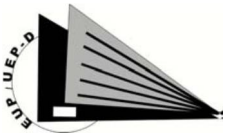
Universidad del País Vasco
Euskal Herriko Unibertsitatea

MÁSTER UNIVERSITARIO EN

INGENIERÍA DE SISTEMAS EMPOTRADOS



Escuela Politécnica

Donostia-San Sebastián

Optimización del software empotrado de los equipos HMI y sDIAG del TCMS COSMOS

Alumno: José María Romero (txema.romero@gmail.com)

Director: Izaskun Etxeberria

Instructor en la empresa: Oihana Azpitarte

Proyecto Fin de Máster, Julio de 2016

Resumen

Los sistemas en tiempo-real (por ejemplo, los que se basan en Linux en tiempo-real) se enfrentan a los mismos retos de desarrollo que el software tradicional: reducción de los presupuestos y marcos temporales más cortos. Pero además, el desarrollo de dichos sistemas puede complicarse debido a requisitos adicionales como seguridad, fiabilidad, uso mínimo de los recursos y, en algunos casos, cumplimiento de estrictas normas industriales, tal y como ocurre en el sector ferroviario. En este sector, los sistemas de control y monitorización de tren (TCMS) son un método innovador a la hora de operarlos. COSMOS es un sistema de control y monitorización de tren de la empresa CAF Power & Automation (filial tecnológica del grupo CAF). Esta empresa está interesada en añadir características de tiempo-real a dos de los equipos más importantes del TCMS COSMOS: HMI (interfaz hombre-máquina) y sDIAG (unidad central de diagnóstico del tren). Junto a ello, el proyecto plantea también la posibilidad de optimizar el sistema operativo de ambos equipos. Por último, todas estas mejoras potenciales se validan en banco de ensayos antes de su puesta en producción.

Palabras clave: ferroviario, TCMS, Linux en tiempo-real, HMI, sDIAG, PREEMPT_RT, validación.

Laburpena

Denbora errealeko sistemek (esaterako, denbora errealeko Linux-ean oinarritzen direnek) ohiko software-ak gainditu beharreko erronka berberak gainditu behar dituzte: aurrekontuak gero eta gehiago murrizten dira eta garapen-denbora gero eta motzagoa da. Baina horrez gain, halako sistemak garatzea konplikatua izan daiteke hainbat baldintza direla eta, hala nola segurtasuna, fidagarritasuna, baliabideen erabilera minimoa eta abar. Zenbait kasutan, gainera, industria arloko arau zorrotzak bete behar dira, eta hala gertatzen da trenbide arloko industrian. Arlo horretan, trenetan eragiteko metodo berritzailetzat hartzen dira egungo kontrol- eta monitorizazio-sistemak (TCMS sistemak ingelesez). COSMOS sistema TCMS sistema bat da, CAF Power & Automation enpresak garatutakoa (CAF taldearen eskumendeko enpresa teknologikoa). Enpresak denbora errealeko ezaugarriak gehitu nahi dizkie COSMOS sistemaren bi osagai garrantzitsuenei: HMI (giza-makina interfazea) eta sDIAG (trenaren diagnostiko-unitate zentrala) osagaiei. Horrekin batera, planteatzen duen proiektuak bi ekipo horien sistema eragilea optimizatzeko aukera ematen du. Balizko hobekuntza horiek guztiak saiakuntza-banku batean egiaztatu behar dira produkziora eraman baino lehen.

Gako-hitzak: trenbide-sistema, TCMS, denbora errealeko Linux-a, HMI, sDIAG, PREEMPT_RT, balidazioa.

Abstract

Real-time systems (e.g., based on Real-Time Linux) face the same development challenges as traditional software: shrinking budgets and shorter timeframes. In addition, these systems can be even more difficult to successfully develop due to additional requirements for safety, reliability, minimal resource use, and, in some cases, the need to support rigorous industry, e.g., railway standards. In this sector, train control and monitoring systems (TCMS) are an innovative method of operating trains. COSMOS is a train control and monitoring system from CAF Power & Automation (technological subsidiary of the CAF group). This company is interested in providing features of real-time to two important parts of TCMS COSMOS: HMI (human-machine interface) and the sDIAG (central train diagnostic unit). As well, the project raises the possibility to optimize the operating systems of these equipments. Finally, all these potential improvements are validated in test bench before put into the series.

Keywords: railway, TCMS, Real-Time Linux, HMI, sDIAG, PREEMPT_RT, validation.

Índice

1. Introducción

Este documento corresponde a la memoria del Proyecto Fin de Máster (PFM) del Máster Universitario en Ingeniería de Sistemas Empotrados, impartido en la Universidad del País Vasco/*Euskal Herriko Unibertsitatea*. Este proyecto lo ha realizado José María Romero en CAF Power & Automation y bajo el título de Optimización del software empotrado de los equipos HMI y sDIAG del TCMS COSMOS.

En esta introducción, primeramente se describe el contexto del proyecto, siguiendo con los objetivos de éste y la planificación que se ha llevado a cabo. Finalmente, se describe el contenido del resto del documento.

1.1. Contexto del proyecto

Los sistemas de control y monitorización de tren (TCMS) de abordo son ampliamente utilizados en todo el mundo, pero todavía se consideran como un medio innovador de operar trenes. Como cualquier otra innovación (está registrada como patente US 8,712,611 B2 [EEUU 14]), las expectativas sobre su potencial varían según un usuario u otro, y se asocian fuertemente al rápido desarrollo de los ordenadores y la capacidad del procesamiento de los datos.

En los inicios de la adopción de la Electrónica embarcada, su objetivo era proporcionar una mejor comprensión de los fallos del tren. La responsabilidad de monitorizar que tiene la Electrónica embarcada se reconoce actualmente aunque es difícil de medir en términos de retorno de inversión [Neil 14].

La Electrónica embarcada probablemente evolucionará acorde a lo siguiente:

- La sustitución de partes obsoletas y ensamblajes por nuevas partes y ensamblajes que realizan al menos la misma función, pero con coste reducido y mayores prestaciones.
- La inclusión de nuevas y valiosas y funciones tecnológicas extendiendo el rango de servicios proporcionados por sistemas ya existentes, en particular con vistas a mejorar el rango de servicios ofrecidos a pasajeros.
- Introducción de un TCMS, y consecuente incremento en la demanda de sistemas TCMS para nuevos y renovados sistemas.

Por otro lado, muchos trenes de pasajeros modernos también hacen uso extensivo de sistemas de monitorización de vídeo embarcado. Típicamente, estos sistemas de vídeo-vigilancia pueden incluir:

- Cámaras CCTV en sala de pasajeros.
- Plataforma para tren CCTV.
- Vigilancia de la alarma de emergencia de pasajeros.
- Sistemas CCTV en interior de cabina.
- Dispositivos de grabación de vídeo.

La mayoría de los proveedores de subsistemas electrónicos modernos también incluyen monitorización y sistemas de diagnóstico. Típicamente, los subsistemas que tienen tales características pueden incluir:

- Equipamiento ATC (en Inglés, *Automatic Train Control*).
- Sistemas electrónicos de propulsión.
- Equipamiento de control de puerta electrónico.
- Módulos de sistemas de frenado.
- Sistemas audio/vídeo.

En general, tales subsistemas notifican un informe resumen de diagnóstico e información de estado directamente al TCMS de a bordo hacia la pantalla del conductor del tren o para mantenimiento técnico. Sin embargo, muchos de estos subsistemas tienen la capacidad de comunicarse directamente a través de un enlace serie de diagnóstico (RS-232, USB, IP-

Ethernet, etc.) a un PC de escritorio u otro dispositivo. Esta característica permite al ingeniero de mantenimiento o especialista acceder a la información que se puede usar para ajuste de parámetros y/o una mayor profundidad en el diagnóstico de fallos.

1.1.1. TCMS COSMOS de CAF Power & Automation

Según la página corporativa de CAF Power & Automation [CAFPOWER 16], COSMOS es el sistema de control y monitorización de tren (TCMS) de la empresa. Está basado en el estándar de comunicaciones de tren TCN. Se trata de un sistema modular y distribuido a lo largo de todo el tren, haciendo posible la incorporación de equipos y funcionalidades al sistema de modo que éstos sean prácticamente "*plug and play*". COSMOS es un sistema robusto, destinado a cubrir un gran abanico de necesidades, y flexible, capaz de adaptarse a cada aplicación o proyecto particular.

Además de la robustez, modularidad y flexibilidad, el sistema COSMOS permite el acceso sencillo y uniforme a toda la información disponible en el tren.

Con ello:

- Se reduce al mínimo la complejidad de la puesta en marcha así como las labores de mantenimiento del tren.
- La configuración de los elementos que componen el sistema COSMOS se realiza mediante sencillas herramientas de configuración.
- Ofrece una mayor facilidad de montaje y menor necesidad de cableado en el tren.

Dos de los equipos más relevantes dentro del TCMS COSMOS y con los que se va a trabajar a lo largo de este proyecto son: el equipo HMI y el equipo sDIAG. El HMI es el equipo en el que ocurre la comunicación entre el usuario y la máquina, desde donde el maquinista puede controlar y monitorizar los distintos parámetros del sistema. Por otro lado, el equipo sDIAG se encarga de gestionar el diagnóstico avanzado del sistema. Por ejemplo, proporcionando capacidad de configuración más amplia, mediante funciones de configuración del sistema desde tierra, o funciones de descarga de registros desde tierra. El software (empotrado) de estos equipos se basa en Linux. Hasta la actualidad, CAF Power & Automation ha partido de una imagen base común para ambos equipos, sobre la que se montan manualmente funciones y servicios correspondientes a cada equipo.

Este proyecto propone mejoras sobre el sistema operativo de éstos, que van desde modificaciones del kernel, para dotarlo de características de tiempo-real, hasta modificaciones a nivel de funciones y servicios de sistema.

1.2. Objetivos del proyecto

El objetivo inicial propuesto por el gestor de producto TCMS de la empresa para este proyecto parte de lo siguiente:

Para la unidad central de diagnóstico del tren (sDIAG), se utiliza actualmente una versión de Linux sin capacidad de tiempo-real. Si bien no hay requisitos de tiempo-real estricto, su capacidad de registro debe cumplir estadísticamente con desvíos temporales controlados. Se trata en el proyecto de validar la introducción de tres versiones de kernel Linux (Linux vanilla, Linux *lowlatency* y Linux PREEMPT_RT), comparando el impacto en la latencia de respuesta a una interrupción y el rendimiento general del sistema. Se validará en paralelo con un HMI para ver su impacto en dicho equipo.

Las tareas eran:

1. Preparación de instalaciones Linux con kernels de diferente capacidad de expulsión.
2. Preparación de un entorno de pruebas con 6 equipos (3 sDIAG, 3 HMI), creación de una estrategia de validación.
3. Automatización de las pruebas y aplicabilidad a nuevos equipos futuros.
4. Informe de pruebas y decisión de introducción en la serie.

Las tareas finales han ido más allá de este alcance inicial, siendo las siguientes:

1. Preparación de instalaciones Linux con kernels de diferente capacidad de expulsión: vanilla, *lowlatency*, PREEMPT_RT y Xenomai.
2. Preparación de un entorno de pruebas con 2 equipos (1 sDIAG, 1 HMI).
3. Comparativa de test de las distintas soluciones: vanilla, *lowlatency*, PREEMPT_RT y Xenomai. Sobre PC, sDIAG y HMI.
4. Validación de PREEMPT_RT sobre sDIAG y HMI.
5. Verificar resultados obtenidos sobre procesos del HMI.
6. Realizar mejoras en las aplicaciones de los equipos HMI y sDIAG.
7. Informe de pruebas y decisión de introducción en la serie.
8. Aprender a programar aplicaciones de tiempo-real sobre PREEMPT_RT y Xenomai.
9. Establecer relaciones con la comunidad científica de sistemas operativos de tiempo-real.
10. Realizar una investigación básica. Y con el fin de incrementar sobre todo el conocimiento de los principios fundamentales de tiempo-real y Linux en tiempo-real, del Departamento de Software de CAF Power & Automation.

Tabla 1. Tareas del proyecto

1.3. Planificación

El proyecto se inició el Lunes, 11 de Enero de 2016 y ha finalizado el Jueves, 30 de Junio. La memoria se presentó antes del Viernes, 1 de Julio.

El desarrollo del proyecto ha seguido el diagrama de Gantt que se muestra en la siguiente tabla.

11-29 Enero	1-12 Febrero	15-29 Febrero	1-18 Marzo	21-31 Marzo	1-15 Abril	18-29 Abril
Estudio del estado del arte	Estudio del estado del arte	Preparación de instalaciones Linux	Preparación de instalaciones Linux	Preparación de instalaciones Linux	Comparativa de test	Verificar resultados sobre procesos del HMI
	Redactar memoria	Redactar memoria	Redactar memoria	Redactar memoria	Redactar memoria	Redactar memoria

2-13 Mayo	16-31 Mayo	1-17 Junio	20-30 Junio
Validación de PREEMPT_RT sobre sDIAG y HMI	Nuevas mejoras	Nuevas mejoras	Nuevas mejoras
Redactar memoria	Redactar memoria	Redactar memoria	Decisión de introducción en la serie
	Investigación básica	Investigación básica	Redactar memoria
	Aprender a programar aplicaciones de tiempo-real		Investigación básica

Tabla 2. Diagrama de Gantt

1.4. Contenido del documento

En la sección 2, se hace una descripción de la tecnología utilizada que se centra sobre todo en los equipos HMI y sDIAG. La sección 3, presenta una revisión del estado del arte en temas de sistemas operativos de tiempo-real (SOTR). En ella, se explican brevemente las distintas soluciones que existen actualmente para dotar a un sistema operativo de características de tiempo-real. Tras ello, la sección 4, corresponde a la implementación de las soluciones elegidas para dotar el kernel de Linux de capacidad de tiempo-real. Y presenta una comparativa de estas soluciones con el objetivo de seleccionar la más conveniente. Las secciones 5 y 6, recogen la validación de las distintas soluciones llevadas a cabo en los equipos HMI y sDIAG, respectivamente. En la sección 7, se mencionan las mejoras adicionales realizadas en el sistema operativo para los equipos HMI y sDIAG. La sección 8, presenta las conclusiones obtenidas a lo largo del proyecto y líneas futuras que le podrían dar continuidad. La sección 9, aporta un glosario de términos. La sección 10, contiene la bibliografía utilizada. Por último, se han reservado unos anexos para profundizar en el contenido de este trabajo.

2. Descripción de la tecnología utilizada

El estudio del equipamiento a utilizar es importante en cualquier proyecto. En esta sección se hará una breve introducción de los principales componentes con los que se ha trabajado, empezando con la definición del concepto de interfaz hombre-máquina, y siguiendo por la unidad central de diagnóstico sDIAG que se nos ha proporcionado para la realización de este proyecto. También se indicará cómo se debe preparar el equipo para comenzar a trabajar.

2.1. HMI (interfaz hombre-máquina / interfaz de usuario)

El HMI es el equipo que permite al usuario interactuar con el sistema TCMS del tren a través de su pantalla táctil. El usuario solicita una información o pide realizar una acción a la máquina (por ejemplo, el tren) y obtiene una respuesta de la misma en función de la solicitud y de los resultados. Por ejemplo, se puede solicitar la visualización del estado de todos los equipos del tren, se puede accionar la apertura de puertas, etc. Estas interfaces deben cumplir una serie de requisitos que les doten de una cierta ergonomía y versatilidad para los usuarios. La plataforma hardware utilizada se trata de un modelo BC3912 de GERSYS [BC3912 16], como el de la siguiente figura.

Figura 1. HMI BC3912 de GERSYS
Fuente: [BC3912 16]

El HMI alberga el sistema operativo en una CFast que es un tipo de dispositivo de almacenamiento de datos y memoria no volátil. Sobre este sistema operativo se incluyen todos los procesos y servicios necesarios para el funcionamiento adecuado del HMI.

Figura 2. CFast usada para el HMI BC3912 de GERSYS

Fuente: [SWISSBIT 16]

El HMI está diseñado para aplicaciones de intenso procesamiento de gráficos y vídeo digital. El dispositivo cumple los estándares ferroviarios y funciona de forma fiable en condiciones atmosféricas extremas.

A continuación, se enumeran y explican algunos de los procesos desarrollados por CAF Power & Automation que son necesarios para el correcto funcionamiento del HMI.

- Servidor de comunicaciones XCommSvr

Este proceso hace posible la comunicación de diferentes buses (MVB, ETH, etc.). Coge toda la información que le llega y almacena en una base de datos interna que se ubica en una memoria compartida para que distintos procesos puedan acceder a la misma y modificar u obtener los datos correspondientes a cada uno. El *XCommSrv* también ofrece la opción de guardar determinadas variables en un dispositivo no volátil para que éstos no se pierdan si se deja de recibir alimentación. Estas variables son las llamadas variables persistentes y se configuran cuáles son en cada proyecto.

- CduDB

Este proceso, ofrece una interfaz para que aplicaciones externas al HMI se puedan conectar a la base de datos de variables del HMI (VarDB) y puedan leer o escribir estas variables. Algunas herramientas externas, por ejemplo, TiMon, utilizan este proceso para consultar el estado de las variables de una forma *on-line*. TiMon es un programa que sirve para obtener o modificar el valor de las variables que maneja el HMI de forma remota y desde un PC [TiMon 09]. Esta herramienta necesita que el proceso CduDB se esté ejecutando correctamente para poder recibir la información del HMI. La siguiente figura muestra la pantalla principal del programa TiMon:

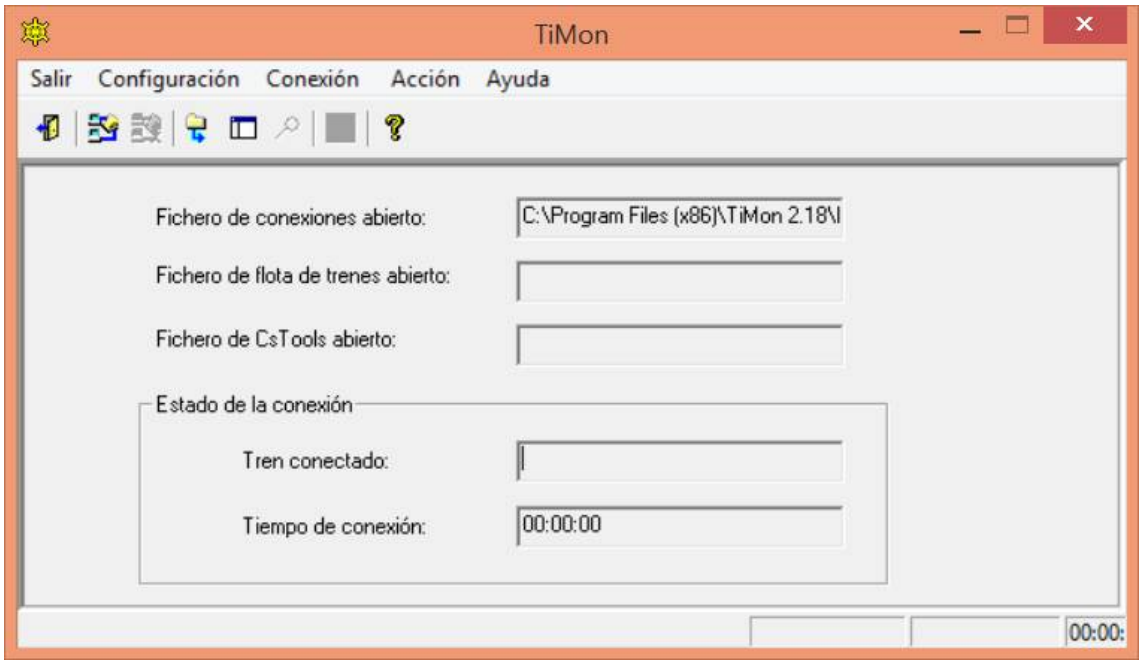


Figura 3. Pantalla principal de TiMon

o Cómo generar fichero de configuración para TiMon Embedded

Por otro lado, TiMon Embedded se encarga de muestrear variables internamente dentro del HMI. Esto se hace de la siguiente manera:

En Configuración > Abrir fichero CsTools. Elegimos fichero *.cdb. Por ejemplo, `CI4_NS.cdb`. Posteriormente, Acción > Monitorizar.

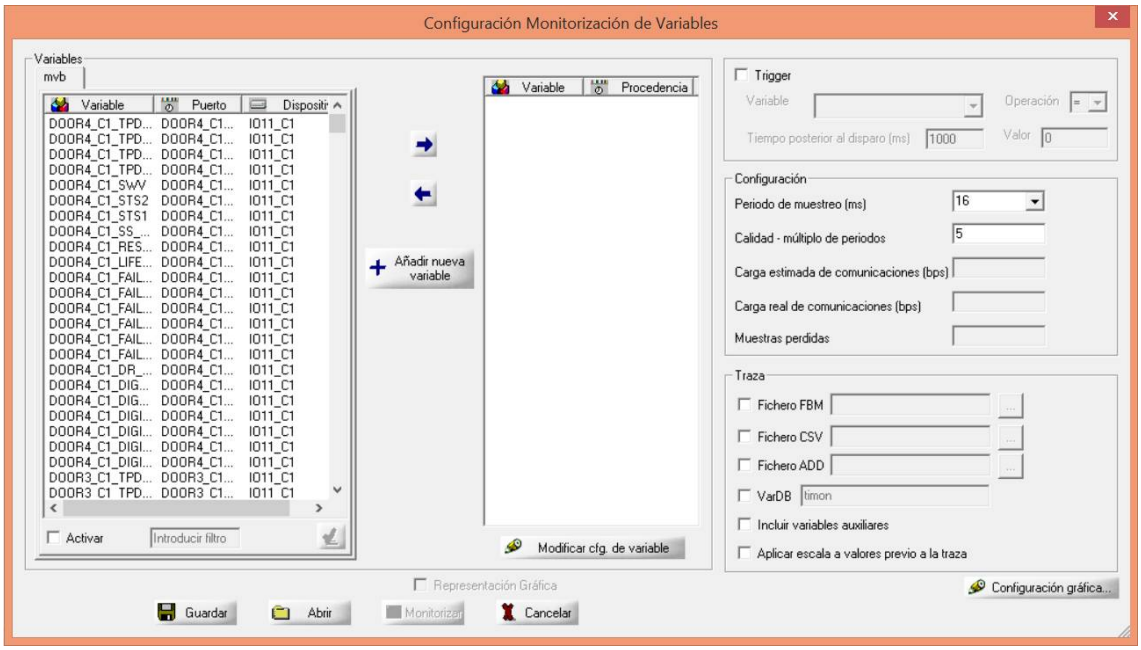


Figura 4. Pantalla principal de monitorización de variables

Elegimos las variables (de la izquierda) a monitorizar con TiMon Embedded.

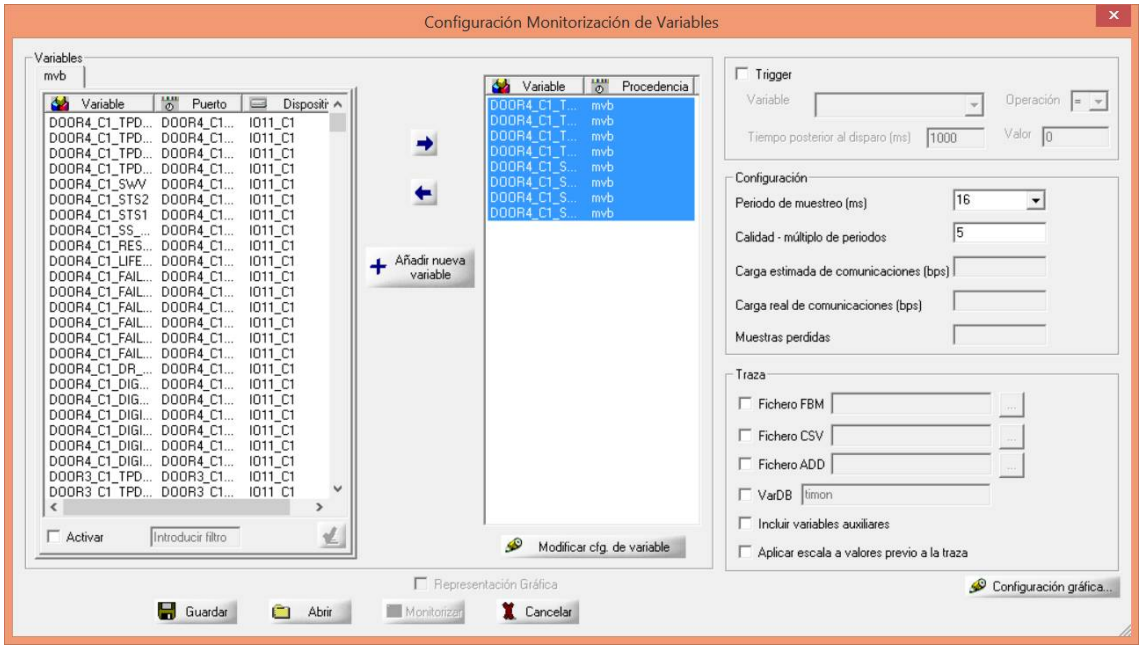


Figura 5. Variables a monitorizar (seleccionadas) con TiMon Embedded

Del mismo modo, podemos configurar el periodo de muestreo (ms). Por ejemplo, 16. Por último, se guarda con nombre "`timon_embedded.xml`".

