_ops	7.4.2	no
OpenSSH	7.1	si

De igual manera, para un soporte óptimo de las colecciones de compiladores modulares, reutilizables y de enlazadores. Se agrega el paquete completo LLVM – v.3.7.1 en SpicaxOS, cuyas bibliotecas de bajo nivel ofrecen un optimizado compilador independiente, junto con el apoyo de generación de código para muchos CPUs populares e igualmente el soporte para algunos menos comunes [3].

5. Resultados y discusión

Las pruebas realizadas a SpicaxOS permitieron determinar la eficiencia de desarrollar una distribución GNU/Linux a medida, se utilizó como primer parámetro la métrica *Standard Build Unit* (SBUs) [39]. Esta métrica da un tiempo estimado de compilar e instalar un paquete (GCC, Boost u Open MPI) en la distribución, se utiliza la función *time* del *bash* (no se debe incluir el tiempo

para descomprimir el paquete) y mediante un archivo de registro "*log*" se determina el tiempo utilizado para instalar el paquete [39]. Por ejemplo, el paquete Boost tiene un estimado de 5.5 SBUs equivalente a 16 minutos aproximadamente en ejecución con una computadora moderna.

El resultado de este parámetro en SpicaxOS, utilizado un procesador Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz y memoria RAM de 8192MB fue de 5.35 SBUs; mientras que en Ubuntu fue de 5.37 SBUs, nuestra hipótesis basándose en la documentación de la comunidad LFS [3] era de 5.48 ~ 5.6 SBUs. Otro parámetro utilizado es el tiempo de finalizar una tarea MPI con 4 cores, se tuvo que configurar un entorno controlado bajo las siguientes condiciones:

- La distribución GNU/Linux debe contar con la versión 4.4.x del Kernel.
- El paquete GCC con una versión superior a la 5.2.
- Es obligatorio ejecutar las pruebas bajo el modo **prompt** del Shell y en el mismo equipo (Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz y memoria RAM de 8192MB).

Además, se desarrolló un algoritmo bajo la sintaxis MPI-C [38], cuya función es **paralelizar** una lectura de un archivo de texto plano, con una gran cantidad de datos numéricos, para concatenar una suma generalizada de esta lectura. A continuación, los resultados obtenidos (ver tabla 2).

Tabla 2. Resultado de las muestras

Distribución	Muestra 1. Tiempo en ejecución (segundos)	Muestra 2. Tiempo en ejecución (segundos)	Muestra 3. Tiempo en ejecución (segundos)	Muestra 4. Tiempo en ejecución (segundos)
Ubuntu 16.04	1.61673	1.22363	1.17181	1.20945
Fedora 23	1.23124	1.30247	1.16236	1.21751
Rocks 6.2	1.20500	1.21904	1.15053	1.20945
SpicaxOS	1.20568	1.19812	1.20669	1.23292

La hipótesis basándose en la ejecución del algoritmo en el clúster de GISES fue de 0.45305 segundos; por consiguiente, se determinó un rango de aceptación de 1.0 ~ 1.5 segundos, todas las distribuciones, incluyendo a SpicaxOS, se encuentra dentro del rango de aceptación.

Un hallazgo importante obtenido fue que a medida que se integraba el soporte a varios dispositivos como: sonido, red y tarjeta de video bajo la categoría de construcción en el kernel, la carga inicial de la distribución sufría una ralentización. Para evitar esto, las distribuciones GNU/Linux utilizan módulos dentro del kernel, cuyo llamado se realiza manualmente por el usuario; por consiguiente, se reduce considerablemente el peso y la carga en el Kernel. Por ello, se optó que los dispositivos adicionales (bluetooth y tarjetas inalámbricas), se integren al Kernel como módulos [3], [5], [28]. Adicionalmente, se optó Busybox en conjunto con un *script bash* para la creación de LiveUSB.

Además, se realizó la integración de un pequeño gestor de paquetes llamado *spicax*, al igual que la distribución construida, con la capacidad de gestionar paquetes y/o módulos propios al sistema.

Otra tarea prioritaria del gestor de paquete *spicax* es realizar las configuraciones PATHs para el funcionamiento de los paquetes MPIs instalados en el sistema.

Como complemento a los resultados de esta investigación, compartiremos el url (http://www.gisespanama.org/distro.html) que aloja la versión beta de SpicaxOS.