2.2. sDIAG

El sDIAG es el equipo del sistema TCMS que se encarga del diagnóstico de los distintos equipos del tren, y que genera unos registros de datos que se pueden descargar de forma remota desde el puesto de mando en tierra.

El diagnóstico es la tarea de identificar, evaluar y registrar una circunstancia que se debe vigilar a través de variables de proceso, con chequeo periódico de reglas aritmético-lógicas definidas para la generación de alarmas y eventos. Los eventos se notifican a varios servicios y, entre otros, se guardan en un registro de eventos, junto con una selección de variables asociadas con el evento. Estas operaciones para configurar el diagnóstico, cargar la configuración, descargar los registros y analizar los registros se pueden llevar a cabo mediante herramientas PC [Zuriarrain 15].

El sDIAG es un equipo en desarrollo y actualmente se están definiendo los procesos que contendrá para su correcto funcionamiento. De hecho, debido a que el hardware del equipo está todavía en fase de pruebas, para este proyecto se ha trabajado sobre una placa de evaluación de tipo conga-CEVAL de Congatec (como la de la siguiente figura) e imagen base genérica como sistema operativo, por no disponer todavía de la imagen personalizada para el equipo.

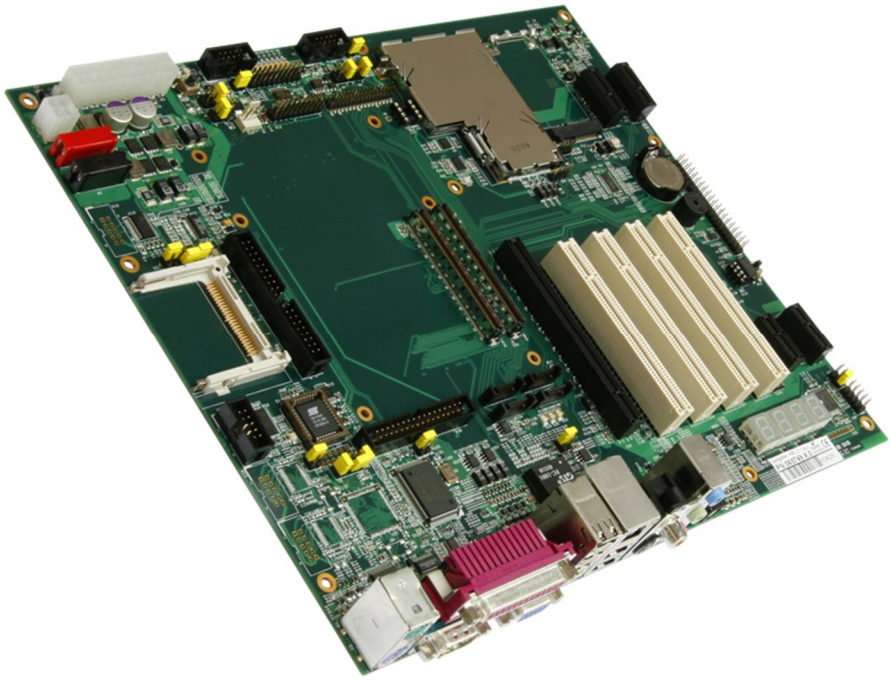


Figura 6. Placa de evaluación
Fuente: [conga-CEVAL 16]

La placa de evaluación alberga el sistema operativo en una Compact Flash (CF) que es un tipo de dispositivo de almacenamiento de datos y con memoria no volátil.

Figura 7. Compact Flash usada para la placa de evaluación
Fuente: [TRANSCEND 16]

En el Departamento de Software existe una necesidad de dotar a estos equipos (HMI y sDIAG) de características de tiempo-real, debido a que la respuesta que den ante distintos eventos del tren debe ser rápida y versátil. Ésta es una de las principales tareas que se abordarán en este proyecto. Antes de comenzar con la implementación de la solución, es necesario realizar previamente un estudio del estado del arte en temas de sistemas operativos de tiempo-real (SOTR) y plantear las soluciones más apropiadas para cubrir esta necesidad.

3. Sistemas operativos de tiempo-real (SOTR)

La diferencia entre un sistema operativo de tiempo-real (SOTR) y un sistema operativo de propósito general es que un SOTR está especialmente diseñado para ejecutar aplicaciones de tiempo vital dentro de restricciones de tiempo. Por ejemplo, el equipo central del diagnóstico de un tren. De este modo, es evidente que el tiempo máximo utilizado para una operación se tiene que tener en cuenta en un SOTR. Estas restricciones de tiempo se usan e implementan en reglas llamadas políticas de planificación que principalmente priorizan tareas basadas en cómo de cruciales son sus atributos de tiempo. Los sistemas de tiempo-real están comúnmente divididos en dos grandes grupos: estricto (*hard*) y no estricto (*soft*). En sistemas de tiempo-real estricto se da por hecho que el tiempo de una aplicación toma el papel principal. De hecho, si el sistema falla en el desempeño de la aplicación en el sistema de tiempo-real estricto, al no cumplir sus específicas restricciones de tiempo, la consecuencia puede ser una catástrofe. Por otro lado, un fallo de tiempo en el desempeño de una aplicación de tiempo-real no estricto no resulta en un desastre. Una aplicación de tiempo-real estricto debe tomar decisiones en tiempos, de lo contrario el resultado será desastroso.

3.1. Kernel en sistemas operativos de tiempo-real

Como principal componente de un sistema operativo, el kernel, es responsable de conectar el mecanismo hardware del sistema a la capa de aplicación. El kernel SOTR es responsable de gestionar diferentes operaciones tareas tales como: creación, ejecución y borrado, acorde a la política que los desarrolladores establecen para ello. También, al igual que en los kernels de propósito general, las tareas de gestión de memoria, manejo de interrupción y gestión de comunicaciones de entrada/salida se realizan en los kernels SOTR. Sin embargo, la principal diferencia entre un kernel SOTR y un kernel de propósito general radica en el hecho de que el kernel que no es de tiempo-real no garantiza que se pueda llevar a cabo una tarea en tiempo-real dentro de sus restricciones de tiempo.

3.1.1. Componentes del kernel en un SOTR

Típicamente, los principales componentes de un kernel incluyen: planificador, manejador de interrupciones, gestor de memoria y otros servicios. Una breve descripción de cada componente se da a continuación:

- Planificador: el planificador tiene el rol de gestión en el kernel. Reparte el tiempo del procesador a las diferentes tareas basado en la política de planificación (o políticas) que se diseñan para ello. Una descripción de algunas de las más comunes políticas de planificación se dan más adelante, en la sección 3.1.4.
- Gestor de interrupciones: el manejador de interrupciones es responsable de gestionar las peticiones de diferentes dispositivos hardware, así como llamadas al sistema. Un manejador de interrupciones puede iniciarse por interrupciones que viene del hardware o de la ejecución de código de interrupción en software.
- Gestor de memoria: la sección de gestión de memoria proporciona a los usuarios del kernel recursos en memoria del sistema (reserva de memoria).
- Otros servicios: dependiendo de los requerimientos de la aplicación así como el tipo y diseño del kernel, el sistema operativo ejecuta un cierto número de actividades *"en background"* (concurrentemente). Estas actividades incluyen la gestión del acceso a recursos compartidos y sincronización de tiempos.

3.1.2. Tarea

En un sistema operativo de propósito general, los procesos de tiempo-real conviven con procesos de tiempo compartido o procesos del propio sistema, por lo que es muy difícil garantizar siempre el cumplimiento de los plazos. Sistemas como Linux o Windows poseen una clase de procesos de tiempo-real, pero la existencia de trozos de código no expulsable en el kernel hace que sea difícil acotar los tiempos de respuesta. Por lo tanto, a veces se precisan sistemas en tiempo-real con políticas de planificación específicas [Lafuente 09].

3.1.2.1. Restricción de tiempos de una tarea

Introduciremos las siguientes definiciones:

- Instante de liberación, F_i. Una tarea i no puede comenzar a ejecutarse antes del instante F_i (antes estará bloquedo o no se habrá creado).
- Tiempo de ejecución, e_i. Tiempo que necesita la tarea i usando los recursos de la máquina para producir una respuesta.
- Plazo, D_i. Instante de tiempo en el que la tarea i debe haber respondido. Es condición necesaria:

$$e_i < D_i - F_i$$

- Periodo, P_i. Tiempo entre dos instantes de liberación de la tarea i. Habitualmente,

$$P_i = D_i - F_i$$

- Tarea periódica. P_i es constante.
- Tarea aperiódica. P_i es variable. Puede estar acotado o no. Si está acotado, una tarea aperiódica se convierte en periódica asignándole como periodo su cota de periodo menor.

En cuanto al cumplimiento de los plazos, en algunos sistemas la ejecución de las tareas tiene un plazo firme, en el sentido de que si se rebasa el plazo, aun mínimamente, la respuesta es inútil o incluso contraproducente. En cambio, los sistemas de plazo flexible admiten algún retraso en el incumplimiento del plazo. El valor de la respuesta en estos sistemas disminuye progresivamente por encima del plazo. Es habitual que una aplicación de tiempo-real admita un plazo flexible con un límite. Cada vez más, muchos sistemas de tiempo-real son componentes de otros sistemas, en los que realizan funciones de control. Se les denomina sistemas empotrados (en Inglés, *embedded systems*). Generalmente, el sistema de control no se percibe desde fuera. Finalmente, hay que advertir que en un sistema de tiempo-real las tareas pueden interaccionar entre sí, pasándose información.

3.1.2.2. Interacción entre tareas

En la mayoría de estos sistemas las tareas interaccionan mediante:

- Datos comunes.
- Mensajes.

En todos estos casos puede ocurrir que una tarea tenga que esperar un suceso de otra menos prioritaria. Esta situación se denomina bloqueo, y produce una inversión de prioridad indeseable. La inversión de prioridad no se puede eliminar completamente, pero es posible limitar su duración. La herencia de prioridad es una forma de reducir la duración de los bloqueos variando dinámicamente la prioridad de las tareas. Cuando una tarea está bloqueando a otra más prioritaria, hereda la prioridad de ésta. La prioridad dinámica de una tarea es el máximo de: su prioridad básica y las prioridades de todas las tareas bloqueadas por ella. Con el protocolo de herencia de prioridad, una tarea se puede bloquear como máximo: una vez por cada recurso y una vez por cada tarea de prioridad inferior.

3.1.3. Criterios para la planificación de tiempo-real

El criterio de planificación de un algoritmo de planificación de tiempo-real para un conjunto de tareas de tiempo-real $\{T_1, T_2, \ldots, T_n\}$ es proporcionar planificación viable, es decir, debe ser posible planificar la ejecución de las tares de forma que se cumplan los plazos de todas ellas. Una condición necesaria para ello es:

$$\sum_i \frac{ei}{Pi} < 1$$

Con tareas periódicas es posible un análisis de viabilidad estático (*a priori*). Si existen tareas aperiódicas, es preciso convertirlas en periódicas con periodo el mínimo con el que la tarea pueda liberarse. En caso contrario, el análisis sólo puede hacerse en tiempo de ejecución.

El criterio de planificación viable es absoluto para los sistemas de tiempo-real crítico. En sistemas acríticos, si los plazos no se pueden cumplir, los criterios a aplicar dependen del tipo de sistema. En sistemas de plazo firme, el criterio es minimizar el número de retrasos; por el contrario, en sistemas de plazo flexible, el criterio es minimizar la magnitud de los retrasos.

3.1.4. Políticas de planificación de tiempo-real

Se dividen en dos grupos, considerando si admiten un plan de ejecución estático, lo que implica que las tareas deben ser periódicas, o si la ejecución debe planificarse dinámicamente. Estas últimas no sirven para sistemas críticos.

3.1.4.1. Planificación estática

Cuando se conocen los periodos y los tiempos de ejecución es posible establecer *a priori* un plan de ejecución. En el caso más sencillo, se establece un ejecutivo cíclico, asociando a instantes periódicos de tiempo la liberación de las tareas. Otros métodos más generales siguen criterios basados en prioridades:

- Frecuencia monótona (en Inglés, *Rate monotonic*, RM)

  Es una política expulsora donde la más prioritaria es la tarea de menor periodo.

- Planificación de plazo más cercano (en Inglés, *Earliest deadline first,* EDF)

  Es una política expulsora que planifica la tarea cuyo plazo está más próximo a expirar. Esta política es óptima con respecto al criterio de planificación viable (es decir, la condición necesaria para la viabilidad es también suficiente, lo que no ocurre con RM).

3.1.4.2. Planificación dinámica

Con tareas aperiódicas no es posible establecer un plan de ejecución *a priori*. En el momento en que se libera una tarea, se establece la planificación con el objetivo de hacerla viable. En algunos sistemas, ni siquiera en ese momento es posible un plan de viabilidad (por ejemplo, si no se conocen los tiempos de ejecución). Sólo una vez cumplido el plazo se sabe que no es viable, y entonces se aborta la ejecución de la tarea. Este último tipo de planificación, habitual en sistemas de propósito general, se denomina planificación dinámica del mejor resultado (en Inglés, *best-effort*). Por ejemplo, Round-robin (RR), que consiste en ejecutar las tareas comenzando por el primer elemento de la lista hasta llegar al último y empezando de nuevo desde el primer elemento.

3.2. Ejemplos de sistemas operativos en tiempo-real existentes

Existen diversos aspectos que se deben analizar para que sistemas operativos soporten actividades de tiempo-real. Todos estos temas se relacionan con el comportamiento no determinista del sistema, que hace que los procesos experimenten latencias de impredecible longitud durante la ejecución. Todas la aplicaciones de tiempo-real tienen restricciones de tiempo (plazos, como hemos visto en la sección 3.1.2.1) que se deben satisfacer, por el contrario el sistema no funciona apropiadamente.

La latencia de un sistema operativo se puede definir de diferentes modos. En general, latencia es el tiempo que trancurre desde que ocurre un evento hasta el inicio de la acción que responderá al evento. En el desarrollo de sistemas de tiempo-real crítico, es necesario contar con el peor escenario posible (en Inglés, *worst-case execution time*, WCET). Las dos principales fuentes de latencia en sistemas operativos de propósito general son la latencia de la tarea y la resolución de tiempos. La latencia de la tarea se experimenta por un proceso que no puede desalojar una tarea de menor prioridad porque se está ejecutando.

Algunos sistemas operativos de tiempo-real se describen a continuación [Priya et al. 14]:

- Free RTOS

Free RTOS es un sencillo pero extendido sistema operativo de tiempo-real. Este sistema operativo está principalmente escrito en el lenguaje de programación C. El hecho de que haga uso de un simple y pequeño kernel hace a este sistema operativo compatible para sistemas empotrados. Del mismo modo, ha mostrado tener baja sobrecarga.
Atributos de las tareas: prioridad. Política de planificación: planificación con expulsión y prioridad fija, Round-robin (RR) para hilos de ejecución con la misma prioridad.

- Enea OSE

Enea OSE es un producto de la firma sueca Enea AB. Este sistema operativo de tiempo-real compacto ha sido ampliamente usado en teléfonos móviles. Una plataforma compatible para este sistema operativo es la familia ARM. La versión multiprocesador de este sistema operativo, Enea OSE Multicore, se lanzó en 2009 con la misma arquitectura kernel que Enea OSE.
Atributos de las tareas: prioridad. Política de planificación: planificación con desalojo y prioridad fija, planificación periódica, planificación Round-robin (RR) para tareas con la misma prioridad.

- Windows Embedded CE

Windows Embedded CE, también conocido como WinCE, es un producto Microsoft desarrollado para sistemas empotrados y microcontroladores. Este sistema operativo, que se escribió en el lenguaje de programación C, se lanzó en 1996 y, actualmente, la séptima versión está disponible. Obviamente, como Windows CE se ha diseñado para sistemas empotrados, utiliza un tamaño de memoria pequeño. Muchas plataformas posteriores tales como: AutoPC, Windows Mobile 2003 y Windows Phone se han ido desarrollando basándose en el kernel de Windows CE.
Atributos de las tareas: prioridad, 256 prioridades. Política de planificación: planificación con desalojo y prioridad fija.

- Lynx OS

Otro sistema operativo de tiempo-real escrito en los lenguajes de programación C, C++ y Ada es Lynx OS. Lynx se lanzó inicialmente en 1986 y, actualmente, la quinta versión de este sistema operativo está disponible para los usuarios. Lynx es un sistema operativo de código cerrado. El planificador publica números de rendimiento de tiempo-real preciso acorde con el rendimiento del sistema en situaciones del mundo real tales como latencia de interrupción, tiempo de cambio de contexto, tiempo de desalojo y otros.
Política de planificación: planificación con desalojo y prioridad fija.

- VxWorks

VxWorks es un sistema operativo de 64 bits que se puede usar en sistemas empotrados. El soporte de servicios de recurso compartido así como colas de mensaje local y distribuido y los beneficios aportados por un diseño compacto, han hecho este SOTR útil en diferentes sectores industriales. Y que hacen uso de sistemas empotrados, tales como aviación y automoción.
Atributos de las tareas: prioridad. Política de planificación: planificación con desalojo y prioridad fija (256 niveles de prioridad), Round-robin, puede usar POSIX (FIFO y RR). POSIX (*Portable Operating System Interface*) es una norma escrita por la IEEE que define una interfaz del sistema operativo y el entorno, incluyendo un intérprete de comandos, y programas de utilidades comunes para apoyar la portabilidad de las aplicaciones a nivel de código fuente.

- RT Linux

RT Linux es una versión modificada del sistema operativo Linux que se lanzó primeramente en 2007. El principal objetivo de este kernel es proporcionar soporte para tareas de tiempo-real

estricto. Además, la transparencia y la modularidad son puntos importantes en el desarrollo de RT Linux. RT Linux se ha escrito en el lenguaje de programación C.
Política de planificación: EDF (*Earliest deadline first*), RM (*Rate monotonic*), puede usar POSIX (FIFO y RR).

- Xenomai

Xenomai es otro proyecto que apunta a proporcionar soporte en tiempo-real para sistemas operativos GNU/Linux. Xenomai es compatible para plataformas tales como: ARM y x86.
Atributos de las tareas: prioridad en el momento de creación. Política de planificación: planificación con desalojo y prioridad fija. FIFO y Round-robin para tareas con la misma prioridad. Xenomai se explicará más en detalle en la sección 3.4.2.

- QNX

QNX es otro sistema operativo de tiempo-real principalmente para sistemas empotrados. Este sistema operativo que lo lanzó en 1982 Research in Motion, es un sistemas operativo tipo Unix que ejecuta la mayoría de las operaciones en forma de pequeñas tareas llamadas servidores.
Atributos de las tareas: no WCET o atributos no funcionales específicos. Políticas de planificación: planificación FIFO, planificación Round-robin y planificación adaptable.

- RTAI

RTAI (*Real-Time Application Interface*) se desarrolló como una extensión de sistema operativo de tiempo-real para Linux. Este proyecto que es compatible con POSIX usa una arquitectura de capa de abstracción de tiempo-real para manejar tareas de tiempo-real.
Atributos de las tareas: prioridad (enteros de 16 bits con 0 siendo la prioridad más alta). Política de planificación: planificación con desalojo y prioridad fija para tareas tanto periódicas como no periódicas.

- eCos

Por último, eCos se desarrolló para ser altamente configurable, y así ofrecer el poder para personalizar el sistema operativo para necesidades particulares de las aplicaciones. eCos funciona sobre varias plataformas tales como: x86, PowerPC, MIPS o ARM. Su desarrollo lo impulsó la empresa Red Hat, que en 2004 delegó los derechos a la Free Software Foundation para que continuara con su desarrollo.

La mayoría de los sistemas operativos de tiempo-real descritos están implementados en C, siendo un lenguaje de programación fácil de manejar y ampliamente conocido. Tienen planificadores que administran las tareas, ya sea que estén listas para ejecutarse, bloqueadas o suspendidas y permitiendo su desalojo.

3.3. Experiencia de CAF Power & Automation con sistemas operativos en tiempo-real existentes

Los proyectos con necesidades de tiempo-real desarrollados en la empresa han ido relacionados con:

- RTAI en un entorno muy restringido y sin necesidades de GUI, orientado principalmente a buses de comunicaciones y ejecución de cierta lógica periódica.
- QNX (versiones 6.1 y 6.3.2) para algunos dispositivos x86.
- Linux from *Scratch*, con kernel optimizado.

La selección del sistemas operativo de tiempo-real depende de múltiples factores, y de los servicios que se quiera proveer. Las necesidades se centraban en:

- Buen entorno para generación de aplicaciones gráficas como por ejemplo vía Qt.
- Buen soporte de tecnologías multimedia, *streaming* de vídeo, audio, etc.

- Sistema POSIX.
- Fácilmente mantenible.
- Bien documentado.
- Algo muy usado.

Estos proyectos datan de hace alrededor de seis años.

El hecho de que buena parte del software empotrado del HMI y del sDIAG esté basado en Linux hace que nos detengamos más en el estudio de Linux en tiempo-real.

3.4. Enfoques hacia sistemas operativos basados en Linux en tiempo-real

Debido a que Linux es un sistema operativo de propósito general, su rendimiento en tiempo-real es relativamente débil, y no puede satisfacer las necesidades que demandan algunas industrias concretas. Por consiguiente, mejorar el tiempo-real en Linux es de interés para su estudio y aplicación [Fayyad-Kazan et al. 13]. El diseño para Linux en tiempo-real sólo puede satisfacer las necesidades de tareas de tiempo-real no estricto (*soft*).

Numerosos proyectos se han propuesto para hacer Linux en tiempo-real. Ello viene motivado por varias razones, entre ellas: disponibilidad de una gran cantidad de aplicaciones distribuidas con licencia de código abierto, robustez y flexibilidad del kernel o su adaptabilidad. El software desarrollado en Linux es de muy bajo coste y puede ejecutarse en varias plataformas hardware [Xu et al. 10].

Los enfoques más importantes que se han tenido en cuenta para llevar los requerimientos de tiempo-real a Linux son dos:

- Enfoque 1: Mejorar el propio kernel de Linux para que adopte requerimientos de tiempo-real, proporcionando latencias acotadas, APIs de tiempo-real, etc. Entre las distintas formas de realizar esta mejora sobre el kernel Linux, tenemos las iniciativas PREEMPT_RT y SCHED_DEADLINE. Éstas se abordarán más adelante, en la sección 3.4.1 y 3.4.3, respectivamente.
- Enfoque 2: Añadir una capa por debajo del kernel Linux que manejará todos los requerimientos de tiempo-real, de modo que el comportamiento de Linux no afecta a las tareas de tiempo-real. Esto se llama abstracción de interrupción. Está basado en un pequeño planificador de tiempo-real que tiene pleno control de las interrupciones y las características clave del procesador, y que ejecuta Linux como un hilo. Para obtener comportamiento de tiempo-real, este planificador de tiempo-real trata el kernel Linux entero como una tarea más de forma que éste se ejecuta cuando no existen tareas de tiempo-real que ejecutar. Así, Linux nunca puede bloquear otras interrupciones que se requieran atender en tiempo-real. Este enfoque se ha implementado satisfactoriamente en diversos entornos, los ejemplos más notables son: RT Linux, RTAI y Xenomai. Se trata de una solución eficiente y obtiene bajas latencias, pero es también invasivo y a menudo, no todas las instalaciones Linux estándar están disponibles para tareas ejecutándose con privilegios de tiempo-real.

Cuando se desarrollan aplicaciones de tiempo-real con un sistema basado Linux, el escenario típico es el siguiente [Petazzoni -]:

Ocurre un evento del mundo físico y se notifica a la CPU por medio de una interrupción. El manejador de interrupciones reconoce y maneja el evento, y entonces despierta a la tarea del espacio de usuario que responderá a este evento. Algún tiempo después, la tarea de espacio de usuario se ejecutará y será capaz de responder al evento del mundo físico.

El tiempo-real hace referencia a proporcionar garantías en las latencias en el peor de los casos.

Latencia de kernel = latencia de interrupción + duración del manejador + latencia del planificador + duración del planificador

- Uso de *spinlocks* o cerrojos

Uno de los mecanismos de concurrencia usados en el kernel es *spinlock* (cerrojo de espera activa). Los cerrojos son el mecanismo de exclusión mutua más utilizado en el kernel de Linux. Un cerrojo es una variable que solamente puede tener dos valores: bloqueado y desbloqueado. De esta manera, cuando un proceso va a entrar en su región crítica, comprueba si el cerrojo está desbloqueado y, de ser así, lo bloquea y entra a ejecutar el código. Si encuentra el cerrojo bloqueado, entra en un blucle de espera en el que comprueba si el cerrojo se desbloquea. El cerrojo se debe desbloquear cuando un proceso abandone la región crítica.

El uso de cerrojos tiene diversas variantes. Una de las variantes comúnmente usadas para prevenir los conflictos de concurrencia entre un contexto de proceso y un contexto de interrupción es la deshabilitación de interrupciones.

Las secciones críticas protegidas por *spinlocks*, u otra sección en que las interrupciones se deshabilitan explícitamente, retrasarán el inicio de la ejecución del manejador de interrupciones. Otra opción es el uso de interrupciones compartidas. En Linux, muchos manejadores de interrupciones se dividen en dos partes:

- La parte de arriba, iniciada por la CPU en el momento en el que se activa la interrupción. Se ejecuta con la línea de interrupción inhibida y debería terminar muy rápido.
- La parte de abajo, planificada por la parte de arriba, comienza cuando la parte de arriba ha terminado la ejecución. Por consiguiente, para interrupciones críticas de tiempo-real, la parte de abajo no se debería usar: su ejecución se puede retrasar por culpa de otras interrupciones del sistema.

El kernel de Linux se dice que es expulsor. Esto es, cuando una tarea se ejecuta en modo de espacio de usuario y una interrupción la interrumpe, se deja de ejecutar la tarea y el manejador de interrupciones despierta a otra tarea para que atienda la interrupción. La tarea cuya ejecución se ha interrumpido se puede planificar para tan pronto como el manejador de interrupciones termine su trabajo.

Sin embargo, cuando la interrupción viene mientras se ejecuta una llamada al sistema, esta llamada al sistema tiene que finalizar antes de que otra tarea se pueda planificar. Por defecto, el kernel Linux no hace expulsión de kernel. Esto significa que el planificador tiene que esperar hasta que termine la tarea que se está ejecutando antes de planificar la tarea que atenderá la interrupción.

- Prioridad de manejador de interrupción

En Linux, los manejadores de interrupciones se ejecutan directamente por los mecanismos de interrupción de CPU, y no bajo control del planificador de Linux. Por consiguiente, todos los manejadores de interrupciones tienen una prioridad más alta que todas las tareas que se están ejecutando en el sistema.

3.4.1. PREEMPT_RT

Algunas de las características que incluye el parche PREEMPT_RT para tiempo-real son las siguientes:

- Temporizadores de alta-resolución propuesto por Thomas Gleixner.
- Validación segura del kernel.
- Interrupciones genéricas para todas las arquitecturas.
- Robustas llamadas al sistema *futex*.
- Herencia de prioridades.

El objetivo es mejorar gradualmente el kernel Linux teniendo en cuenta los requerimientos de tiempo-real y conseguir que estas mejoras se incorporen a la línea principal del kernel. Muchas de las mejoras diseñadas, desarrolladas y depuradas dentro de PREEMPT_RT durante años son ahora parte de la línea principal del kernel Linux.

3.4.1.1. Mejoras incluidas en PREEMPT_RT

Desde el inicio de 2.6:

- Planificador O(1).
- Kernel expulsor.
- Mejor soporte API de tiempo-real POSIX.

Desde la 2.6.18:

Soporte de herencia de prioridad para mutexes. Un mutex es un semáforo binario que se usa en un sistema para asegurar acceso compartido al mismo recurso.

Desde 2.6.21:

Temporizadores de alta-resolución.

Desde 2.6.30:

Interrupciones en hilos.

Desde 2.6.33:

Anotaciones *spinlock*.

3.4.1.2. Nuevas opciones de expulsión en Linux 2.6

El estándar Linux 2.6 ofrece tres nuevos modelos de expulsión:

1ª opción (expulsión no forzada):

CONFIG_PREEMPT_NONE

El código kernel (interrupciones, excepciones y llamadas al sistema) nunca es expulsor. Es el comportamiento por defecto en el kernel estándar, idóneo para sistemas que hacen intensos cálculos computacionales en el que el rendimiento total es clave. Es lo mejor para reducir el cambio de tarea para maximizar la CPU y el uso de caché (reduciendo el cambio de contexto).

2ª opción (expulsión kernel voluntario):

CONFIG_PREEMPT_VOLUNTARY

El código kernel puede ser expulsor por sí mismo. Se utiliza típicamente para sistemas de escritorio, para obtener una reacción más rápida de las aplicaciones que dan respuesta a entradas de usuario. Añade puntos de replanificación explícita a través del código de kernel. Esto supone menor impacto en rendimiento.

3ª opción (kernel expulsable):

CONFIG_PREEMPT

La mayoría del código kernel puede ser involuntariamente expulsor en cualquier momento. Cuando un proceso se convierte ejecutable, no hay más necesidad para esperar por el código de kernel (típicamente una llamada al sistema) para devolver antes de ejecutar el planificador.

Típicamente para sistemas empotrados con requerimientos de latencia en el rango de milisegundos. Esto supone un relativamente menor impacto en rendimiento.

3.4.1.3. Temporizadores de alta-resolución

La resolución de los temporizadores acostumbra a estar limitada a la frecuencia del *tick* (del reloj del sistema). Usualmente 100 Hz ó 250 Hz, dependiendo de la arquitectura y la configuración. Una resolución de sólo 10 ms ó 4 ms. Incrementar la frecuencia tick (del reloj del sistema) no es una opción porque consumiría demasiados recursos. Es un parámetro importante para aplicaciones próximas a tiempo-real y multimedia. Es la capacidad de procesar eventos a altas frecuencias. El término alto en este contexto significa usualmente más grande que 100 Hz (10 ms). Antiguamente, la configuración de kernel CONFIG_HZ era la configuración más importante para la frecuencia de temporizador de kernel. Se definía en tiempo de compilación del kernel y las configuraciones variaban entre distribuciones y versiones del kernel. Hoy en día, un número de configuraciones kernel, tales como NO_HZ, HIGH_RES_TIMERS impactan en la frecuencia de interrupción del temporizador del kernel también. La disponibilidad de soporte HPET (*High Precision Event Timer*) también es una característica importante. Todas estas configuraciones se pueden recuperar fácilmente de la actual configuración del kernel. Sin embargo, para una aplicación próxima a tiempo-real o multimedia cuenta la frecuencia de interrupción de tiempo. Esta frecuencia se puede estimar bastante bien realizando peticiones de interrupciones de tiempo.

El siguiente programa de ejemplo [ADVENAGE 16] representa un algoritmo para determinar la frecuencia de interrupción de tiempo alcanzable haciendo peticiones de interrupciones de tiempo a altas frecuencias:

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#define USECREQ 250
#define LOOPS   1000

void event_handler (int signum)
{
 static unsigned long cnt = 0;
 static struct timeval tsFirst;
 if (cnt == 0) {
   gettimeofday (&tsFirst, 0);
 }
 cnt ++;
 if (cnt >= LOOPS) {
   struct timeval tsNow;
   struct timeval diff;
   setitimer (ITIMER_REAL, NULL, NULL);
   gettimeofday (&tsNow, 0);
   timersub(&tsNow, &tsFirst, &diff);
   unsigned long long udiff = (diff.tv_sec * 1000000) + diff.tv_usec;
   double delta = (double)(udiff/cnt)/1000000;
   int hz = (unsigned)(1.0/delta);
   printf ("kernel timer interrupt frequency is approx. %d Hz", hz);
   if (hz >= (int) (1.0/((double)(USECREQ)/1000000))) {
     printf (" or higher");
   }
   printf ("\n");
   exit (0);
 }
}

int main (int argc, char **argv)
```

```
{
 struct sigaction sa;
 struct itimerval timer;

 memset (&sa, 0, sizeof (sa));
 sa.sa_handler = &event_handler;
 sigaction (SIGALRM, &sa, NULL);
 timer.it_value.tv_sec = 0;
 timer.it_value.tv_usec = USECREQ;
 timer.it_interval.tv_sec = 0;
 timer.it_interval.tv_usec = USECREQ;
 setitimer (ITIMER_REAL, &timer, NULL);
 while (1);
}
```

Figura 8. Programa en C [ADVENAGE 16] para determinar
la frecuencia de interrupción de tiempo

Por ejemplo, tras ejecutar el anterior código fuente, la frecuencia de interrupción de tiempo en la placa de evaluación del sDIAG es aproximadamente 3952 Hz.

La infraestructura de temporizadores de alta-resolución [Gleixner et al. 06], se añadió en la versión 2.6.21 del kernel Linux, permite usar los temporizadores de hardware disponibles para programar interrupciones en el momento correcto. Los temporizadores hardware se multiplexan, por lo que un temporizador de hardware único es suficiente para manejar un gran número de temporizadores programados de software. Se pueden utilizar directamente desde el espacio de usuario a través de las APIs de temporizador.

3.4.1.4. CONFIG_PREEMPT_RT

La solución PREEMPT_RT añade un nuevo nivel de expulsión, llamado CONFIG_PREEMPT_RT. Este nivel de expulsión sustituye todos los *spinlocks* del kernel por mutexes (o también llamados *spinlocks* dormidos). En lugar de proporcionar exclusión mutua deshabilitando interrupciones y expulsión, son cerrojos normales: cuando la contención sucede, el proceso se bloquea y el planificador selecciona otro proceso.

En resumen, PREEMPT_RT es un parche de kernel para hacer un sistema Linux más predecible y determinista. Esto se realiza a través de diversas optimizaciones. Este parche hace casi todo el código del kernel expulsable, excepto para la mayoría de rutinas críticas de kernel, reduciendo la latencia máxima que experimenta una tarea (del coste de una latencia media ligeramente más alta). Esta característica se logra sustituyendo cada *spinlock* en el código del kernel con mutexes expulsables con herencia de prioridad. El parche también proporciona interrupciones en hilos convirtiendo manejadores de interrupción en hilos de kernel expulsable que el planificador de Linux habitual gestiona en contexto de proceso. Desde que algunas partes del parche PREEMPT_RT se consideran suficientemente estables (ejemplo, algoritmo de herencia de prioridad), se han adoptado en la distribución oficial del kernel Linux.

3.4.2. Xenomai

Xenomai [Regnier et al. 08] es un subsistema de tiempo-real que se puede integrar con el kernel de Linux para incrementar la predicibilidad del sistema. En su última versión mejorada, utiliza un pequeño kernel adicional que coopera con el kernel de Linux. Una gran diferencia de Xenomai frente al resto de soluciones es que permite que las tareas de tiempo-real trabajen en el espacio de usuario. De hecho, Xenomai hace uso del concepto de introducir dos dominios para ejecutar las tareas, un dominio primario, controlado por la sección de tiempo-real, y un dominio secundario controlado por el kernel de Linux. Los hilos de tiempo-real se manejan en el dominio primario a no ser que se llame al API estándar Linux. Hay que mencionar que Xenomai tiene la capacidad de separar los procesos de tiempo-real de las tareas normales que crea el API POSIX estándar de Linux. Esto significa que los hilos de tiempo-real en el kernel Xenomai heredan la capacidad de invocar las funciones Linux cuando no se ejecutan en modo

tiempo-real. Una ventaja de esta separación de tareas es que las tareas de tiempo-real no usan el mismo espacio de memoria como las otras tareas, por consiguiente, la posibilidad que ocurra una colisión debido a la existencia de errores en una tarea de tiempo-real se reduce. Un problema con Xenomai es la latencia que seguirá experimentando cuando se trata con tareas de tiempo-real en dominio primario. Esta latencia es más grande que las latencias impuestas en RTAI. Por otra parte, el dominio secundario, controlado por el planificador del kernel de Linux, también impone otra latencia al sistema.

3.4.3. Otros enfoques: RTAI, LITMUSRT, TimeSys y SCHED_DEADLINE.

- RTAI

RTAI (*Real-Time Application Interface*) ha mostrado ser un enfoque robusto y maduro que la comunidad ha ido desarrollando constantemente. El proyecto RTAI está principalmente basado en módulos y el parche del kernel que se aplica sólo cambia unas pocas líneas de código en el kernel de Linux estándar. Las tareas de tiempo-real que se construyen como módulos tienen la ventaja de que el SOTR y la tarea de tiempo-real comparten el mismo espacio de ejecución y llamada al sistema. Por consiguiente, se implementan como llamadas de función simple en lugar de interrupciones de software de bajo nivel. Además, las tareas se ejecutan en modo de supervisión de procesador dando pleno acceso al hardware. Mientras que la técnica principal usada en RTAI es bastante similar a la usada en RT Linux (sección 3.2.), la mayor diferencia de arquitectura entre estos dos proyectos es cómo añaden las características de tiempo-real a Linux. La capa de abstracción de interrupción en RTAI tiene una estructura de punteros al vector de interrupciones que confina la gestión de interrupciones. Esta capa de abstracción se construye con menos de 20 líneas de modificaciones y sólo hay 50 líneas que se añaden al código original. Además cuando no se requieran operaciones de tiempo-real, para volver a la estructura original es suficiente con cambiar los punteros en la capa de abstracción de tiempo-real a la posición que tenían antes de implementar RTAI. El trabajo del planificador de tareas RTAI se basa en una planificación expulsora de prioridades fijas. También, RTAI da permiso para tener acceso inmediato al hardware del PC eliminando la necesidad de tener que pasar primero por las capas de gestión de interrupciones del kernel de Linux original [Zhang et al. 06]. También, al usar un módulo cargable, RTAI ejecuta un subconjunto de POSIX 1003.1.c. El soporte POSIX en RTAI es bastante similar al estándar Linux excepto en el hecho de que no soporta la gestión de señales y funciones de creación de procesos hijo que incluye el Linux estándar (que son inapropiados para propósitos de tiempo-real, porque todos los hilos de ejecución se consideran como partes de un único proceso).

En [Scordino et al. 06] se hace una comparación entre el rendimiento obtenido con Linux estándar, PREEMPT_RT y un sistema que emplea tanto Xenomai como RTAI, siendo la latencia más alta la del Linux estándar.

- LITMUSRT

LITMUSRT (*Linux Testbed for Multiprocessor Scheduling in Real-Time Systems*) es un proyecto basado en modificaciones aplicadas al kernel de Linux 2.6.20 configurado para una arquitectura SMP (*Symmetric Multiprocessor*). En este proyecto se crea una interfaz de planificación que permite que se añadan nuevos algoritmos de planificación. Otro subgrupo principal de LITMUSRT son las librerías de espacio de usuario para soportar la creación y ejecución de tareas de tiempo-real, ya que en LITMUSRT sólo se planifican las tareas de tiempo-real. LITMUSRT opera de dos modos: tiempo-real y no tiempo-real. El sistema está básicamente en modo no tiempo-real al inicio para configurar las tareas completas antes realizar la planificación. La función *core* "*LITMUSRT timer tick*" (`rt_scheduler_tick()`) es responsable del cambio de modo. Para lanzar y encolar, las tareas de tiempo-real, LITMUSRT proporciona la abstracción de un dominio de tiempo-real implementado por la estructura `rt-domain-t`. Este dominio de abstracción consiste de una cola *ready* y una cola *release*. Una función de orden EDF se usa para organizar la cola *ready*. Aunque un algoritmo de control de retroalimentación EDF (FC-EDF) se introdujo más tarde como función de orden en LITMUSRT. Las funciones *wrapper* se crean para operaciones de listado tales como el encolamiento o la exclusión de tareas de la cola, y funciones tales como `list_insert()` (usado para poner tareas en la cola) y `list_qsort()` (usado para ordenar la cola) se usan para extender el API

Linux `list.h`. Hay trece tipos de *plug-in* de planificación que se usan en LITMUSRT, de forma que se usa uno de ellos en cada momento. De hecho, cada algoritmo de planificación se crea como un *plug-in*. Durante el arranque del sistema, el parámetro de línea de comando del kernel `rtsched` determina qué decisiones de planificación se deberían tomar. Una breve descripción de estos pasos se describe en [Brandenburg et al. 07].

- TimeSys

La idea de TimeSys [TimeSys 16] es usar las llamadas *spinlock* para gestionar las expulsiones. En TimeSys, en lugar de los contadores de interrupción se usan mutexes. Mientras que en el enfoque de contador de interrupciones la expulsión está prohibida en las secciones bloqueadas por el *spinlock*, en el enfoque mutex la expulsión está disponible para un proceso de prioridad más alta cuando la prioridad más baja está usando un recurso diferente. Esto significa que es posible dar a las tareas de tiempo-real una prioridad más alta que algunas interrupciones. El mutex también usa un protocolo de herencia de prioridad para solventar el problema de inversión de prioridad. En la práctica, implementar este protocolo es difícil, pero da al kernel la posibilidad de reducir más la latencia del kernel hasta el nivel de las latencias obtenidas a través del método de abstracción de interrupciones. Junto a la extensión hecha en el planificador, los sistemas de interrupción se cambian en TimeSys. De hecho, las rutinas de servicio de interrupción (en Inglés, ISR) e IRQs se transforman a hilos de kernel. Esto significa que se planificarán también bajo el planificador de tiempo-real expulsor, que es cómo los hilos de tiempo-real son capaces de tener una prioridad más alta que las interrupciones. Además, desde que todos los manejadores de interrupción se tratan como hilos del planificador, es posible dormir un manejador de interrupciones. De este modo, es posible sustituir las interrupciones *spinlock* con mutexes. El enfoque TimeSys mejora las características de tiempo-real estricto (*hard*) en Linux, pero todavía no es posible desalojar el proceso kernel en cualquier punto arbitrario con seguridad.

- SCHED_DEADLINE

SCHED_DEADLINE [Lelli 14] es un parche para el kernel de Linux desarrollado por la empresa consultora Evidence (Italia) en el contexto del proyecto europeo ACTORS. SCHED_DEADLINE añade un planificador basado en plazos con reserva de recursos en el Linux kernel estándar. El algoritmo implementado es un CBS (*Constant Bandwith Server*) global, implementado como un algoritmo particionado con migración. La migración se puede deshabilitar, de este modo puede también trabajar como algoritmo particionado.

El parche añade una nueva clase de planificación al planificador de Linux, de forma que las tareas normales pueden comportarse como tal cuando el parche no está aplicado. Este parche también añade una llamada de sistema adicional para configurar la estimación de tiempo y el periodo para las tareas SCHED_DEADLINE.

El parche es independiente de plataforma; por consiguiente, soporta cualquier plataforma empotrada que admita Linux. También nativamente soporta arquitecturas *multicore*. Diversas opciones están disponibles en tiempo de ejecución como: reserva de ancho de banda, herencia de límites de tiempo, etc. Todavía está en fase de desarrollo, y se lanzan nuevas versiones en el LKML (en Inglés, *Linux Kernel Mailing List*). El planificador SCHED_DEADLINE está incorporado en el kernel de Linux desde su versión 3.14 [EVIDENCE 16].

En resumen, puede haber varios motivos para optar por una u otra solución de Linux en tiempo-real en estos momentos, pero los requisitos principales a tener en cuenta para este proyecto serían los siguientes:

- Entorno conocido, bien documentado y aceptado en la industria.
- Necesidades de tiempo-real ajustadas a las capacidades tiempo-real de Linux. No hay requisitos que apunten a respuestas menores a 1 ms.
- *Time to market* muy reducido.

En la siguiente sección se implementan diversas soluciones elegidas de Linux en tiempo-real. Y en función de los resultados se opta por la más adecuada para el HMI y el sDIAG.

4. Implementación de kernel de tiempo-real en Linux

El núcleo Linux, o kernel, es el encargado de la coordinación entre los programas que solicitan recursos y hardware. También se gestiona las tareas de una forma eficiente. Sobre el kernel se instalan los servicios, las aplicaciones y las interfaces gráficas.

En esta sección se muestra el proceso que se ha seguido para aplicar características de tiempo-real a un kernel de Linux estándar. Se comenzará con la instalación del kernel de Linux estándar. Después se explicarán las distintas formas elegidas para convertirlo en kernel de tiempo-real. La implementación de kernel de tiempo-real se ha llevado a cabo inicialmente en un PC/Windows con máquina virtual Linux. Posteriormente, en un PC/Linux nativo. Por último, se realizará una comparativa entre los resultados de las distintas soluciones propuestas, valorando cuál de ellas sería la más óptima para los equipos HMI y sDIAG, introducidos en las secciones 2.1 y 2.2, respectivamente.

Las soluciones que se han elegido para aplicar y comparar son:

1. Reducción de las latencias (*lowlatency*) del repositorio Ubuntu.
2. PREEMPT_RT (que se introdujo en la sección 3.4.1).
3. Xenomai (que se introdujo en la sección 3.4.2).

La máquina virtual nos permite realizar pruebas de configuración del kernel sin provocar daños graves al sistema por errores que se puedan cometer a la hora de ejecutar comandos en la línea *shell*.

4.1. Intérprete de comandos: *shell*

El intérprete de comandos (*shell*) es el programa que recibe lo que se escribe en el terminal y lo convierte en instrucciones para el sistema operativo. El objetivo de cualquier intérprete de comandos es ejecutar los comandos (órdenes), es decir, funciones del *shell* y programas (ya sean del usuario o del propio *shell*) que el usuario teclea en el *prompt* del mismo. El *prompt* es una indicación que muestra el intérprete para anunciar que espera una orden del usuario ($ para *user* y # para *superuser*). El comando principal es `man`, ya que con este comando se pude saber todo sobre los demás comandos haciendo `man` “comando”.

En Linux existen varias familias de *shell*. Éstas se diferencian entre sí básicamente en la sintaxis de sus comandos y en la interacción con el usuario. Concretamente, la *shell* incluida en el sistema operativo que se ha utilizado en este proyecto es *Bash* (*Bourne Again Shell*).

4.2. Instalación de la máquina virtual

La versión de máquina virtual que hemos instalado sobre Windows es la 4.3.36-105129 de VirtualBox. También hemos instalado su correspondiente paquete de extensión (Oracle_VM_VirtualBox_Extension_Pack-4.3.34-104062). Como punto de inicio, hemos partido de una distribución GNU/Linux Ubuntu 14.04.1 LTS que se puede conseguir libremente.

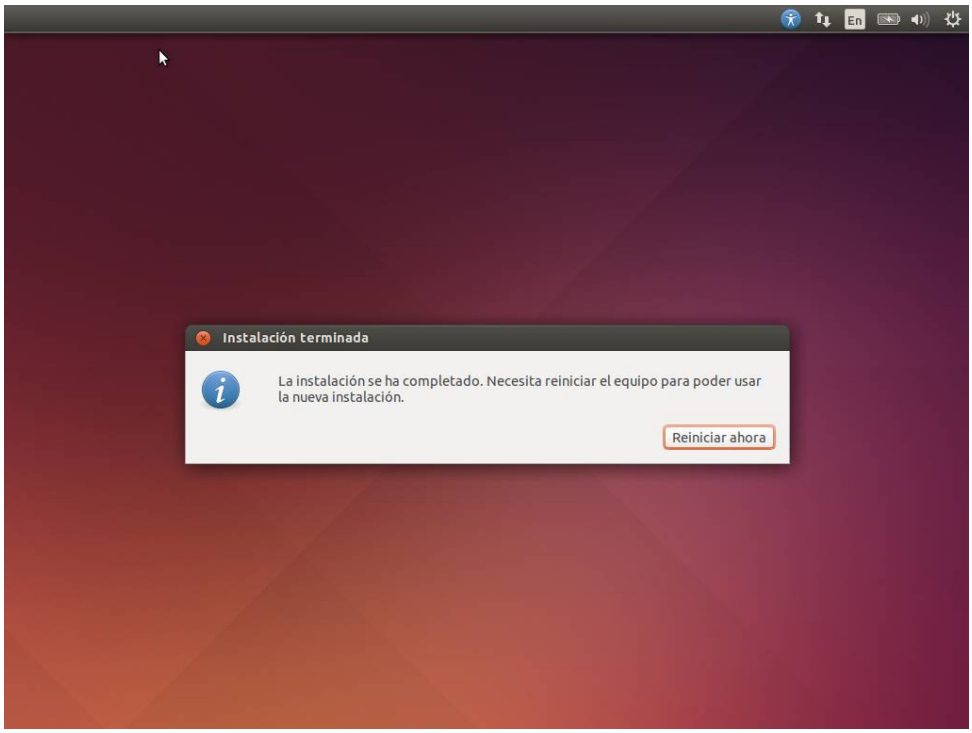


Figura 9. Instalación de Ubuntu 14.04.1 LTS

Después de su instalación, tenemos que introducir los siguientes comandos para tener la pantalla a máxima resolución posible.