6. Conclusiones

Con el desarrollo del presente estudio se logró completar el objetivo propuesto, obtener una distribución estable y modificable según las necesidades del entorno científico y académico con soporte a la computación en paralelo.

Al mismo tiempo, se logró demostrar todos los pasos involucrados en la construcción de una distribución GNU/Linux y de permite a los usuarios desarrollar sus módulos, repositorios o utilizar a SpicaxOS, como sistema huésped para crear sus variantes o propias distribuciones.

Como trabajos futuros se tiene contemplado el soporte para sistemas embebidos y de clúster. Además, sería de interés implementar versiones más recientes del kernel (4.9.9), que incluye el soporte a OrangeFS y USB 3.1 [40].

7. Agradecimiento

Este trabajo fue apoyado en parte por el grupo de investigación en salud electrónica y supercomputación (GISES) de la Universidad Tecnológica de Panamá y al programa Sistema Nacional de Investigación (SNI) de la SENACYT, el cual uno de los dos autores es miembro.

Referencias

[1] J. González, "UPCBLAS: a numerical library for unified parallel C with architecture-aware optimizations," La Coruña, España, 2012.

[2] J. Esteve y R. Suppi, Administración de Sistemas GNU/Linux, 4 ed., 2014.

[3] G. Beekmans y B. Dubbs, Linux From Scratch, 2016, p. 364.

[4] W. J. Shotts, The Linux Command Line: A Complete Introduction, 1 ed., No Starch Press, 2012, p. 480.

[5] H. Calderón y L. Dominguez, "Desarrollo De Una Distribución Linux Para La Implementación De Aplicaciones Paralelas Científicas De Código Abierto," Panamá, 2015.

[6] I. Arenaza, "Proyecto escomposlinux: Uso de Listas de Control de Acceso (ACLs) en Linux," 28 Marzo 2004. [En línea]. Available: http://webcache.googleusercontent.com/search?q=cache:http://www.escomposlinux.org/iarenaza/articulo-acls/acls-linux-samba.html. [Último acceso: 23 Agosto 2016].

[7] E. Fernández, "mpi-start and MPI-Utils," CSIC, 2013.

[8] S. Alexander y Y. Artem, The Parallel Universe: 20 Years of the MPI Standard, Intel, 2014.

[9] B. Schäling, The Boost C++ Libraries, 2 ed., 2014.

[10] K. Ahnert y M. Mulansky, "Odeint – Solving ordinary differential equations in C++," de *Numerical Analysis and applied mathematics: ICNAAM 2011*, 2011.

[11] J. Hall, "crear distribuciones a medida," *Linux magazine,* p. 90, 2013.

[12] D. Herrera, S. Lambert, Y. Fernández, M. Albalat, J. Castillo, H. Baranda y R. Mejías, "Estado actual de los sistemas de construcción de paquetes en diferentes distribuciones de GNU/Linux," *REVISTA CUBANA DE CIENCIAS INFORMÁTICAS,* vol. 6, nº 2, p. 17, 2012.

[13] Linux Foundation, "Linux Foundation: FHS," 3 Junio 2015. [En línea]. Available: https://wiki.linuxfoundation.org/en/FHS. [Último acceso: 31 Julio 2016].

[14] Ubuntu, "Change log for openmpi package in Ubuntu," 11 Febrero 2017. [En línea]. Available: https://launchpad.net/ubuntu/+source/openmpi/+changelog. [Último acceso: 27 Abril 2017].

[15] D. Jansen, "Red Hat Buzilla: Bug 1301533 - mpif90 fails to locate mpi.mod," 25 Enero 2016. [En línea]. Available: https://bugzilla.redhat.com/show_bug.cgi?id=1301533. [Último acceso: 27 Abril 2017].

[16] Open MPI, "Open MPI: Version Timeline," 21 Marzo 2017. [En línea]. Available: https://www.open-mpi.org/software/ompi/versions/timeline.php. [Último acceso: 27 Abril 2017].

[17] Open MPI, "Open MPI: Version Number Methodology," 27 Julio 2016. [En línea]. Available: https://www.open-mpi.org/software/ompi/versions/. [Último acceso: 27 Abril 2017].