```
sudo -s
apt-get install dpkg-dev
dpkg --get-selections | grep linux-headers
uname -r
apt-get install linux-headers-`uname -r` dkms build-essential
```

También habrá que habilitar el CD para ejecutar el VBoxGuestAdditions_4.3.36. Previamente, descargado desde esta URL:

http://download.virtualbox.org/virtualbox/4.3.36/VBoxGuestAdditions_4.3.36.iso

Tras esto, hay que reiniciar el equipo para que las modificaciones surtan efecto.

4.3. Aplicar parches, compilar y configurar

Las acciones realizadas aplicar características de tiempo-real al sistema operativo utilizado en este proyecto, han consistido en modificar un kernel Linux estándar (vanilla), que no tiene añadidos, con la funcionalidad aportada por cada una de las soluciones propuestas: *lowlatency* (sección 4.3.1), PREEMPT_RT (sección 4.3.2) y Xenomai (sección 4.3.3). Para este proyecto, se va a trabajar con una distribución Ubuntu 14.04.1 LTS que viene con un kernel 3.13.0-32-generic. Éste es el kernel que incluye la imagen base Linux utilizada en el Departamento de Software de CAF Power & Automation. En todas las soluciones propuestas se ha utilizado el propio intérprete de comandos:

4.3.1. Solución propuesta 1: Reducción de las latencias (*lowlatency*) del repositorio de Ubuntu

En [Ubuntu 16] está disponible esta solución que podría resultar válida.

Ubuntu, para conseguir un mejor acercamiento a un comportamiento de tiempo-real, ofrece un paquete específico de kernel con el sufijo *lowlatency*. En él incluye un kernel con modificaciones realizadas para optimizar la respuesta en tiempo que da este kernel a la hora de planificar las tareas. Es la solución más sencilla de instalar si se usa una distribución Ubuntu ya que simplemente requiere la instalación de un paquete que remplaza al kernel antiguo con el nuevo kernel. Sólo es aplicable para unas versiones determinadas de kernel, ya que los paquetes no se generan para todas las versiones. Nosotros en este proyecto hemos usado la versión 3.14.2.

Para su instalación, descargamos primero los paquetes necesarios del repositorio de Ubuntu a través de los siguientes comandos:

```
wget –c kernel.ubuntu.com/~kernel-ppa/mainline/v3.13.2-trusty/linux-headers-3.14.2-031402_3.14.2-031402.201404262053_all.deb

wget –c kernel.ubuntu.com/~kernel-ppa/mainline/v3.13.2-trusty/linux-headers-3.14.2-031402-lowlatency_3.14.2-031402.201404262053_i386.deb

wget -c kernel.ubuntu.com/~kernel-ppa/mainline/v3.13.2-trusty/linux-image-3.14.2-031402-lowlatency_3.14.2-031402.201404262053_i386.deb
```

A continuación, ejecutamos el comando que instalará los paquetes descargados sobre el sistema operativo:

```
sudo dpkg -i linux-headers-3.14.2*.deb linux-image-3.14.2*.deb
```

Con esto ya conseguimos dotar al sistema de ciertas capacidades de tiempo-real aunque sean más limitadas que las soluciones que se plantean a continuación.

4.3.2. Solución propuesta 2: PREEMPT_RT

PREEMPT_RT (que se introdujo en la sección 3.4.1.) utiliza un parche que modifica el kernel actual. Este parche aplica varios cambios en el código fuente del kernel para dotarlo de características de tiempo-real. Después de aplicar el parche, es necesario compilar el kernel e introducirlo en el equipo reemplazando el kernel antiguo. Para aplicar esta solución, la versión del parche a utilizar tiene que ser la misma versión que la asociada al kernel. El parche no está disponible para todas las versiones de kernel. Nosotros en este proyecto hemos usado la versión 3.14.2 (y 3.18.7). En el anexo A.1 se describen en detalle los pasos de instalación de PREEMPT_RT para Linux en tiempo-real.

4.3.3. Solución propuesta 3: Xenomai

Xenomai (que se introdujo en la sección 3.4.2) utiliza un pequeño kernel adicional que coopera con el kernel de Linux. Este pequeño kernel adicional que introduce Xenomai, se encarga de planificar las tareas, dando prioridad a las tareas de tiempo-real sobre el resto. Éste trata al kernel de Linux como una tarea común, de forma que el kernel de Linux no podrá bloquear las tareas de tiempo-real del sistema. Para ello, usamos la misma versión de parche asociada al kernel. Como en PREEMPT_RT, el parche no está disponible para todas las versiones de kernel. Nosotros en este proyecto hemos usado la versión 3.14.17. En el anexo A.2 se describen en detalle los pasos de instalación de Xenomai para Linux en tiempo-real.

4.4. Implementar una aplicación en tiempo-real

Una aplicación de tiempo-real sólo es capaz de operar correctamente si el sistema operativo y el hardware de debajo es capaz de proporcionar el determinismo necesario. Esto significa que una tarea de prioridad más alta puede desalojar a una tarea de prioridad más baja. Si por ejemplo, una BIOS decide usar todos los ciclos de la CPU durante un largo tiempo, ningún sistema operativo puede proporcionar garantías de latencia. El sistema entero necesita ajustarse y configurarse adecuadamente.

Los pasos a seguir para escribir programas de Linux en tiempo-real sobre PREEMPT_RT y Xenomai se explican en el anexo B.1 y B.2, respectivamente.

4.5. Herramientas de test

En el Ciclo de Vida de un sistema de tiempo-real, testear es el proceso de validación, verificación y medición de fiabilidad que asegura que el software se ejecuta como lo esperado y que satisface los requerimientos. El testeo requiere un esfuerzo, tanto en términos de coste como de tiempo. La automatización es un buen modo de reducir los esfuerzos económicos y humanos en el momento del testeo. Diversos entornos de testeo automatizados para el kernel de Linux se han propuesto en la literatura, cada uno con sus fortalezas y debilidades. Por ejemplo: IBM autobench [Bligh et al. 06]; Autotest [Admanski et al. 09]; Crackerjack; rt-tests, que usa cyclictest [CYCLICTEST 16]); LTP (*Linux Test Project*) [Yoshioka 07]; y Lachesis [Claudi et al. 11].

La utilidad escogida para efectuar las pruebas de comportamiento de las soluciones implementadas es *rt-tests* desarrollada por Thomas Gleixner y Clark Williams. Del mismo modo, también nos la sugiere usar el investigador Dr. Claudio Scordino, referente en este campo de sistemas operativos de tiempo-real.

Podemos descargar el código fuente de esta utilidad desde esta URL al escritorio:

https://www.kernel.org/pub/linux/utils/rt-tests/rt-tests-0.96.tar.gz

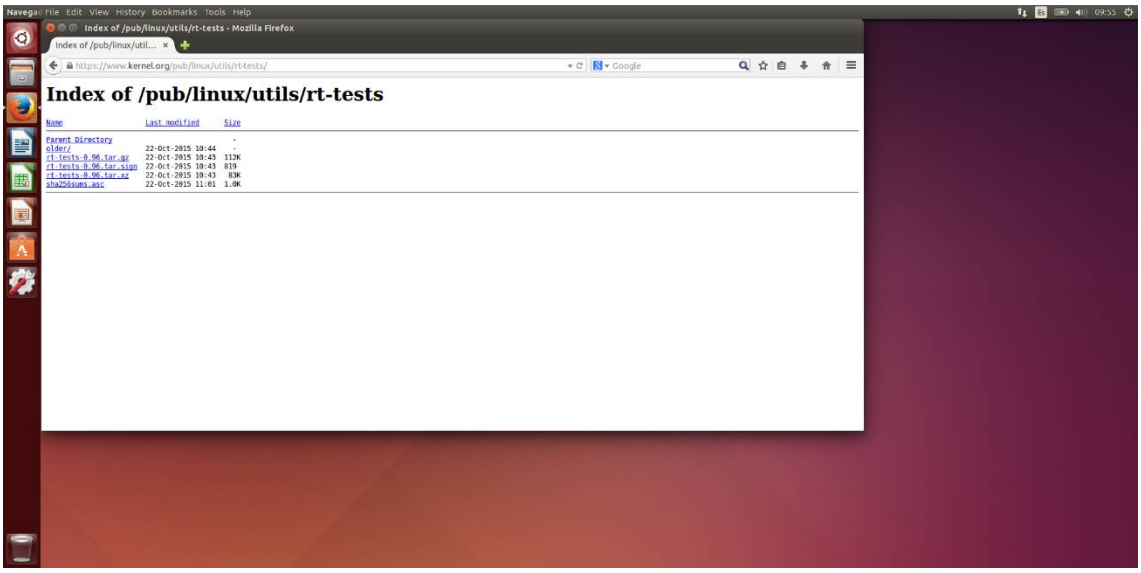


Figura 10. Página Web para descargar rt-tests

Una vez descargado el fichero comprimido, esta utilidad se puede compilar e instalar de la siguiente forma:

Primero, se descomprime el paquete en la ubicación correspondiente:

```
sudo -s
cd /usr/bin
tar xzvf /home/real-time/Escritorio/rt-tests-0.96.tar.gz
```

En este caso, `real-time` es el nombre de usuario de la máquina.

Después, instalamos las dependiencias necesarias para la compilación de la utilidad:

```
apt-get install libnuma-dev
```

Por ultimo, procedemos con la compilación y la instalación del binario resultante en el directorio donde se guardan algunos binarios añadidos por el usuario al sistema:

```
cd rt-tests-0.96
make all
cp cyclictest /usr/bin/
```

rt-tests, incluye cyclictest, que es la aplicación de test que vamos a utilizar en este caso.

4.5.1. rt-tests: cyclictest

Con *cyclictest* se abordará la medición de los tiempos de latencia involucrados en las operaciones de dormir y despertar de hilos de ejecución con una prioridad elevada. La unidad de los tiempos de respuesta es de microsegundos.

A continuación, se comentarán brevemente las opciones que ofrece *cyclictest* a la hora de la ejecución del test. El comando que se ha utilizado tiene la siguiente forma:

`./cyclictest –a –t2 –n –q –p99 –i10000 –l10000`

Opciones:

-a

Con -a ejecuta cada hilo de ejecución i en el procesador i.

-bUSEC

Envía un comando de traza de ruptura cuando la latencia > USEC. Ésta es una opción de debugeo para controlar la latencia en el parche de tiempo-real aplicado. A partir del kernel 2.6.24 esta opción sólo funciona con las siguientes configuraciones de kernel:

```
CONFIG_PREEMPT_RT=y
CONFIG_FTRACE=y
CONFIG_IRQSOFF_TRACER=y
CONFIG_PREEMPT_TRACER=y
CONFIG_SCHED_TRACER=y
CONFIG_WAKEUP_LATENCY_HIST=y
```

Esto se puede comprobar filtrando el contenido del fichero de configuración del kernel, que se puede hacer a través del siguiente comando, sustituyendo `X` por las distintas opciones a chequear:

```
grep CONFIG_X .config
```

`-n`

Con esta opción se usa el temporizador *nanosleep* en vez del *interval timers* de POSIX. En concreto, con esta opción, se llama a la función *clock_nanosleep* y no a la función estándar de POSIX.

`-q`

Ejecución silenciosa, solo se presenta por pantalla el sumario final de la ejecución.

`-t#THREADS`

Establece el número de hilos de ejecución que se lanzarán en la ejecución de *cyclictest*. En el caso de las pruebas realizadas sobre PC el número de hilos lanzados es 2.

`-d#DIST`

Distancia de los intervalos del hilo de ejecución (el valor por defecto es 500 microsegundos). Cuando cyclictest se llama con la opción -t y se crea más de un hilo de ejecución, entonces esta distancia DIST se añade al intervalo de los hilo de ejecución, tal que:

Intervalo(hilo N) = Intervalo(hilo N-1) + DIST

`-p#PRIO`

Establece la prioridad con la que será creado cada uno de los hilos de ejecución. Al primero de los hilos creados se le asignará la prioridad que acompaña al parámetro y al resto un valor decreciente a partir de la prioridad inicial (hilo 1 con prioridad N, hilo 2 con prioridad N-1, hilo 3 con prioridad N-3, etc).

`-i#INTV`

Establece la base de tiempo de cada uno de los temporizadores de los hilos de ejecución (el valor por defecto es 1000 microsegundos).

`-l#LOOPS`

Establece el número de ciclos que los hilos de ejecución deberán ejecutar en la CPU. Por defecto es 0, es decir, un bucle sin fin. Esta opción es útil para automatizar tests con un número dado de pruebas cíclicas. *cyclictest* termina cuando el primer hilo de ejecución consigue llegar al número de vueltas establecido.

Luego,

`./cyclictest –a –t2 –n –q –p99 –i10000 –l10000`

significa que un test *nanosleep* (-n) de 10 ms (-i10000) se realiza 10.000 veces (-l10000). Del mismo modo, en este test se ejecutan 2 hilos de ejecución (-t2) cada uno de ellos en un procesador distinto (-a). Al primer hilo se le asociará la prioridad 99 (-p99) de forma que el segundo obtendrá una prioridad de 98.

Ahora usando la opción de debugeo -b de *cyclictest* con el valor 3000 el resultado que nos da es el siguiente:

`./cyclictest –a –t2 –n` **`–b3000`** `–p99 –i10000 –l10000`

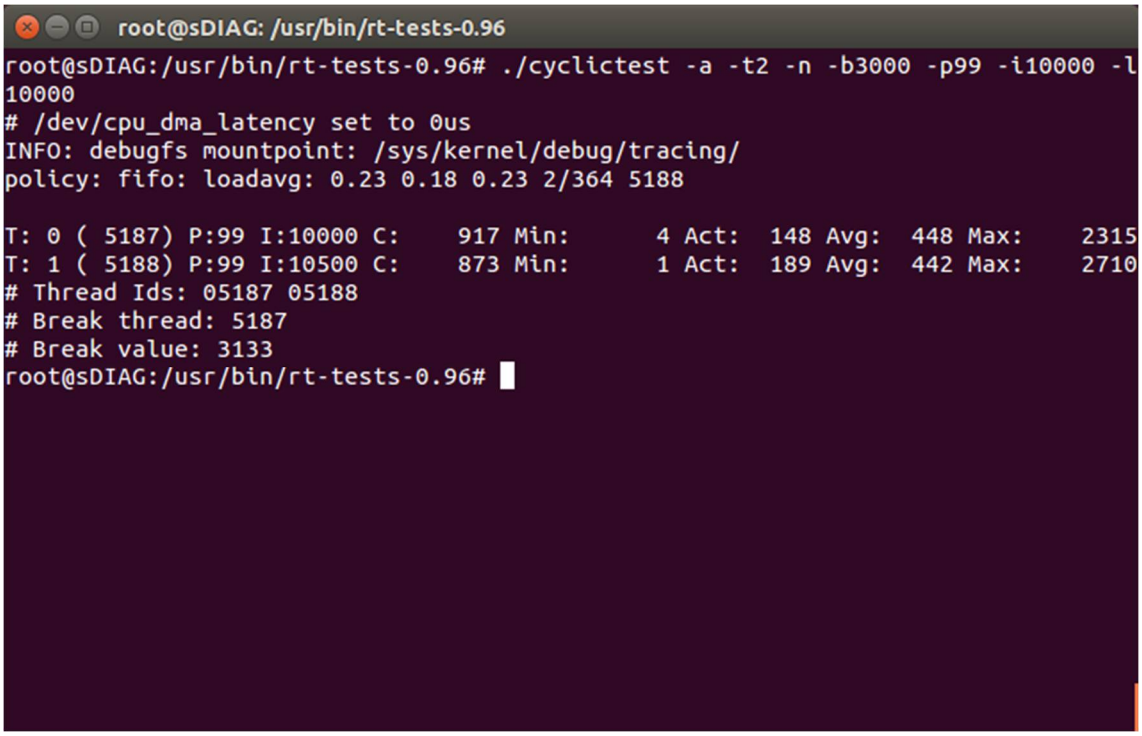


Figura 11. Cyclictest con opción de debugeo -b

La información sobre la traza de ruptura se podría ver escribiendo:

```
cat /sys/kernel/debug/tracing/trace
```

4.5.2. Otros tests de rt-tests

Además de cyclictest la utilidad rt-tests incluye también otras aplicaciones de test como: hackbench, pip_stress, pi_stress, pmqtest y signaltest [Martorell 15], pero no se han utilizado en este proyecto.

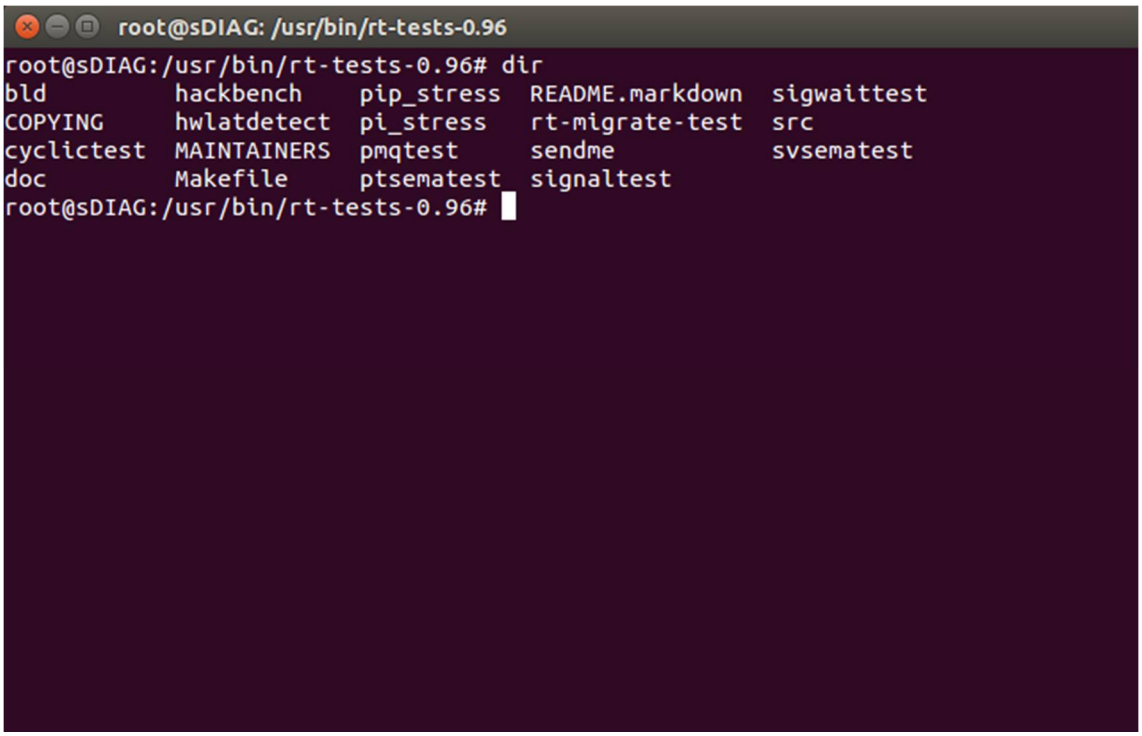


Figura 12. Estructura de ficheros de rt-tests

4.5.3. Comparativa del test de las distintas soluciones sobre PC

Una vez comentadas las opciones usadas, se procederá a mostrar los resultados obtenidos en los diferentes escenarios. La máquina PC sobre la que se han hecho los tests, y posee los distintos kernel modificados de tiempo-real, tiene las siguientes prestaciones:

```
model name: Intel(R) Core(TM)2 Duo CPU T6400 @ 2.00GHz
model name: Intel(R) Core(TM)2 Duo CPU T6400 @ 2.00GHz
cpu cores: 2
cpu cores: 2
```

Figura 13. Prestaciones del PC: obtenido tras ejecutar
`cat /proc/cpuinfo | grep "model name"; cat /proc/cpuinfo | grep "cpu cores"`

Se ha utilizado sobre un PC/Linux nativo (sin máquina virtual). Dado que hemos visto que la máquina virtual distorsiona los tiempos de latencia.

A continuación, se muestran unos resultados con los valores mínimo, promedio y máximo de los tiempos de latencia de planificación obtenidos en las diferentes pruebas realizadas. Tanto para vanilla, PREEMPT_RT, *lowlatency* y Xenomai. Y con kernel 3.13.0-32-generic, 3.14.2, 3.14.2 y 3.14.17, respectivamente. Esto es, con una versión de kernel lo más similar posible en las pruebas para favorecer una comparativa de cuál es mejor.

Para explicar un resultado de *cyclictest*, tomaremos un test, donde:

T: Es el hilo de ejecución.
P: Es la prioridad.
Min: Es el valor mínimo.
Avg: Es el valor promedio.
Max: Es el valor máximo.

- Vanilla (kernel 3.13.0-32-generic)

```
# /dev/cpu_dma_latency set to 0us
T: 0 ( 2491) P:99 I:10000 C:  10000 Min:      9 Act:   18 Avg:   18
Max:    1893
T: 1 ( 2492) P:99 I:10500 C:   9526 Min:      7 Act:   23 Avg:   17
Max:    1081
```

- **PREEMPT_RT (kernel 3.14.2)**

```
# /dev/cpu_dma_latency set to 0us
T: 0 ( 2561) P:99 I:10000 C:  10000 Min:      3 Act:    5 Avg:    5
Max:      17
T: 1 ( 2562) P:99 I:10500 C:   9524 Min:      3 Act:    4 Avg:    5
Max:      26
```

- *lowlatency* (kernel 3.14.2)

```
# /dev/cpu_dma_latency set to 0us
T: 0 ( 2513) P:99 I:10000 C:  10000 Min:      7 Act:   16 Avg:   14
Max:     100
T: 1 ( 2514) P:99 I:10500 C:   9525 Min:      6 Act:   15 Avg:   15
Max:     344
```

- Xenomai (kernel 3.14.17)

```
# /dev/cpu_dma_latency set to 0us
T: 0 ( 2834) P:99 I:10000 C:  10000 Min:       2 Act:     4 Avg:     4
Max:      69
T: 1 ( 2835) P:99 I:10500 C:   9525 Min:       2 Act:     6 Avg:     4
Max:     132
```

Recordamos, las condiciones de *cyclictest* son:

`./cyclictest –a –t2 –n –q –p99 –i10000 –l10000`

Los resultados anteriores se resumen en la siguiente tabla.

Solución propuesta	Hilo de ejecución	Valor mínimo	Valor promedio	**Valor máximo**
Vanilla kernel 3.13.0-32-generic	T: 0	9	18	1893
	T: 1	7	17	1081
PREEMPT_RT kernel 3.14.2	T: 0	3	5	**17**
	T: 1	3	5	**26**
lowlatency kernel 3.14.2	T: 0	7	14	100
	T: 1	6	15	344
Xenomai kernel 3.14.17	T: 0	2	4	69
	T: 1	2	4	132

Tabla 3. Tiempos de latencia de las distintas soluciones propuestas sobre PC

Tal y como podemos observar en la tabla, PREEMPT_RT es la solución propuesta que arroja valores de tiempos de latencia menores. Cualquiera de las soluciones propuestas mejora mucho el comportamiento comparando con vanilla. *lowlatency* incluye bastante más latencia que las otras dos soluciones (PREEMPT_RT y Xenomai) pero según vimos en la anterior sección es la más sencilla de implementar con diferencia, por lo que puede ser interesante para algunos casos. Los resultados de Xenomai están más o menos en la mitad de los de *lowlatency* y de PREEMPT_RT.

En resumen, atendiendo a los tiempos de latencia en test, podemos determinar que *a priori* la mejor opción para dotar de tiempo-real al HMI (sección 5) y al sDIAG (sección 6) debería ser PREEMPT_RT.

5. Validación del kernel de tiempo-real en HMI (interfaz hombre-máquina / interfaz de usuario)

Todo proyecto ha de ser validado a través de la realización de una serie de pruebas. Es aconsejable redactar de antemano un protocolo de pruebas, es decir, una lista de pruebas a las que se le debe someter al proyecto para considerar que funciona correctamente. Esta sección corresponde a la validación del HMI. Se ha dispuesto en el último tramo del proyecto de un dispositivo HMI. Y de acceso ocasionalmente a un banco de pruebas donde está instalado el HMI.

Sobre este equipo se ha testeado la solución PREEMPT_RT considerada, según los resultados obtenidos en la sección anterior, la mejor opción para reducir las latencias del kernel y dotarlo de características necesarias para funcionar en tiempo-real. Para realizar los tests, se han utilizado las mismas herramientas usadas en la sección anterior para testear las soluciones sobre un PC.

Figura 14. Banco de pruebas con HMI

5.1. Validación de las modificaciones sobre el kernel en la imagen del HMI (sección 2.1.)

5.1.1. rt-tests: cyclictest

Se ha lanzado *cyclictest* (sección 4.5.1.) sobre el HMI en las siguientes condiciones, siendo éstos los resultados:

```
time cyclictest –t1 –p99 –i1000 –n –l10000000 –d86400 –m –a 1
```

Para explicar un resultado de *cyclictest*, tomaremos un test, donde:

T: Es el hilo de ejecución.
P: Es la prioridad.
Min: Es el valor mínimo.
Avg: Es el valor promedio.
Max: Es el valor máximo.

- Vanilla (kernel 3.13.0-32-generic)

```
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.35 0.49 0.56 1/264 3854
T: 0 ( 2192) P:99 I:1000 C:  10000000 Min:      4 Act:   25 Avg:   18
Max:    19775

Real 166m41.630s
user 1m31.080s
sys 4m14.964s
```

  - Ídem, pero cambiando el test **-d 86400** (-d: distancia de los intervalos del hilo de ejecución), por el de defecto, no ponemos nada, esto es 500 microsegundos:

Para este caso:

```
time cyclictest –t1 –p99 –i1000 –n –l10000000 –m –a 1
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.55 0.52 0.56 1/264 5609
T: 0 ( 3951) P:99 I:1000 C:  10000000 Min:      4 Act:   16 Avg:   17
Max:    19892

Real 166m41.356s
user 1m29.032s
sys 4m5.292s
```

  - Ídem, pero con cuatro hilos de ejecución:

Para este caso:

```
time cyclictest –t4 –p99 –i1000 –n –l10000000 –m

# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.50 0.50 0.50 2/267 7439
T: 0 ( 5773) P:99 I:1000 C:  10000000 Min:      4 Act:   14 Avg:   16
Max:    19769
T: 1 ( 5774) P:99 I:1500 C:  6666253  Min:      4 Act:   7 Avg:   18
Max:    19860
T: 2 ( 5775) P:99 I:2000 C:  4999914  Min:      4 Act:   6 Avg:   18
Max:    19282
T: 3 ( 5776) P:99 I:2500 C:  4000130  Min:      4 Act:   12 Avg:   18
Max:    19429

Real 166m40.991s
user 4m7.148s
sys 7m40.932s
```

Como vemos, *cyclictest* sobre el HMI arroja latencias máximas del orden de 19 ms (0,019 s). Tal y como está, con kernel vanilla (3.13.0-32-generic). Aplicando el parche PREEMPT_RT con un kernel 3.14.2, los resultados de *cyclictest* han sido los siguientes:

- PREEMPT_RT (kernel 3.14.2)

  - Más todo FTRACE deshabilitado, las tres de ACPI deshabilitadas (*Button*, *Fan* y *Processor*) y CONFIG_CC_OPTIMIZE_FOR_SIZE habilitado el resultado es:

```
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.26 0.34 0.32 1/292 3798
T: 0 ( 1970) P:99 I:1000 C:  10000000 Min:      4 Act:   6 Avg:   7
Max:    34

Real 166m40.104s
user 0m37.436s
sys 2m20.036s
```

- Ídem, pero cambiando el test **-d 86400**, por el de defecto, no ponemos nada, esto es 500 microsegundos:

Para este caso:

```
time cyclictest –t1 –p99 –i1000 –n –l10000000 –m –a 1

# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.31 0.31 0.31 2/292 6385
T: 0 ( 4549) P:99 I:1000 C:  10000000 Min:      4 Act:   9 Avg:   7
Max:    37

Real 166m40.111s
user 0m38.040s
sys 2m18.596s
```

- Ídem, pero con cuatro hilos de ejecución:

Para este caso:

```
time cyclictest –t4 –p99 –i1000 –n –l10000000 –m

# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.19 0.29 0.31 1/295 3903
T: 0 ( 1905) P:99 I:1000 C:  10000000 Min:      4 Act:   7 Avg:   7
Max:    34
T: 1 ( 1906) P:99 I:1500 C:  6666661  Min:      4 Act:   8 Avg:   7
Max:    49
T: 2 ( 1907) P:99 I:2000 C:  4999990  Min:      5 Act:   7 Avg:   7
Max:    32
T: 3 ( 1908) P:99 I:2500 C:  3999987  Min:      4 Act:   7 Avg:   7
Max:    34

Real 166m40.119s
user 1m35.628s
sys 4m9.656s
```

Los resultados anteriores se resumen en la siguiente tabla.

Solución	Kernel	Opción -d	Nº de hilos de ejecución	Valor máximo
Vanilla	3.13.0-32-generic	86400	1	T: 0 (19775)
		500	1	T: 0 (19892)
		500	4	T: 0 (19769)
				T: 1 (19860)
				T: 2 (19282)
				T: 3 (19429)
PREEMPT_RT	3.14.2	86400	1	T: 0 (34)
		500	1	T: 0 (37)
		500	4	T: 0 (34)
				T: 1 (49)
				T: 2 (32)
				T: 3 (34)

Tabla 4. Tiempos de latencia de vanilla y PREEMPT_RT sobre HMI

Los tiempos de latencia valor máximo que presenta vanilla son importantes, del orden de 19 ms, tanto para un hilo como para cuatro hilos de ejecución. La solución PREEMPT_RT mejora estos resultados, al igual que como se vió en la sección 4 para PC. Lo valores máximo que arroja están comprendidos entre 32 y 49 microsegundos, tres órdenes de magnitud menos que vanilla. La opción -d de *cyclictest*, que hace referencia a la distancia de los intervalos del hilo de ejecución, no parece que afecte a los resultados. Se ha probado esta opción con -d 86400 y con el valor por defecto, 500 microsegundos.

5.1.2. Timon Embedded

TiMon Embedded es un proceso incluido en el equipo HMI, que permite muestrear de forma periódica el valor de las variables de la base de datos del HMI (VarDB). Este proceso se puede configurar para obtener el valor de una serie de variables con una periodicidad concreta. Cuanto menor sea la periodicidad de muestreo mayor tendrá que ser la rapidez del sistema en obtener y devolver el valor de las variables. En el anexo C se muestra una tabla con errores de lecturas de timon_emb distintas a 16 ms (0,016 s) con kernel vanilla (3.13.0-32-generic). Para este nivel de periodicidad, sería de interés aplicar el parche PREEMPT_RT de modo que ayudase a reducir las latencias y hacer que se produzcan menos errores en los tiempos de muestreo. Tras aplicar PREEMPT_RT con un kernel 3.14.2, como se ve en el mismo anexo C, las mejoras son notables. Las variaciones en aquellos casos que no es de 16 ms es de 1 ms (0,001 s). Mientras que con el kernel vanilla la variación máxima en los numerosos casos que se producen es de hasta 6 ms (0,016 + 0,006 = 0,022 s).

6. Validación del kernel de tiempo-real en sDIAG

6.1. Validación de las modificaciones sobre el kernel en la imagen del sDIAG (sección 2.2.)

Sobre este equipo se han testeado las soluciones PREEMPT_RT y *lowlatency* planteadas en la sección 4, como posibles soluciones para reducir las latencias del kernel y dotarlo de características necesarias para funcionar en tiempo-real. Para realizar los tests, se han utilizado las mismas herramientas usadas en la sección 4 para testear las soluciones sobre un PC.

6.1.1. rt-tests: cyclictest

Se ha lanzado *cyclictest* (sección 4.5.1.) sobre el sDIAG (placa de evaluación) en las siguientes condiciones, siendo éstos los resultados:

`time cyclictest –t1 –p99 –i1000 –n –l10000000 –d86400 –m –a 1`

Para explicar un resultado de *cyclictest*, tomaremos un test, donde:

T: Es el hilo de ejecución.
P: Es la prioridad.
Min: Es el valor mínimo.
Avg: Es el valor promedio.
Max: Es el valor máximo.

- Vanilla (kernel 3.13.0-32-generic)

```
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.18 0.11 0.12 1/93 2675
T: 0 ( 1761) P:99 I:1000 C:  10000000 Min:      5 Act:   26 Avg:   32
Max:    499

Real 166m40.090s
user 1m49.320s
sys 13m35.844s
```

  - Ídem, pero con cuatro hilos de ejecución y cambiando el test **-d 86400** (-d: distancia de los intervalos del hilo de ejecución), por el de defecto, no ponemos nada, esto es 500 microsegundos:

Para este caso:
`time cyclictest –t4 –p99 –i1000 –n –l10000000 –m`

```
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.80 0.84 0.92 1/96 2335
T: 0 (1412) P:99 I:1000 C:  10000000 Min:      4 Act:   23 Avg:   32
Max:    51645
T: 0 (1413) P:99 I:1500 C:  6666693  Min:      4 Act:   24 Avg:   30
Max:    51607
T: 0 (1414) P:99 I:2000 C:  5000082  Min:      4 Act:   47 Avg:   24
Max:    14766
T: 0 (1415) P:99 I:2500 C:  4000014  Min:      5 Act:   25 Avg:   26
Max:    51179

Real 166m40.318s
user 3m19.512s
sys 23m55.524s
```

- PREEMPT_RT (kernel 3.18.7)

    - Más todo FTRACE deshabilitado, las tres de ACPI deshabilitadas (*Button*, *Fan* y *Processor*) y CONFIG_CC_OPTIMIZE_FOR_SIZE habilitado el resultado es:

```
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.00 0.02 0.05 1/119 2334
T: 0 ( 1427) P:99 I:1000 C:  10000000 Min:      5 Act:   9 Avg:   7
Max:    48
Real 166m40.109s
user 0m23.308s
sys 5m8.144s
```

    - Ídem, pero cambiando el test **-d 86400**, por el de defecto, no ponemos nada, esto es 500 microsegundos:

Para este caso:

```
time cyclictest –t1 –p99 –i1000 –n –l10000000 –m –a 1

# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.09 0.07 0.05 1/119 7997
T: 0 ( 7067) P:99 I:1000 C:  10000000 Min:      5 Act:   8 Avg:   7
Max:    38

Real 166m40.110s
user 0m25.432s
sys 5m45.044s
```

    - Ídem, pero con cuatro hilos de ejecución:

Para este caso:

```
time cyclictest –t4 –p99 –i1000 –n –l10000000 –m

# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.17 0.10 0.06 1/112 31966
T: 0 (31029) P:99 I:1000 C:  10000000 Min:      5 Act:   7 Avg:   7
Max:    33
T: 0 (31030) P:99 I:1500 C:  6666659  Min:      5 Act:   9 Avg:   8
Max:    55
T: 0 (31031) P:99 I:2000 C:  4999987  Min:      6 Act:   7 Avg:   8
Max:    37
T: 0 (31032) P:99 I:2500 C:  3999983  Min:      5 Act:   7 Avg:   8
Max:    28

Real 166m40.112s
user 0m43.560s
sys 14m40.648s
```

- *lowlatency* (kernel 3.14.2)

```
time cyclictest –t1 –p99 –i1000 –n –l10000000 –d86400 –m –a 1

# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.07 0.05 0.05 1/97 2328
T: 0 ( 1405) P:99 I:1000 C:  10000000 Min:      4 Act:   6 Avg:   7
Max:    103

Real 166m40.081s
user 0m28.583s
sys 4m22.611s
```

- Ídem, pero con cuatro hilos de ejecución y cambiando el test **-d 86400**, por el de defecto, no ponemos nada, esto es 500 microsegundos:

Para este caso:
time cyclictest –t4 –p99 –i1000 –n –l10000000 –m

```
# /dev/cpu_dma_latency set to 0us
Policy: fifo: loadavg: 0.15 0.14 0.14 1/100 2333
T: 0 (1403) P:99 I:1000 C:  10000000 Min:      4 Act:   7 Avg:   6
Max:    81
T: 0 (1404) P:99 I:1500 C:  6666670  Min:      4 Act:   8 Avg:   7
Max:    92
T: 0 (1405) P:99 I:2000 C:  5000000  Min:      4 Act:   9 Avg:   6
Max:    95
T: 0 (1406) P:99 I:2500 C:  3999998  Min:      4 Act:   7 Avg:   6
Max:    88
Real 166m40.090s
user 0m38.964s
sys 10m21.843s
```

Los resultados anteriores se resumen en la siguiente tabla.

Solución	Kernel	Opción -d	Nº de hilos de ejecución	Valor máximo
Vanilla	3.13.0-32-generic	86400	1	T: 0 (499)
		500	4	T: 0 (51645)
				T: 1 (51607)
				T: 2 (14766)
				T: 3 (51179)
PREEMPT_RT	3.18.7	86400	1	T: 0 (48)
		500	1	T: 0 (38)
		500	4	T: 0 (33)
				T: 1 (55)
				T: 2 (37)
				T: 3 (28)
lowlatency	3.14.2	86400	1	T: 0 (103)
		500	4	T: 0 (81)
				T: 1 (92)
				T: 2 (95)
				T: 3 (88)

Tabla 5. Tiempos de latencia valor máximo de vanilla, PREEMPT_RT y *lowlatency* sobre sDIAG

Como conclusión, los tiempos de latencia valor máximo que presenta vanilla son importantes, del orden de 51 ms para cuatro hilos de ejecución. La solución PREEMPT_RT mejora estos resultados, al igual que como se vió tanto para PC, como para HMI. Los valores máximos que arroja están comprendidos entre 28 y 55 microsegundos, tres órdenes de magnitud menos que vanilla. Aunque los mejores resultados corresponden a las soluciones PREEMPT_RT, también podemos ver que *lowlatency* reduce las latencias de forma considerable con muy poco esfuerzo de implementación por lo que sigue siendo una opción interesante para los casos en los que la mejora que introduce es suficiente. La opción -d de *cyclictest*, que hace referencia a la distancia de los intervalos del hilo de ejecución, no parece que afecte a los resultados. Se ha probado esta opción con -d 86400 y con el valor por defecto, 500 microsegundos.

7. Mejoras realizadas en las aplicaciones de los equipos HMI y sDIAG

Una vez modificado satisfactoriamente el kernel de las imágenes del HMI y del sDIAG para dotarlo de capacidades de tiempo-real, se han añadido las siguientes funcionalidades de cara a mejorar el sistema operativo utilizado por ambos equipos y completar la solución de tiempo-real haciéndola más robusta y usable.

7.1. Optimización del proceso de actualización de Linux a través de dispositivo USB autoarrancable.

Esta optimización consiste en proporcionar un mecanismo para poder aplicar las soluciones propuestas a los equipos de forma sencilla. El mecanismo que se propone es que la imagen a la que se aplica la solución de tiempo-real se inserte en un dispositivo USB autoarrancable, que nada más arrancar ejecuta un comando que vuelca la imagen de tiempo-real a la Compact Flash que contiene el sistema operativo del equipo. De esta forma, lo único que hay que hacer para actualizar la imagen antigua por la imagen de tiempo-real es insertar el dispositivo en el puerto USB del equipo y reiniciar el equipo.

En el anexo D se describe en detalle la implementación de esta mejora.

7.2. Robustecimiento de Linux ante corrupciones de memoria

En un tren no siempre se controlan los apagados del sistema. Muchas veces “se tira de los térmicos” de los distintos sistemas para realizar el apagado por lo que a los equipos se les quita la alimentación sin tener en cuenta el estado en el que están. Este tipo de acciones pueden provocar corrupciones en el sistema operativo. Si algún proceso está realizando escrituras en el momento del apagado, es muy probable que la memoria en la que se está escribiendo quede corrupta. Esto es, con los datos guardados a medias y de forma incorrecta.

El hecho de poner las particiones de la imagen (usaremos la del sDIAG) en modo sólo lectura puede evitar este tipo de corrupciones de memoria. Se evita que los procesos escriban en la memoria correspondiente a estas particiones. Los pasos para hacer que las particiones se monten en modo sólo lectura son los siguientes:

Escribimos en el intérprete de comandos y como *root*, para poder editar el fichero que contiene la tabla de particiones:

```
nano /etc/fstab
```

Añadimos la opción `,ro` en la columna `<options>` de cada partición. Reiniciamos el quipo (`reboot`). Y accedemos como *root*. Vemos que da los siguientes errores:

```
Unable to setup logging. [Errno 30] Read-only file system:
'/var/log/landscape/sysinfo.log'
run-parts: /etc/update-motd.d//50-landscape-sysinfo exited with return
code 1

/usr/lib/update-notifier/update-motd-fsck-at-reboot: 33:
/usr/lib/update-notifier/update-motd-fsck-at-reboot: cannot create
/var/lib/update-notifier/fsck-at-reboot: Read-only file system
```

Como vemos, algunos procesos de sistema que requieren escribir en alguna de las particiones muestran errores. Para poder solventar estos errores, montamos la partición que contiene el fichero anterior en modo de lectura/escritura y editamos de nuevo el fichero.

Ejecutamos:

```
mount -o remount, rw /
nano /etc/fstab
```

Montamos en RAM el directorio `/tmp` de la imagen, para que tengamos alguna ubicación donde poder escribir. Aunque sea en memoria volátil, lo cual significa que en cada arranque se perderán los cambios realizados en esta ubicación. Para ello, añadimos en `/etc/fstab` las siguientes líneas:

<file system>	<mount point>	<type>	<options>	<dump>	<pass>
tmpfs	/tmp	tmpfs	defaults	0	0
proc	/proc	proc	nodev,noexec,nosuid	0	0

También quitamos `,ro` en todas las filas de la columna `<options>`. Reiniciamos (`reboot`).

- Para quitar el mensaje de error asociado a: `/var/log/landscape/sysinfo.log` [LANDSCAPE 16], simplemente desinstalamos este paquete que no es necesario para los equipos utilizados en este proyecto:

```
apt-get remove landscape-common
apt-get purge landscape-common
```

- Para quitar el mensaje de error asociado a: `/usr/lib/update-notifier/update-motd-fsck-at-reboot`, hacemos que a través de un enlace simbólico, el fichero apunte a la memoria volátil en vez de intentar crearlo en una de las particiones de memoria de la CF.

```
cd /var/lib/update-notifier
rm fsck-at-reboot
ln -s /tmp/fsck-at-reboot fsck-at-reboot
ls –ls
```

- Para quitar el mensaje de error asociado a: `/var/lib/ubuntu-release-upgrader/release-upgrade-available`

Editamos el *script* de nombre `/usr/lib/ubuntu-release-upgrader/release-upgrade-motd` y cambiamos la ruta `/var/lib/ubuntu-release-upgrader/release-upgrade-available` de la variable *stamp* por esta otra: **`/tmp`**`/release-upgrade-available` de forma que el fichero donde se intente escribir esté en memoria volátil.

Ahora hacemos, `nano /etc/fstab`

Escribimos `,ro` en todas las filas de la columna `<options>`. Reiniciamos (`reboot`). Verificamos que ya no da los errores. Para ello, escribimos:

```
sudo -s
run-parts /etc/update-motd.d/ | sudo tee /var/run/motd.dynamic
```

7.3. Poner imagen corporativa en el arranque

En el caso del HMI, para que el arranque sea más atractivo y, aparezca la menor cantidad posible de las trazas de arranque del sistema, se puede utilizar una imagen que se muestre durante este tiempo de arranque. Para poner la imagen tanto en el arranque como en el cierre (esto se ve cuando se ejecuta `sudo reboot` en línea de comandos de *shell*), se ha creado el tema `caf_pa` para el servicio de plymouth de Linux.

Éste consiste en una carpeta de nombre `caf_pa` con los siguientes ficheros:

- caf_pa.plymouth

```
[Plymouth Theme]
Name=CAF P&A
Description=CAF P&A theme
ModuleName=script
[script]
ImageDir=/lib/plymouth/themes/caf_pa
ScriptFile=/lib/plymouth/themes/caf_pa/caf_pa.script
```

- caf_pa.script

```
Window.SetBackgroundTopColor(1.00, 1.00, 1.00);
Window.SetBackgroundBottomColor(1.00, 1.00, 1.00);

logo_filename="logo.png";
logo.image=Image(logo_filename);
logo.sprite=Sprite();
logo.sprite.SetImage(logo.image);
logo.width=logo.image.GetWidth();
logo.height=logo.image.GetHeight();
logo.x=Window.GetX() + Window.GetWidth()/2 -logo.width/2;
logo.y=Window.GetY() + Window.GetHeight()/2 - logo.height/2;
logo.z=1000;
logo.sprite.SetX(logo.x);
logo.sprite.SetY(logo.y);
logo.sprite.SetZ(logo.z);
logo.sprite.SetOpacity(1);
```

- logo.png

Se trata de una imagen .png de 800 píxeles de ancho por 480 píxeles de alto, con 32 bits de profundidad.

Figura 15. Imagen corporativa

La secuencia de comandos para aplicar esta imagen ha sido la siguiente:

```
scp -r caf_pa user@172.20.154.252:/tmp
cd /tmp
mv caf_pa /lib/plymouth/themes/
cd /lib/plymouth/themes/
```

```
cp caf_pa/caf_pa.plymouth ../default.plymouth
update-initramfs –u
```

Por último, para ver los cambios escribimos:

```
reboot
```

7.4. Crear imagen base para sDIAG a partir de la última versión de Ubuntu Server (16.04)

En la actualidad, la imagen como sistema operativo que se utiliza en los equipos embarcados del tren de CAF Power & Automation se basa en un Ubuntu Server 14.01.1. Esta imagen base se crea a partir de una distribución Ubuntu Server y un programa *script* llamado `install_packages.sh` junto con ficheros de configuración para los paquetes, que se encargan de aplicar la personalización de la imagen requerida por CAF Power & Automation.

Esta mejora ha consistido en crear una imagen base a partir de la última versión de Ubuntu Server 16.04 disponible en el repositorio oficial. Para ello, se ha adaptado el programa *script* `install_packages.sh` para esta nueva versión, ya que algunas partes del código no eran compatibles con la nueva versión 16.04. El fichero resultante se llama `install_packages_1604.sh`. La imagen base de Ubuntu Server 16.04 estable se ha realizado mediante PC/Windows y máquina virtual.

Se ha creado `image_base_1604.zip`. Éste contiene la siguiente estructura de directorios y ficheros:

- Directorio: `\built_packages`
- Directorio: `\files`
- Fichero: `install_packages_1604.sh`

Esta mejora realizada permite crear una imagen base de Ubuntu Server 16.04 para un equipo. Los pasos necesarios a seguir son:

1. Descargar Ubuntu Server 16.04 del repositorio oficial. Disponible en: http://releases.ubuntu.com/16.04/ubuntu-16.04-server-i386.iso
2. Instalar Ubuntu Server 16.04. Por ejemplo, siguiendo los pasos de instalación de [CAFPOWER -] para Ubuntu Server 14.04.01.
3. Arrancar equipo y abrir sesión.
4. Copiar `image_base_1604.zip` a carpeta local de Ubuntu Server 16.04.
5. Descomprimir `image_base_1604.zip` de la carpeta local.
6. Acceder al directorio `image_base_1604` creado tras descomprimir el fichero .zip.
7. Ejecutar `install_packages_1604.sh` con privilegios de *root*. Este paso requerirá de una conexión a Internet para la instalación de paquetes.

8. Conclusiones y líneas futuras

Cada vez más sistemas de nuestro entorno (ferroviario, automoción, aeroespacial, máquina-herramienta) requieren de prestaciones de tiempo-real para el control de los dispositivos. En muchos casos, se encuentran empotrados. El sector que nos ha ocupado en este proyecto es el ferroviario, donde la empresa CAF Power & Automation sienta su modelo de negocio. En el Departamento de Software había una necesidad de dotar de tiempo-real al software (empotrado) de los equipos HMI y sDIAG del TCMS COSMOS. Estos equipos están basados actualmente en Linux. En una primera fase, se ha realizado un estado del arte tecnológico y se han planteado distintas soluciones para el problema propuesto. Posteriormente, se han implementado las soluciones más interesantes tanto en un entorno PC como en equipo sDIAG, que han servido para luego llevarlas también a un banco de pruebas del tren que incluye un equipo HMI. Todas las soluciones implementadas se han validado con herramientas de test adecuadas para pruebas de tiempo-real que permiten, por ejemplo, ver tiempos de latencia. Por último, se ha elegido la solución que se ha considerado más adecuada para los equipos antes mencionados y se han visto los resultados obtenidos con procesos y herramientas reales (TiMon Embedded y TiMon) utilizados en estos equipos. Las mejoras realizadas sobre el sistema operativo de éstos van desde modificaciones del kernel hasta modificaciones a nivel de aplicación.

Entre las distintas soluciones planteadas para la modificación a Linux en tiempo-real, PREEMPT_RT ha sido la solución elegida debido a su buena relación resultados/esfuerzo de implementación. Por ejemplo, las aplicaciones de tiempo-real escritas sobre PREEMPT_RT sólo requieren seguir unas buenas prácticas de programación [Scordino 16]. PREEMPT_RT no proporciona un API de programación a nivel de usuario como Xenomai. Hemos visto justificada la modificación del kernel con PREEMPT_RT en la imagen actual del HMI y sDIAG. A medida que mejore la versión del kernel Linux y su correspondiente versión de PREEMPT_RT a aplicar (se ha usado la 3.14.2) habrá mayores mejoras en los resultados. Y se podría reducir más la granularidad. Se prevé continuar con el protocolo de validación interno como paso previo a su posible puesta en producción (en los trenes).

Como líneas futuras y complemento a este trabajo, se plantea profundizar en otros enfoques prometedores de Linux en tiempo-real como pueda ser Xenomai. Por ejemplo, en la implementación de aplicaciones en tiempo-real sobre esta arquitectura. Sería útil hacer un *benchmark* de varios de estos enfoques, aunque hemos visto que cada uno de ellos tienen su propia herramienta de testeo. Por ejemplo, PREEMPT_RT utiliza cyclictest y Xenomai utiliza xeno-test. Si bien, en la documentación consultada a lo largo de este proyecto, como son los artículos científicos hay líneas de trabajo por crear un *testsuite* homogéneo que favorezca el *benchmark* de las distintas soluciones. Esto se corrobora con iniciativas como Lachesis [Claudi et al. 11]. Del mismo modo, se podría estrechar relaciones con la comunidad científica de sistemas operativos de tiempo-real con algunos de los cuáles, ya hemos contactado en el transcurso del proyecto. Hemos visto que mucho conocimiento en el área reside en grupos de investigación (de universidades o centros tecnológicos como el *Max-Planck-Institut für Softwaresysteme* (Alemania)). O a través de empresas consultoras (Evidence de Italia, Feabhas de Reino Unido, Free Electrons de Francia o BIS de Polonia) que imparten cursos de formación para profesionales. Algunos de estos cursos se citan a continuación:

- *"Developing for Embedded Linux"*. Feabhas de Reino Unido.
  Información en: https://www.feabhas.com/content/developing-embedded-linux-2

- *"Embedded Linux system development training"*. Free Electrons de Francia.
  Información en:
  http://free-electrons.com/doc/training/embedded-linux/embedded-linux-agenda.pdf

- *"Real-Time Linux in Industrial Appliances"*. BIS de Polonia.
  Información en: http://bis-linux.com/real_time_linux

- "Programación Linux en Tiempo Real" (a distancia). Escuela Politécnica Superior de la Universidad Autónoma de Madrid.
  Información en: http://electratraining.org/2016/linux-tiempo-real/

Por último, este documento recoge también la investigación básica que se ha realizado. Y con el fin de incrementar sobre todo el conocimiento de los principios fundamentales de tiempo-real y Linux en tiempo-real, del Departamento de Software de CAF Power & Automation. Junto a este documento y el anexo E de información adicional, un CD recopilatorio con las publicaciones científicas (artículos y tesis doctorales) consultadas en el área se deja en el Departamento.

9. Glosario

API	En Inglés, *Application Programming Interface*. Es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para utilizarse por otro software como una capa de abstracción.
Benchmark	Comparativa.
Conocimiento	Hechos o información adquiridos por una persona a través de la experiencia o la educación.
Desalojar	Expulsar. Sinónimo utilizado.
Expulsar	Desalojar. Sinónimo utilizado.
Expulsión	Desalojo. Sinónimo utilizado.
Expulsor	No apropiativo.
Hilo de ejecución	Es la unidad de procesamiento más pequeña que se puede planificar por un sistema operativo. La creación de un nuevo hilo es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente).
Latencia	Es el tiempo que toma desde que ocurre un evento y el inicio de la acción que responderá al evento.
Mutex	Es un semáforo binario que se usa en un sistema para asegurar acceso compartido al mismo recurso.
POSIX	Interfaz estándar del sistema operativo y el entorno, incluyendo un intérprete de comandos (o *shell*), y programas de utilidades comunes para apoyar la portabilidad de las aplicaciones a nivel de código fuente.
Proceso	Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados.
Script	Es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.
Spinlock	En tiempo-real, cerrojo de espera activa.
Target	Objetivo.
TCMS	En Inglés, *Train Control and Monitoring System*. Sistema de control y monitorización de tren.

10. Bibliografía

Todas las referencias bibliográficas están disponibles a fecha de Junio de 2016.

[Admanski et al. 09] J. Admanski y S. Howard. *"Autotest-Testing the Untestable"*. Proceedings of the Linux Symposium, 2009.
Disponible en: https://www.kernel.org/doc/ols/2009/ols2009-pages-9-18.pdf

[ADVENAGE 16] ADVENAGE. *"How To determine Linux Kernel Timer Interrupt Frequency"*, 2016.
Disponible en: http://www.advenage.com/topics/linux-timer-interrupt-frequency.php

[BC3912 16] GERSYS GmbH. *"BC3912 User Manual"*. Issue: B – Febrero 2016.

[Bligh et al. 06] M. Bligh y A. Whitcroft. *"Fully Automated Testing of the Linux kernel"*. Proceedings of the Linux Symposium, vol. 1, 2006, páginas 113-125.
Disponible en: http://www.landley.net/kdocs/ols/2006/ols2006v1-pages-113-126.pdf

[Bolado 16] J. A. Bolado. Información interna. CAF Power & Automation, 25 Abril 2016.

[Brandenburg et al. 07] B. Brandenburg, A. Block, J. Calandrino, U. Devi, H. Leontyev y J. Anderson. *"LITMUSRT: A status report"*. In Proceedings of the 9th real-time Linux workshop, páginas 107-123, 2007.

[Buttazzo et al. 11] G. Buttazzo. *"Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications"*. Springer. Tercera edición, 2011. ISBN 978-1-4614-0676-1.

[CAFPOWER 16] CAF Power & Automation. "COSMOS: Soluciones de Control y Monitorización de Tren (TCMS)". Página corporativa, 2016.
http://www.cafpower.com/es/sistemas-ferroviarios-informacion-comunicaciones/cosmos-sistema-tcms

[CAFPOWER -] CAF Power & Automation. "Instalación de Ubuntu-Server 14.04.01". Documento interno.

[Claudi et al. 11] A. Claudi y A.F. Dragoni. *"Testing Linux-based real-time systems: Lachesis"*. IEEE International Conference on Service_Oriented Computing and Applications (SOCA), Irvine (California, USA), 12-14 Diciembre 2011. Páginas 1-8.

[conga-CEVAL 16] http://www.congatec.com/us/products/accessories/conga-ceval.html

[CYCLICTEST 16] https://rt.wiki.kernel.org/index.php/Cyclictest

[EEUU 14] US 8,712,611 B2 (número de patente): *"Computerized on-board system for controlling a train"*. Patente de EE.UU, 29 Abril 2014.
https://patentimages.storage.googleapis.com/pdfs/US8712611.pdf

[Emde 09] Emde, C. *"Enable real-time capabilities of the mainline kernel"*. OSADL (*Open Source Automation Development Lab*), 7 Noviembre 2009.
Disponible en: https://www.osadl.org/fileadmin/dam/interface/docbook/howtos/kernel-rt.pdf

[EVIDENCE 16] Evidence. *"SCHED_DEADLINE in the Linux 3.14 kernel"*, 2016.
Disponible en: http://www.evidence.eu.com/sched_deadline.html

[Fayyad-Kazan et al. 14] Fayyad-Kazan, H., Perneel, L., y Timmerman, M. *"Linux PREEMPT-RT v2.6.33 versus v3.6.6 - Better or worse for real-time applications?"*. ACM SIGBED Review - Special Issue on the 3rd Embedded Operating System Workshop (EWiLi 2013). ACM New York (USA), Febrero 2014, Volumen 11 Tema 1. Páginas 26-31.
Disponible en:
http://sigbed.seas.upenn.edu/archives/2014-02/ewili13_submission_1.pdf

[Fayyad-Kazan et al. 13] Fayyad-Kazan, H., Perneel, L., y Timmerman, M. *"Linux PREEMPT-RT vs. Commercial RTOSs: How Big is the Performance Gap?"*. GSTF Journal on Computing (JoC) Volumen 3 Número 1, Marzo 2013.
Disponible en:
http://dl6.globalstf.org/index.php/joc/article/download/555/571

[FREE 16] Free Electrons. *"Embedded Linux system development training"*. Apuntes de este curso de formación impartido por esta empresa consultora francesa. 9 Junio 2016.
Disponible en:
http://free-electrons.com/doc/training/embedded-linux/

[Gleixner et al. 06] Gleixner, T., y Niehaus, D. *"Hrtimers and beyond: Transforming the linux time subsystems"*. In: Linux Symposium. Volumen 1. Citeseer, páginas 333-346.
Disponible en: http://www.landley.net/kdocs/ols/2006/ols2006v1-pages-333-346.pdf

[Hallinan 11] Hallinan, C. *"Embedded Linux Primer: a practical real-world approach"*. Pearson Education. Segunda edición, 2011. ISBN 978-0-13-701783-6.

[Lafuente 09] Lafuente, A. "Sistemas Operativos II". Departamento de Arquitectura y Tecnología de Computadores, Facultad de Informática de Donostia-San Sebastián (País Vasco). Universidad del País Vasco. Apuntes de las clases magistrales, curso 2008/2009.

[LANDSCAPE 16] http://manpages.ubuntu.com/manpages//raring/en/man1/landscape-sysinfo.1.html

[Larson 01] P. Larson. *"Testing Linux with the Linux Test Project"*. Ottawa Linux Symposium, 2001, página 265.
Disponible en: https://www.kernel.org/doc/ols/2002/ols2002-pages-265-273.pdf

[Lelli 14] Lelli, J. *"SCHED_DEADLINE – How to use it"*. Retis Lab. *Institute of Communication, Information and Perception Technologies. Scuola Superiore Sant'Anna*. Pisa (Italia), 26 Junio 2014.
Disponible en:
http://retis.sssup.it/~jlelli/talks/rts-like14/SCHED_DEADLINE.pdf

[Martorell 15] Martorell, X. *"Suport per temps real"*. Conceptes Avançats de Sistemes Operatius. Dept. d'Arquitectura de Computadors, Facultat d'Informàtica de Barcelona (Catalunya). Universitat Politècnica de Catalunya. Apuntes de las clases magistrales, curso 2015/2016.

[MontaVista 16] http://www.mvista.com/solution-real-time.html

[Neil 14] Neil, G. *"On board train control and monitoring systems"*. Professional Development Course on Electric Traction Systems, IET 13th. London, 3-6 Noviembre 2014. Páginas 1-27.

[Petazzoni -] Petazzoni, T. *"Linux and real-time"*. Transparencias de este curso de formación impartido por Free Electrons (Francia).

[Plymouth 16] https://wiki.ubuntu.com/Plymouth

[Priya et al. 14] K. Priya y A. K. Srivastava. *"Real Time Operating Systems: A Review"*. International Journal for Scientific Research & Development, volumen 2, Tema 01, 2014.

[Regnier et al. 08] Regnier, P., Lima, G., y Barreto, L. *"Evaluation of Interrupt Handling Timeliness in Real-Time Linux Operating Systems"*. ACM SIGOPS Operating Systems, páginas 52-63, Octubre 2008.
Disponible en: http://homes.dcc.ufba.br/~pregnier/pub/LinuxRT08.pdf

[SCHEDULING 16] http://ck.wikia.com/wiki/SchedulingPolicies

[Scordino 16] C.Scordino. Información escrita en el foro. Stack Overflow, 3 Marzo 2016.

Disponible en:
http://stackoverflow.com/questions/35766811/build-an-rt-application-using-preempt-rt

[Scordino et al. 06] C. Scordino y G. Lipari. *"Linux and real time: Current approaches and future opportunities"*. ANIPLA International Congress, Roma, 2006.
Disponible en: http://retis.sssup.it/~lipari/papers/ANIPLA_scordino_lipari.pdf

[SWISSBIT 16] https://www.rutronik24.com/product/swissbit/sfca4096h2bv4to-i-ms-226-std/3582149.html

[TimeSys 16] http://www.timesys.com

[TiMon 09] CAF Power & Automation. "COSMOS SISTEMA DE DIAGNOSIS Manual de usuario de TiMon". Décima edición, 24 Marzo 2009.

[TRANSCEND 16] http://uk.rs-online.com/web/p/compact-flash-cards/6843771/

[Ubuntu 16] http://kernel.ubuntu.com/~kernel-ppa/mainline/v3.13.2-trusty/

[XenomaiAPINativa 06] Xenomai. "A Tour of the Native API". RevC, 20 Marzo 2006.
Disponible en:
https://xenomai.org/documentation/branches/v2.4.x/pdf/Native-API-Tour-rev-C.pdf

[XenomaiAPIPOSIX 14] http://xenomai.org/2014/08/porting-a-linux-application-to-xenomai-dual-kernel/

[XenomaiCódigoFuente 16] Xenomai Linux Exercises. Robot Lab. Computing Science Department. Institute for Computing and Information Sciences. Radboud University Nijmegen. Nijmegen (Países Bajos).
Disponible en:
http://www.cs.ru.nl/lab/xenomai/exercises/

[Xu et al. 10] H. Xu y R. Tang. *"Study and improvements for the real-time performance of Linux kernel"*. 3rd International Conference on Biomedical Engineering and Informatics (BMEI 2010), 16-18 Octubre 2010.

[Yoshioka 07] H. Yoshioka. *"Regression Test Framework and Kernel Execution Coverage"*. Proceedings of the Linux Symposium, 2007, páginas 285-296.
Disponible en: http://landley.net/kdocs/ols/2007/ols2007v2-pages-285-296.pdf

[Zhang et al. 06] Zhang, G., Chen, L., y Yao, A. *"Study and Comparison of the RTHAL-Based and ADEOS-Based RTAI Real-time Solutions for Linux"*. In First International MultySymposiums on Computer and Computational Sciences, IMSCCS, páginas 771-775, Junio 2006.

[Zuriarrain 15] Zuriarrain, I. *"COSMOS Diagnosis System Design Concept"*. Documento interno. CAF Power & Automation, 1 Septiembre 2015.

A. Anexo: Pasos de instalación de las soluciones propuestas para Linux en tiempo-real

A.1. Solución propuesta 2: PREEMPT_RT

Para asegurar que se aplica bien el parche PREEMPT_RT, tenemos que usar la misma versión de parche asociada al kernel. En nuestro caso, usaremos el parche 3.14.2 para el kernel 3.14.2.

Partimos de este estado en el que ambos ficheros están descargados en el escritorio:

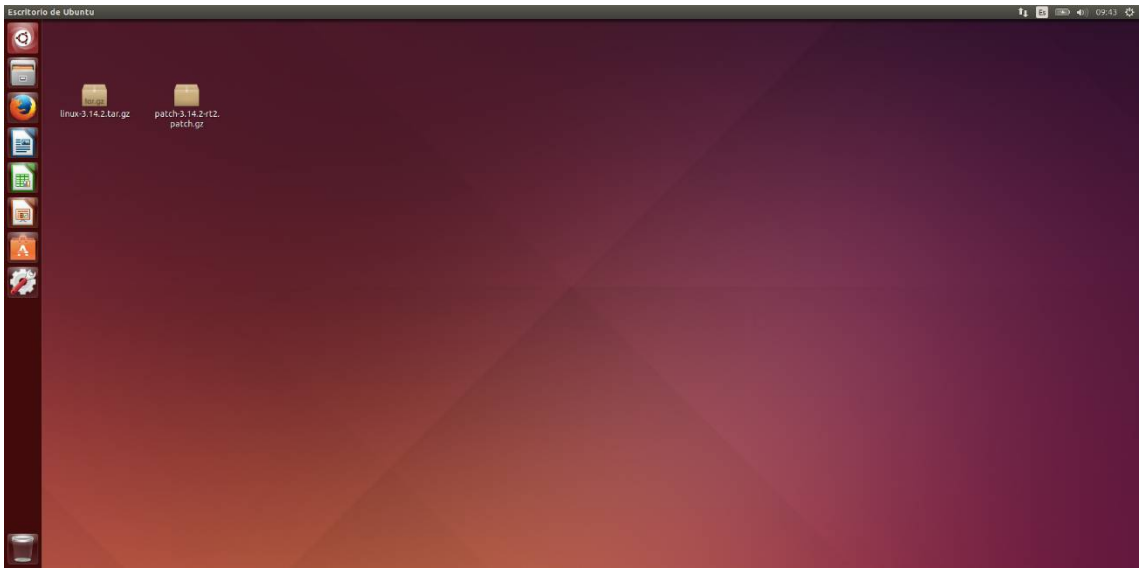


Figura 16. Ficheros descargados en el escritorio: linux-3.14.2.tar.gz y patch-3.14.2-rt2.patch.gz

Los hemos descargado desde estas URL:

https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.14.2.tar.gz
https://www.kernel.org/pub/linux/kernel/projects/rt/3.14/older/patch-3.14.2-rt2.patch.gz

Copiamos las fuentes del nuevo kernel que está en un comprimido en el escritorio a la carpeta /usr/src. Posteriormente, copiamos el parche a la misma carpeta /usr/src.

Para ello, abrimos un terminal de Linux. Escribimos:

```
sudo -s
cd /home/real-time/Escritorio
```

`real-time` es el nombre de usuario de la máquina.

```
cp linux-3.14.2.tar.gz /usr/src/
cp patch-3.14.2-rt2.patch.gz /usr/src/
```

Vamos a la carpeta `/usr/src`

```
cd /usr/src
```

Descomprimimos las fuentes:

```
tar -xzvf linux-3.14.2.tar.gz
```

Renombramos la carpeta:

```
mv linux-3.14.2 linux-3.14.2-rt2
```

Descomprimimos el parche:

```
gzip -d patch-3.14.2-rt2.patch.gz
```

Vamos a la carpeta de fuentes del kernel.

```
cd linux-3.14.2-rt2
```

A continuación, se aplica el parche PREEMPT_RT:

```
patch –p1 < ../patch-3.14.2-rt2.patch
```

En este último paso es posible que, en función de la distribución de trabajo usada, no se tenga la utilidad `patch` instalada de antemano. En este tipo de situaciones, habría que descargar e instalar la utilidad, o bien manualmente, o bien usando algún gestor de paquetes que el sistema pueda albergar. Por ejemplo, ejecutando:

```
apt-get install patch
```

Antes de compilar el nuevo kernel se crea el enlace `/usr/src/linux` apuntando al directorio de las fuentes, ya que durante la compilación algunos programas buscan este directorio:

```
ln -s  /usr/src/linux-3.14.2-rt2 /usr/src/linux
```

Para preservar estas opciones y añadir las de tiempo-real, u otras, se debe copiar el archivo de configuración al directorio del kernel a compilar.