[18] Open MPI, "FAQ: Running MPI jobs - 2. What ABI guarantees does Open MPI provide?," 2016 Julio 21. [En línea]. Available: https://www.open-mpi.org/faq/?category=running. [Último acceso: 27 Abril 2017].

[19] C. V. Albalat, "Musicogrames mitjançant les TIC del sistema operatiu LliureX," Repositori Universitat Jaume I, Valenciana, 2015.

[20] Comunitat Valenciana, "Proyecto LliureX," Comunitat Valenciana, [En línea]. Available: http://mestreacasa.gva.es/web/lliurex/proyecto. [Último acceso: 26 Abril 2017].

[21] PelicanHPC, "PelicanHPC: Features," Michael Creel, [En línea]. Available: http://www.pelicanhpc.org/index.html. [Último acceso: 27 Abril 2017].

[22] KestrelHPC, "What is KestrelHPC?," [En línea]. Available: http://kestrelcluster.github.io/index.html#About. [Último acceso: 17 Abril 2017].

[23] I. Castanos, "ABC GNU/Linux," [En línea]. Available: http://www.ehu.eus/AC/ABC.htm. [Último acceso: 27 Abril 2017].

[24] Rocks, "Rocks: About," University of California, [En línea]. Available: http://www.rocksclusters.org/wordpress/?page_id=48. [Último acceso: 27 Abril 2017].

[25] D. Bovet y M. Cesati, Understanding the Linux Kernel, 3 ed., O'REILLY, 2005, p. 944.

[26] W. Almesberger, "Booting Linux: The History and the Future," de *Proceedings of the Ottawa Linux Symposium*, 2000.

[27] J. Rajasekhar y R. Venkatesh, "Efficient initial RAM disk creation". Estados Unidos Patente US 8468334 B1, 18 Junio 2013.

[28] G. Kroah-Hartman, Linux Kernel in a Nutshell, O'Reilly, 2006, p. 202.

[29] E. López, "GNU/Linux embebido: Compilación cruzada," 2013. [En línea]. Available: http://linuxemb.wikidot.com/tesis-c3. [Último acceso: 30 Julio 2016].

[30] L. McFearin, Encyclopedia of Cryptography and Security: Chroot Jail, Springer, 2011.

[31] D. Morril, Configuración de sistemas Linux, España: Anaya Multimedia, 2002.

[32] Gentoo, "Núcleo/Configuración," 24 Noviembre 2015. [En línea]. Available: https://wiki.gentoo.org/wiki/Kernel/Configuration/es. [Último acceso: 23 Agosto 2016].

[33] P. Calleja, I. Miguel y J. Carmona, "Linux embebido en FPGA para sistemas de monitoreo industrial," *Revista Cubana de Ciencias Informáticas,* vol. 7, nº 1, p. 11, 2013.

[34] R. Sánchez, "Estado del arte de los gestores de ventanas en GNU/Linux," Universitat Oberta de Catalunya, España, 2013.

[35] T. Drilling, Justo Antes de las 5: compilación de software KDE 4.10, Linux magazine, 2013, pp. 24-27.

[36] R. Suvorov, M. Nagappan, A. Hassan, Y. Zou y B. Adams, "An empirical study of build system migrations in practice: Case studies on KDE and the Linux kernel," de *Software Maintenance (ICSM), 2012 28th IEEE International Conference on* , Trento, 2012.

[37] B. Barney, "Message Passing Interface (MPI)," Lawrence Livermore National Laboratory, 30 Junio 2016. [En línea]. Available: https://computing.llnl.gov/tutorials/mpi/. [Último acceso: 31 Julio 2016].

[38] Open MPI, "Open MPI v.1.8.8 documentation," Open MPI, 27 julio 2016. [En línea]. Available: https://www.open-mpi.org/doc/v1.8/. [Último acceso: 1 agosto 2016].

[39] B. Dubbs, "About SBUs," Linux From Scratch, 1 Septiembre 2014. [En línea]. Available: http://www.linuxfromscratch.org/~bdubbs/about.html. [Último acceso: 23 Agosto 2016].

[40] Linux Foundation, "The Linux Kernel Archives," 16 Agosto 2016. [En línea]. Available: https://cdn.kernel.org/pub/linux/kernel/v4.x/ChangeLog-4.6.7. [Último acceso: 17 Agosto 2016].