```
cp /boot/config-`uname -r` .config
```

```
apt-get install libgtk-3-dev libglib2.0-dev libglade2-dev
```

```
make oldconfig
```

A continuación, debemos responder a algunas preguntas con Y (*Yes*) y las demás con Intro. Las que tenemos que responder con Y son las siguientes:

- *Debug preemptible kernel (DEBUG_PREEMPT) [Y/n/?] (NEW)*

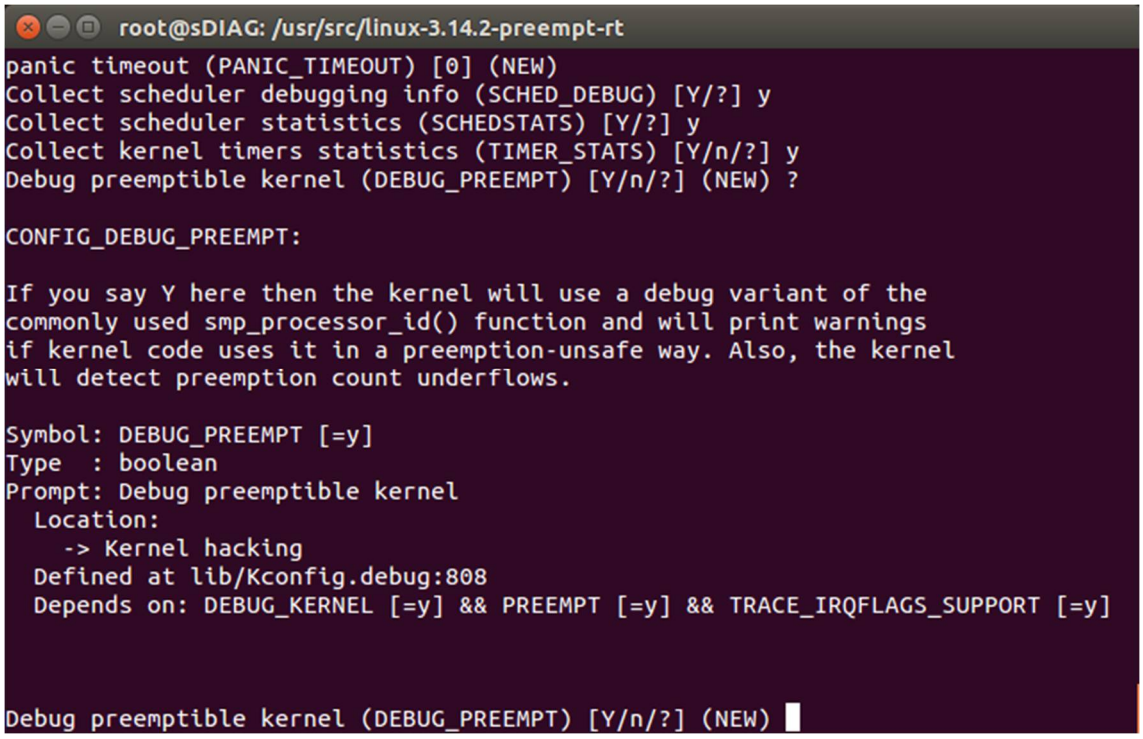


Figura 17. Configuración del kernel: DEBUG_PREEMPT

- *Preemption-off Latency Tracer (PREEMPT_TRACER) [N/y/?] (NEW) ?*

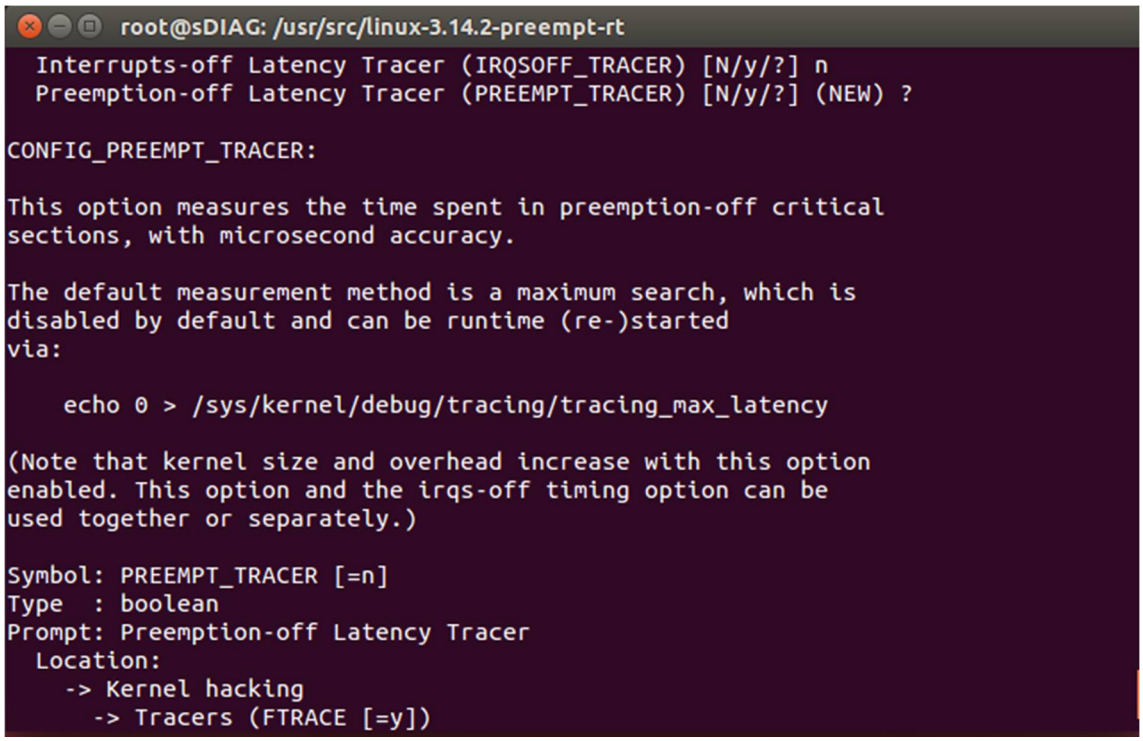


Figura 18. Configuración del kernel: PREEMPT_TRACER

- *Preemption-off Latency Histogram (PREEMPT_OFF_HIST) [N/y/?] (NEW) ?*

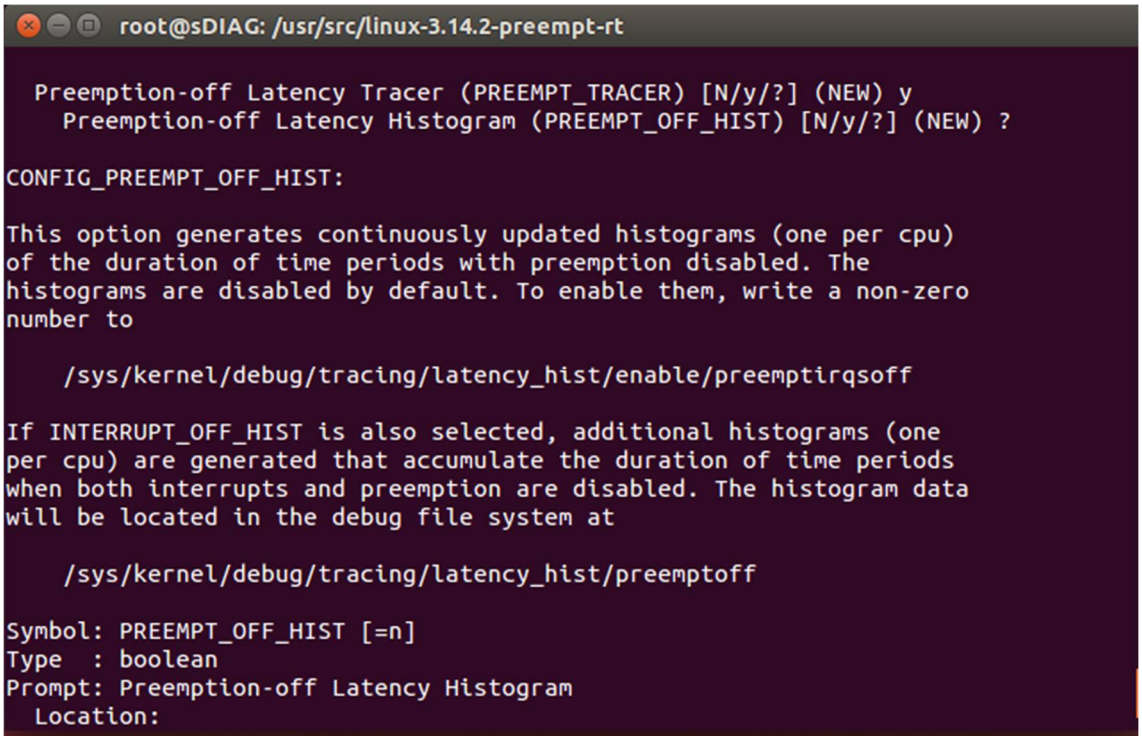


Figura 19. Configuración del kernel: PREEMPT_OFF_HIST

- *Scheduling Latency Histogram (WAKEUP_LATENCY_HIST) [N/y/?] (NEW)*

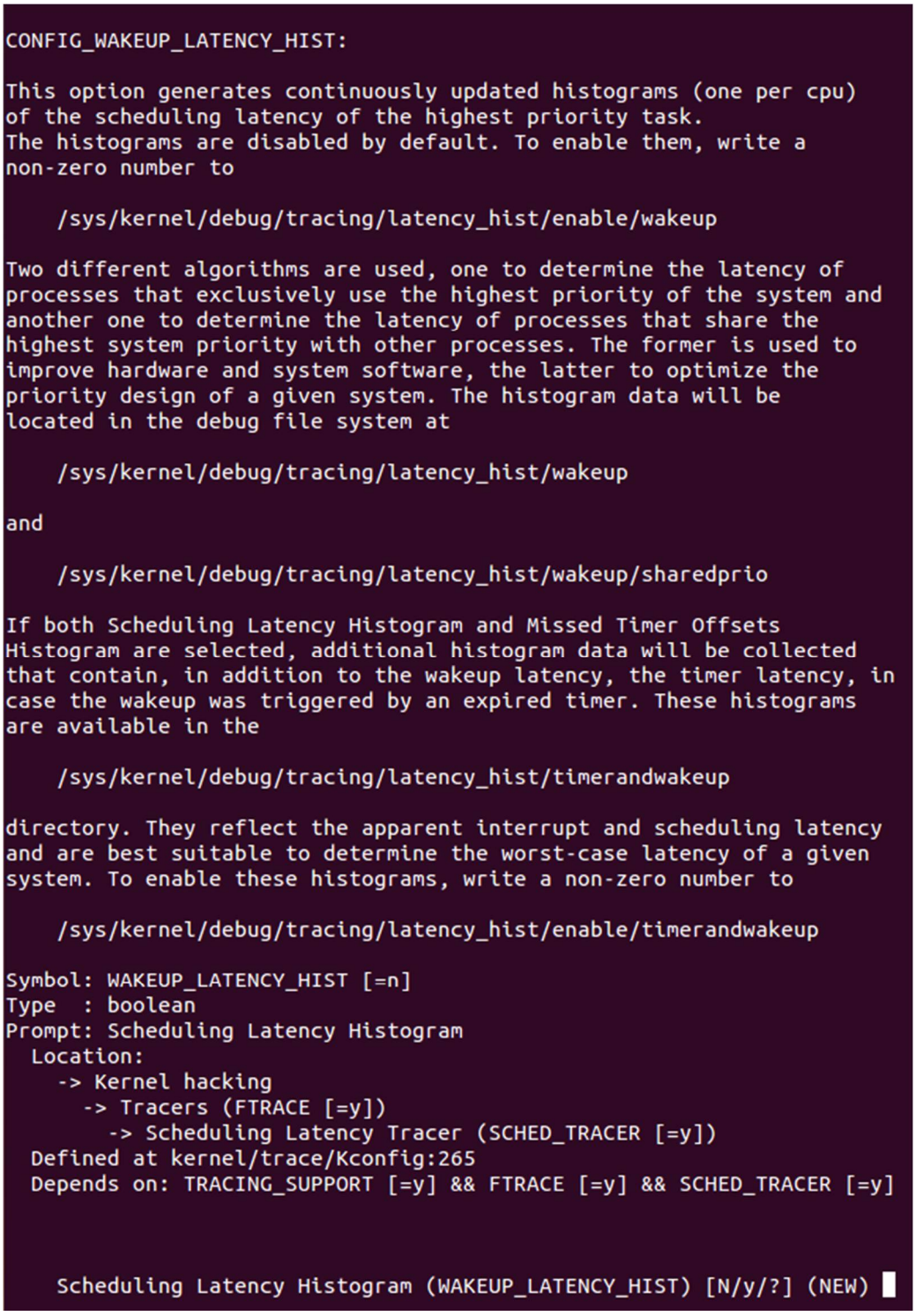


Figura 20. Configuración del kernel: WAKEUP_LATENCY_HIST

- *Missed Timer Offsets Histogram (MISSED_TIMER_OFFSETS_HIST) [N/y/?] (NEW)*

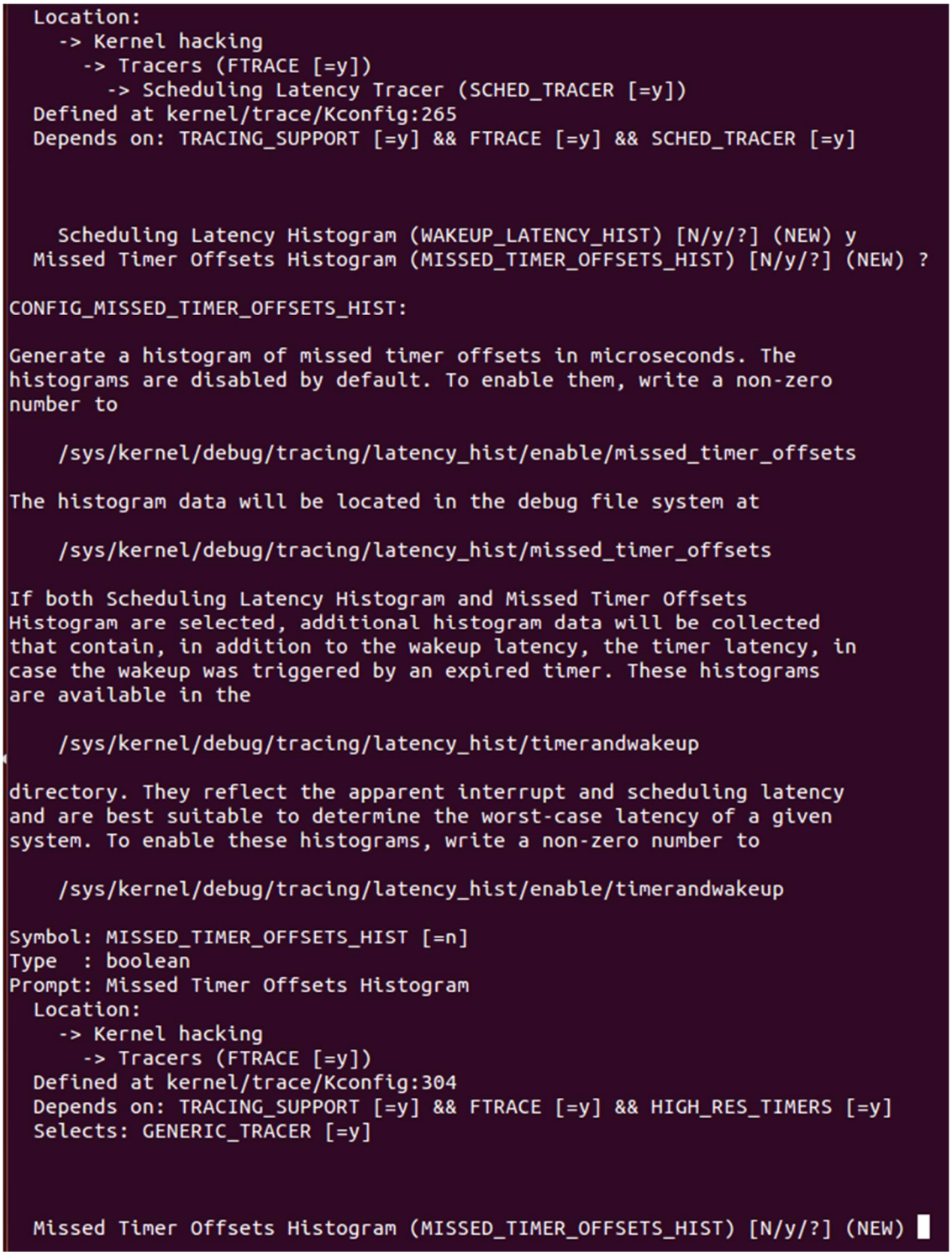


Figura 21. Configuración del kernel: MISSED_TIMER_OFFSETS_HIST

Después de realizar esto, cuando se vuelve a la línea de comandos, hay que teclear:

```
make gconfig
```

Cabe destacar que pueden aparecer ciertos contratiempos por no disponer del software necesario. Por ejemplo, el comando `make` no siempre está preinstalado en el sistema, entonces, se deberá hacer uso, nuevamente, de algún gestor de paquetes que el sistema tenga, o bien descargar e instalar manualmente el software necesario.

Con este último comando (`make gconfig`), habilitamos las siguientes opciones:

Processor type and features > Preemption Model > Fully Preemptible Kernel (RT):

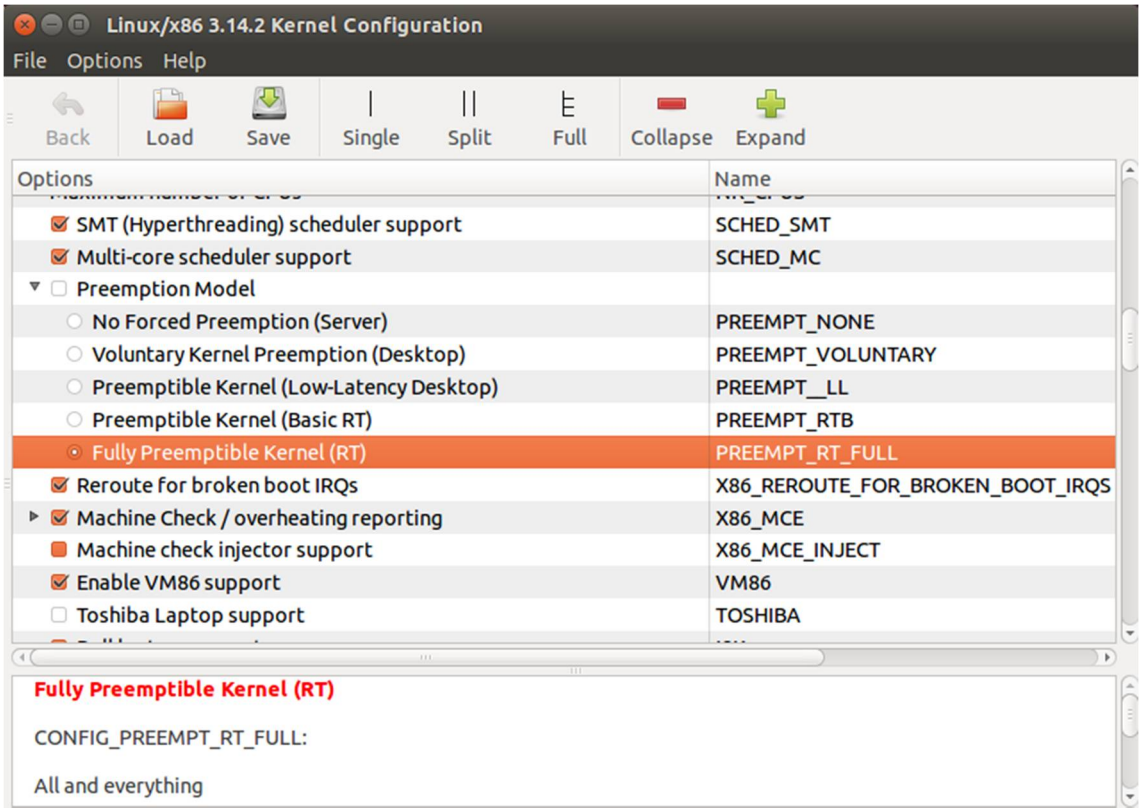


Figura 22. Configuración del kernel: CONFIG_PREEMPT_RT_FULL

- *No Forced Preemption*. Sin desalojo obligado. En general, la latencia es buena en la media, pero algunos retrasos ocasionales pueden ocurrir. La más adecuada para aplicaciones en las que el rendimiento total es el criterio de diseño clave.
- *Voluntary Kernel Preemption*. El primer paso en la reducción de la latencia. Puntos de desalojo explícitos adicionales se colocan en zonas estratégicas del kernel para reducir la latencia. A cambio, hay algo de pérdida de rendimiento total.
- *Preemptible Kernel*. Este modo habilita desalojo en cualquier lugar en el kernel excepto cuando se procesa dentro de secciones críticas. Este modo es útil para aplicaciones de tiempo-real *soft* tales como audio y multimedia.
- *Fully Preemptible Kernel*. Se añaden características del parche para tiempo-real como sustitución de *spinlocks* (hilo de ejecución que simplemente espera en un bucle (*spins*)). Esto habilita desalojo involuntario en cualquier lugar dentro del kernel excepto para áreas protegidas por `preempt_disable()`. Este modo significativamente reduce la variación en latencia (*jitter*) y permite una latencia baja y predictible para aplicaciones de tiempo-real de tiempo crítico.

Nosotros elegiremos esta última opción.

Power management and ACPI options > ACPI (Advanced Configuration and Power Interface) Support> Button, Fan y Processor (deshabilitados)

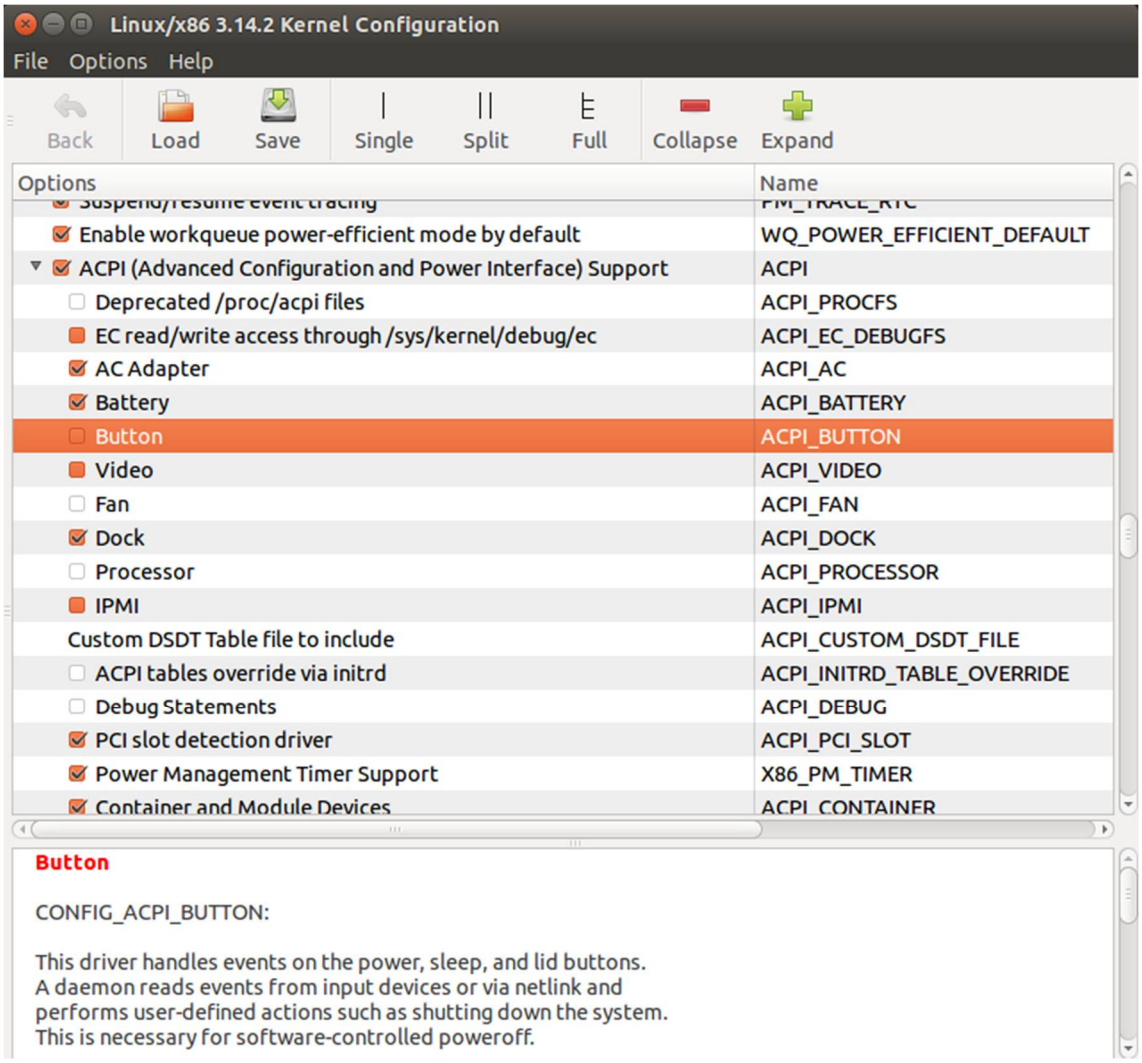


Figura 23. Configuración del kernel: CONFIG_ACPI_BUTTON

General setup > *Optimize for size (activado)* [Fayyad-Kazan et al. 14]

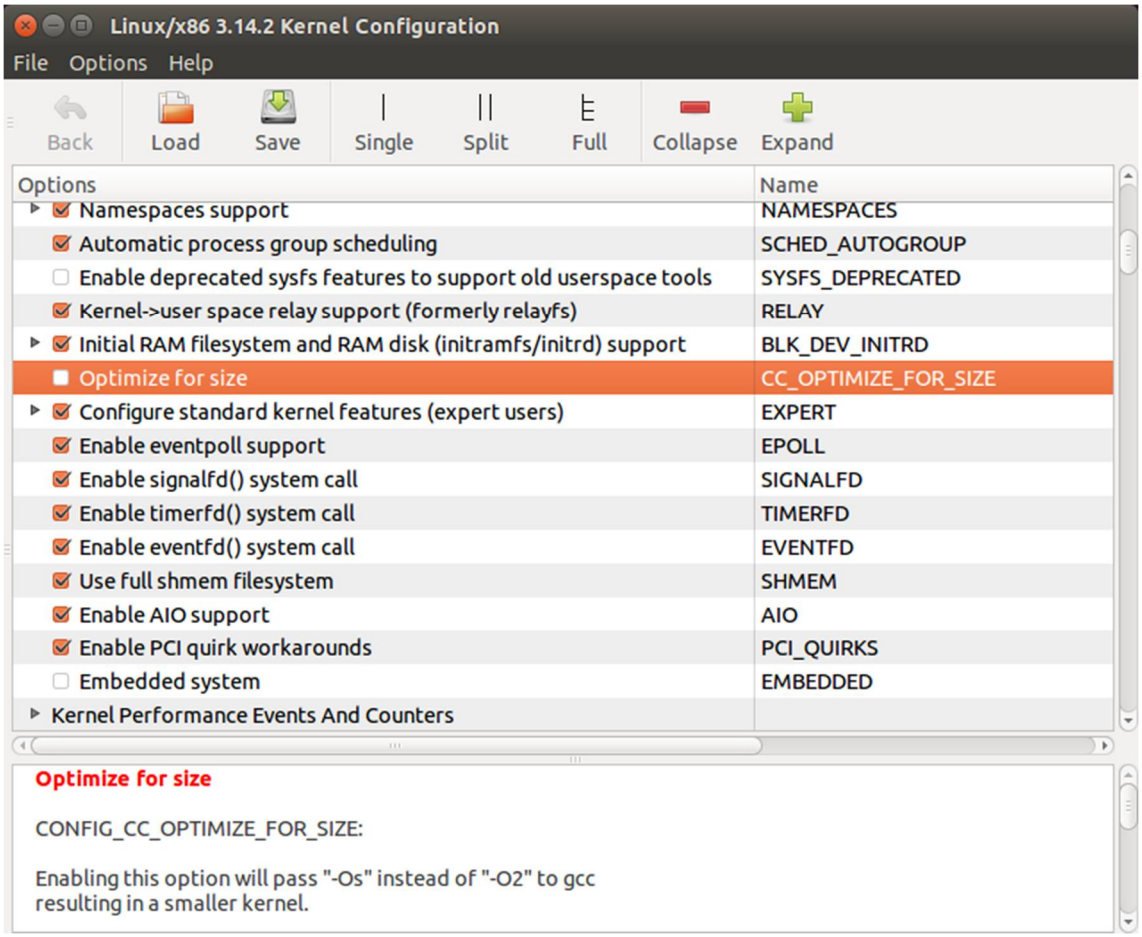


Figura 24. Configuración del kernel: CONFIG_CC_OPTIMIZE_FOR_SIZE

Del mismo modo, en *Kernel hacking* > *Tracers* dejamos habilitado. Si fuese necesario desplegamos la opción y habilitamos:

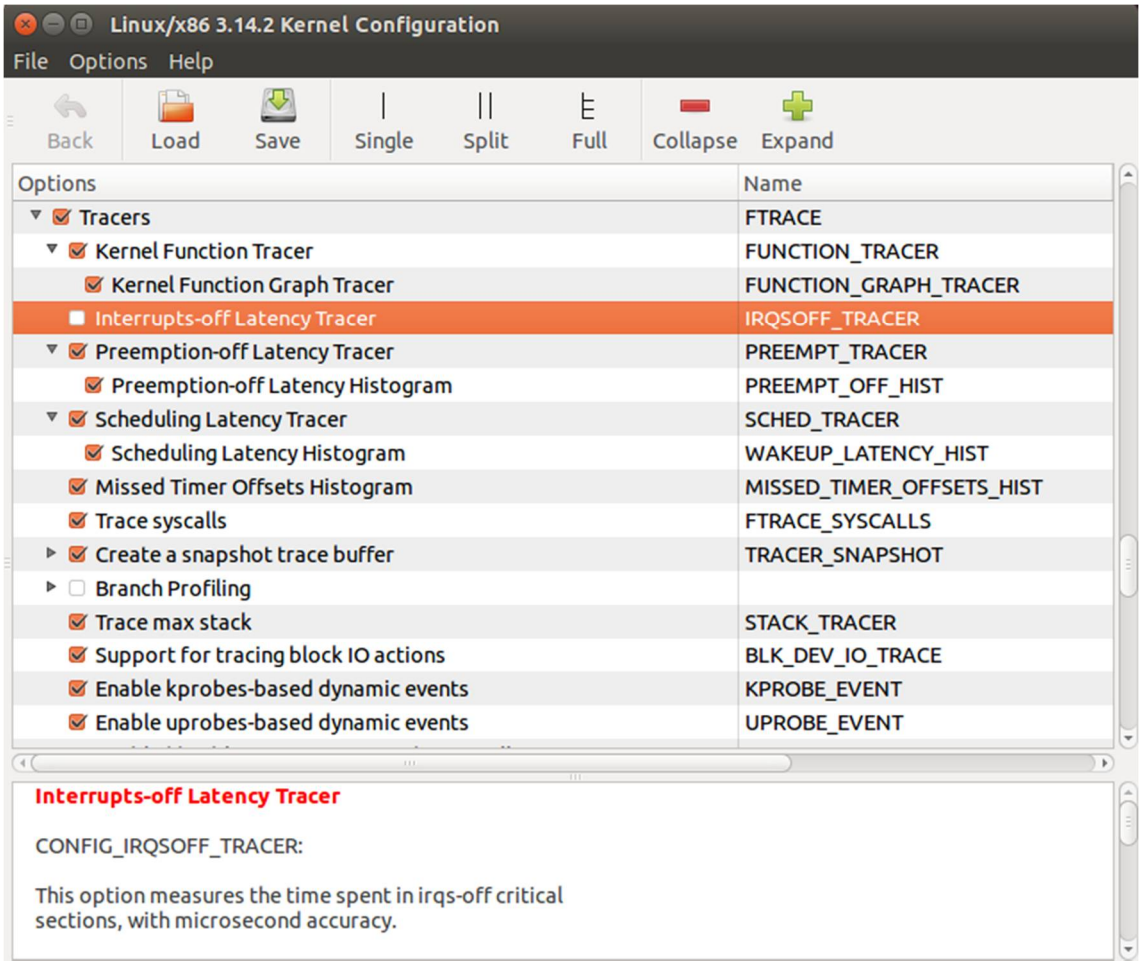


Figura 25. Configuración del kernel: CONFIG_IRQSOFF_TRACER

Esto es, activamos una serie de opciones para llevar a cabo actividades de depuración del kernel, así como, realizar trazas de ciertas partes del código relativas a la capacidad de tiempo-real. También existen herramientas de diagnóstico que se pueden activar.

A continuación, ya se está en disposición de compilar el kernel:

```
apt-get install kernel-package

make-kpkg clean

time CONCURRENCY_LEVEL=? fakeroot make-kpkg --initrd --append-to-version=-preempt kernel_image kernel_headers
```

El nivel de concurrencia del comando anterior lo podemos calcular en función del número de núcleos (*cores*) que tiene la máquina. En GNU/Linux esto se puede averiguar escribiendo:

```
cat /proc/cpuinfo | grep “model name”
cat /proc/cpuinfo | grep “cpu cores”
```

para generar los dos .deb (*image* y *headers*), en la carpeta inmediatamente superior (/usr/src). Este proceso puede tardar un tiempo considerable, en función de la máquina usada.

```
cd ..
dpkg -i <nombre_de_paquete_image_generado.deb>
dpkg –i <nombre_de_paquete_headers_generado.deb>
```

Tras esto:

```
nano /etc/default/grub
```

ponemos como comentario (#) las entradas con `GRUB_HIDDEN`

Y actualizamos:

```
update-grub
```

Con esta serie de modificaciones, el sistema operativo GNU/Linux tiene todo lo necesario para arrancar con el nuevo kernel con capacidad de tiempo-real.

Reiniciamos. Y comprobamos si la versión del kernel ha cambiado.

```
reboot
uname -a
```

donde en la salida devuelta se observa claramente la característica **PREEMPT_RT**. También se puede ver el contenido del fichero *version* que se encuentra en el pseudo sistema de ficheros `/proc`:

En lugar de lanzar el comando y los siguientes comandos, otra opción es:

```
cat /proc/version
```

En ambos casos se puede observar que la opción de expropiación (**PREEMPT**) del kernel y la opción de tiempo-real (**RT**) están activadas. Por último, podemos ejecutar el siguiente código para comprobar que estamos con el kernel modificado:

```
#include <string.h>
#include <stdio.h>
#include <sys/utsname.h>

int main(int argc, char **argv)
{
    struct utsname u;
    int crit1, crit2 = 0;
    FILE *fd;

    uname(&u);
    crit1 = strcasestr (u.version, "PREEMPT RT");

    if ((fd = fopen("/sys/kernel/realtime","r")) != NULL) {
        int flag;
        crit2 = ((fscanf(fd, "%d", &flag) == 1) && (flag == 1));
        fclose(fd);
    }
    fprintf(stderr, "this is a %s kernel\n",
            (crit1 && crit2)  ? "PREEMPT RT" : "vanilla");
}
```

Figura 26. Programa en C para comprobar que se ha modificado el kernel a PREEMPT_RT

Para ello:

```
gcc -o test_rt test_rt.c –lrt
./test_rt
```

Si todo está bien, veremos por la consola el siguiente mensaje:

```
this is a PREEMPT RT kernel
```

Otra forma de verificar que todo ha ido bien es usando las herramientas que el propio kernel ofrece. Para acceder a las herramientas de diagnóstico, se debe montar el sistema de ficheros de debugeo. Esto se puede hacer añadiendo las siguientes líneas al fichero `/etc/fstab` para que se monte automáticamente cada vez que se arranque.

```
nodev /sys sysfs defaults 0 0
nodev /sys/kernel/debug debugfs defaults 0 0
```

Por último, se pueden hacer histogramas de latencias, también instalados tras la activación de las opciones de *kernel hacking* durante la configuración del kernel. Para ello, hay que habilitar el *wakeup latency histograms* escribiendo por línea de comandos:

```
echo 1 > /sys/kernel/debug/tracing/latency_hist/enable/wakeup
```

Para resetear los contadores del histograma, se puede usar el siguiente *script* [Emde 09]:

```
#!/bin/sh
TRACINGDIR=/sys/kernel/debug/tracing
HISTDIR=$TRACINGDIR/latency_hist
if test -d $HISTDIR
then
      cd $HISTDIR
      for i in `find . | grep /reset$`
      do
            echo 1 >$i
      done
fi
```

Figura 27. Programa *script* [Emde 09] para resetear los contadores del histograma

Estos histogramas se encuentran en */sys/kernel/debug/tracing/latency_hist/wakeup/CPU**0***, en el caso del ordenador usado, ya que es monoprocesador.

Ejecutando en la *shell* el siguiente comando, se obtiene una lista de la cantidad de tareas ejecutadas en el sistema y de sus tiempos de latencia:

```
grep -v " 0$" /sys/kernel/debug/tracing/latency_hist/wakeup/CPU0
```

Como hemos visto es tedioso introducir cada vez toda la ruta /sys/kernel/debug/tracing/. Para ello, creamos el siguiente enlace:

```
ln -s /sys/kernel/debug/tracing /tracing
```

Dentro de esta área de debugeo, Ftrace ha sustituido los mecanismos viejos de rastreo del kernel. Nosotros ya lo habíamos habilitado en la configuración del kernel. En [Hallinan 11] obtienen más estadísticas de latencias basándose en Ftrace, que pasamos a comentar a continuación:

- *Preemption Off Latency Meaurement*.

El kernel usa llamadas para deshabilitar el desalojo durante el procesado en estructuras críticas de datos compartidos. Cuando el desalojo está deshabilitado, las interrupciones pueden todavía ocurrir, pero un proceso de prioridad más alta no puede ejecutarse. Uno puede configurar los tiempos *preempt off* usando la funcionalidad *preemptoff* de Ftrace.

Para habilitar la medición de latencia *preempt off* ya habilitamos CONFIG_PREEMPT_TRACER y CONFIG_PREEMPT_OFF_HIST en el submenú de *Kernel hacking* de la configuración del kernel y que son necesarios. El método general para lanzar Ftrace para una medición *preemptoff* es el siguiente:

```
echo preemptoff > /tracing/current_tracer
echo latency-format > /tracing/trace_options
echo 1 > /tracing/tracing_on
echo 0 > /tracing/tracing_max_latency
ls –ltr
echo 0 > /tracing/tracing_on
cat /tracing/trace | head -20
cat /tracing/tracing_max_latency
```

- *Wakeup Latency Measurement*

Cuando un proceso de tiempo-real (uno con el atributo de planificación `SCHED_FIFO` o `SCHED_RR`) se está ejecutando en el sistema, está por definición compartiendo el procesador con otras tareas. Cuando un evento necesita atención, la tarea de tiempo-real se despierta. En otras palabras, se informa al planificador que necesita ejecutarse. El tiempo de despierte (*wakeup timing*) es el tiempo que va desde que el evento se despierta hasta que la tarea consigue la CPU y empieza a ejecutarse.

Ftrace tiene una traza de despierte para tiempo-real (*wakeup_rt*). Esto registra y traza la latencia más larga desde que se despierta hasta que se ejecuta. Esto se puede ver utililizando los siguientes comandos:

```
echo 0 > /tracing/tracing_on
echo 0 > /tracing/tracing_max_latency
echo wakeup > /tracing/current_tracer
echo 1 > /tracing/tracing_on
cat /tracing/trace | head -20
cat /tracing/tracing_max_latency
```

Y se notifica la latencia de despierte (*wakeup latency*) máxima.

- *Interrupt Off Timing*

Para habilitar la medición del máximo tiempo de *interrupt off*, hay que asegurarse que el kernel tiene configurado CONFIG_IRQSOFF_TRACER en la configuración del kernel. Para ello, escribimos en la *shell* de Linux:

```
cd /usr/src/linux
grep CONFIG_IRQSOFF_TRACER .config
```

Esta opción mide el tiempo que permanece en secciones críticas con IRQs (peticiones de interrupción) deshabilitadas. Esta funcionalidad trabaja de la misma manera que *wakeup latency timing*. Para habilitar esta medición, hay que escribir lo siguiente como *root*:

```
echo irqsoff > /tracing/current_tracer
echo latency-format > /tracing/trace_options
echo 1 > /tracing/tracing_on
echo 0 > /tracing/tracing_max_latency
ls -ltr
echo 0 > /tracing/tracing_on
```

Para leer el máximo actual, simplemente visualizamos el contenido de /tracing/tracing_max_latency:

```
cat /tracing/tracing_max_latency
```

Como vemos, las mediciones de latencia para tanto *wakeup latency* como para *interrupt off* se visualizan usando el mismo fichero. Esto significa que sólo una medición se puede configurar cada momento, o los resultados no son válidos. Como las mediciones añaden significativa sobrecarga de ejecución sería imprudente habilitarlas todas a la vez.

Con todo lo expuesto, se puede estar en disposición de afirmar que el sistema operativo de propósito general GNU/Linux se comporta ahora como un sistema operativo de tiempo-real.

A.2. Solución propuesta 3: Xenomai

Para asegurar que se aplica bien esta otra solución, tenemos que usar la misma versión de parche asociada al kernel. En nuestro caso, en el que se va a trabajar en el *target* con el kernel 3.14.17, usamos el fichero xenomai-2.6.4. Éste incluye el parche `ipipe-core-3.14.17-x86-4.patch`.

Partimos de este estado en el que ambos ficheros están descargados en el escritorio:

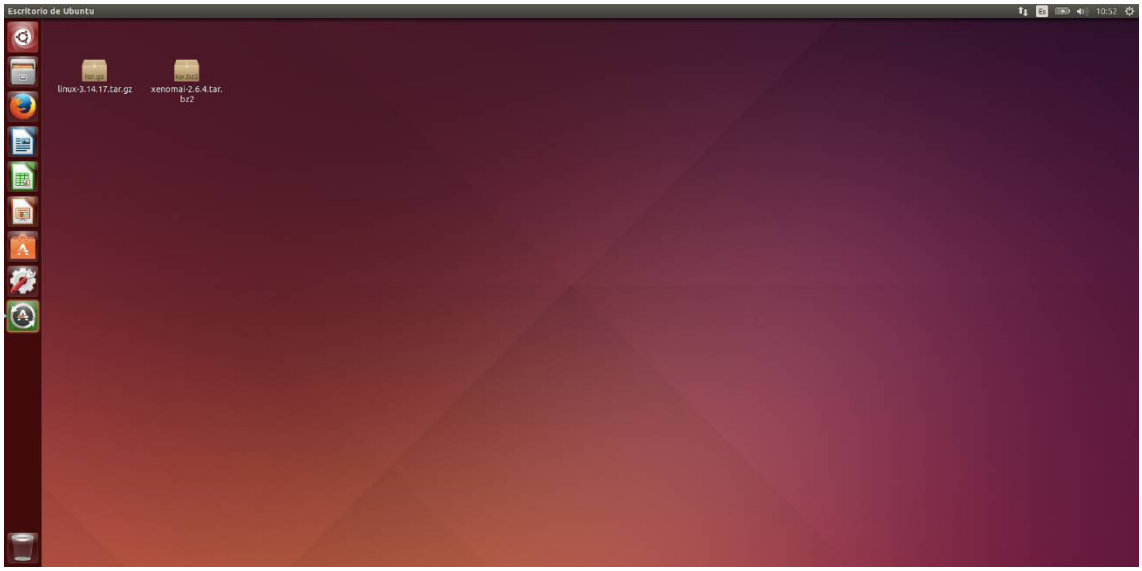


Figura 28. Ficheros descargados en el escritorio: linux-3.14.17.tar.gz y xenomai-2.6.4.tar.bz2

Los hemos descargado desde estas URL:

https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.14.17.tar.gz
https://xenomai.org/downloads/xenomai/stable/xenomai-2.6.4.tar.bz2

Paso 1: Instalar el kernel 3.14.17

```
sudo -s
cp /home/real-time/Escritorio/linux-3.14.17.tar.gz /usr/src
```

`real-time` es el nombre de usuario de la máquina.

```
cd /usr/src
tar –xzvf linux-3.14.17.tar.gz
ln –s linux-3.14.17 linux
cd linux
make clean
cp /boot/config-`uname –r` .config
make oldconfig
```

Pulsamos todo intro, a todas las preguntas que nos hacen.

```
make
make modules_install
make install
cd /boot
mkinitramfs –o initrd.img-3.14.17 3.14.17
nano /etc/default/grub
```

Ponemos como comentario (#) las entradas GRUB_HIDDEN

```
update-grub
```

Paso 2: Reiniciar con el nuevo kernel 3.14.17 e instalar Xenomai

```
reboot
sudo –s
uname –a
pwd (/home/real-time)
```

`real-time` es el nombre de usuario de la máquina.

```
cd Escritorio
tar –xzvf linux-3.14.17.tar.gz
mv linux-3.14.17 /usr/src/linux-3.14.17-xenomai
cp xenomai-2.6.4.tar.bz2 /usr/src
cd /usr/src
rm linux
ln –s linux-3.14.17-xenomai linux
tar –xvjf xenomai-2.6.4.tar.bz2
ln –s xenomai-2.6.4 xenomai
cp linux-3.14.17/.config linux-3.14.17-xenomai/
cd linux
nano Makefile
```

Ponemos en la entrada `EXTRAVERSION = -xenomai`

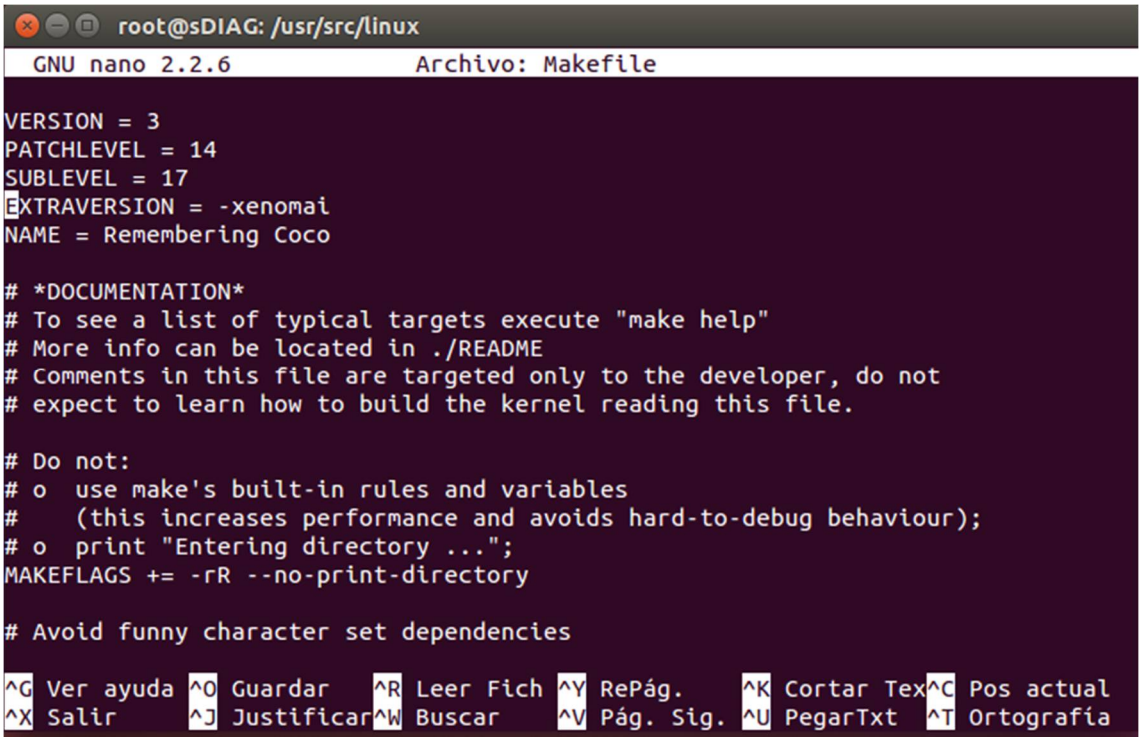


Figura 29. Fichero Makefile

```
cd /usr/src/xenomai
./scripts/prepare-kernel.sh --linux=/usr/src/linux-3.14.17-xenomai
```

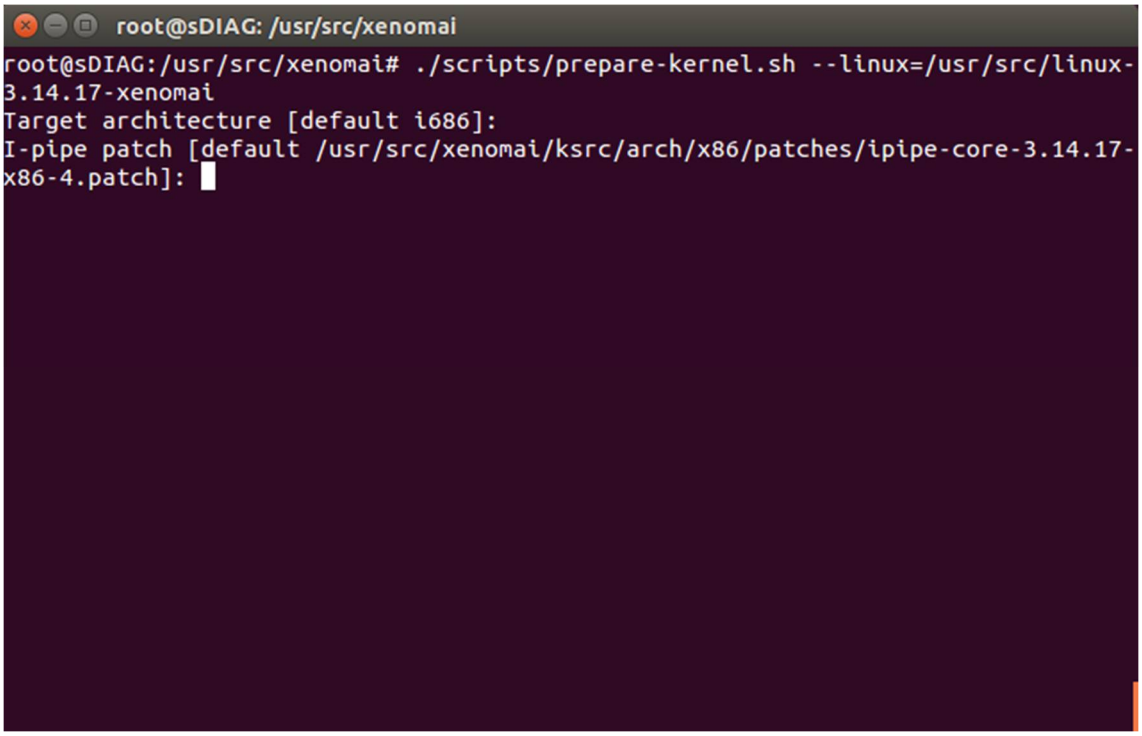


Figura 30. prepare-kernel.sh

Veremos:

Target architecture [default i686]:
Damos Intro
I-pipe patch [default /usr/src/xenomai/ksrc/arch/x86/patches/ipipe-core-3.14.17-x86-4.patch]:
Damos Intro

Una vez hecho esto, deberíamos ver el terminal como la siguiente figura:

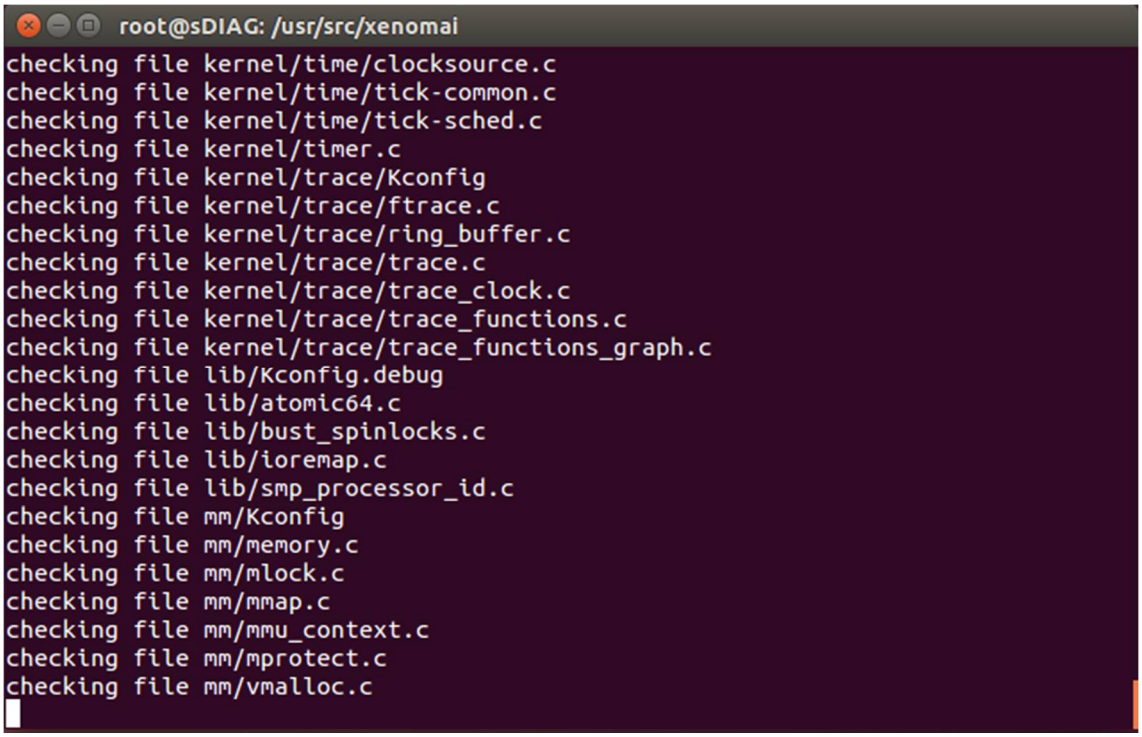


Figura 31. Aplicando el parche Xenomai

Una vez que se vuelve al *prompt* del sistema, escribimos:

```
cd /usr/src/linux-3.14.17-xenomai
apt-get install libgtk-3-dev libglib2.0-dev libglade2-dev
make gconfig
```

Y configuramos el kernel Xenomai siguiendo las instrucciones de la siguiente URL:

Configuring for x86-based dual kernels
https://xenomai.org/2014/06/configuring-for-x86-based-dual-kernels/

Vemos el siguiente mensaje de aviso en la entrada de *Real-time sub-system*. Luego, deshabilitamos:

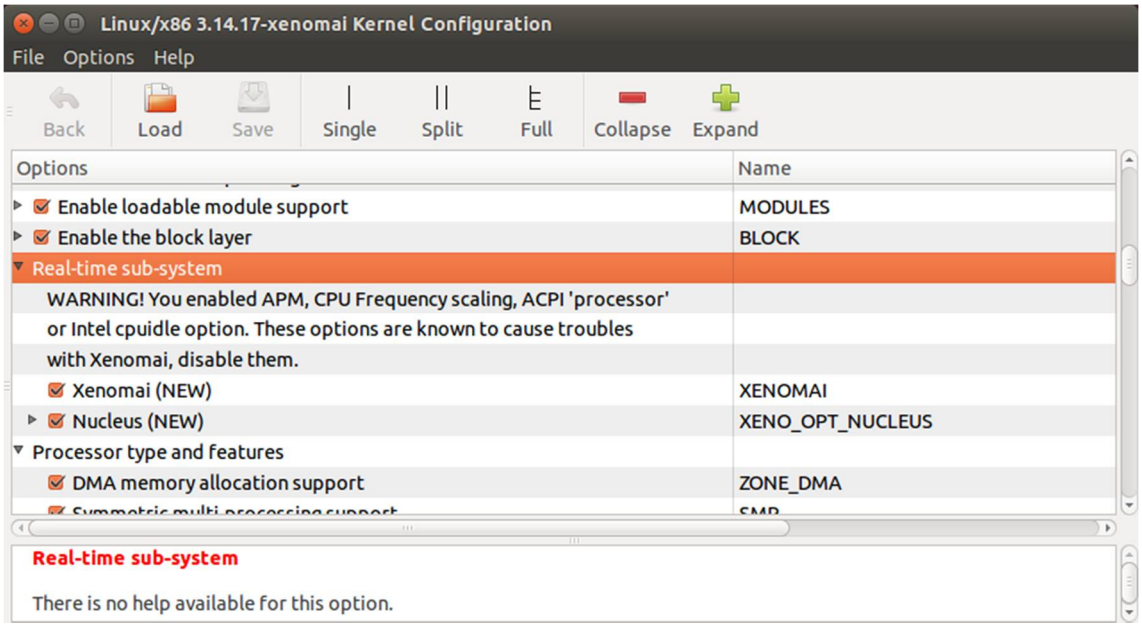


Figura 32. Configuración del kernel: Real-time sub-system

CONFIG_CPU_FREQ (deshabilitado)

Dentro de *Power management and ACPI options* > *CPU Frequency scaling*

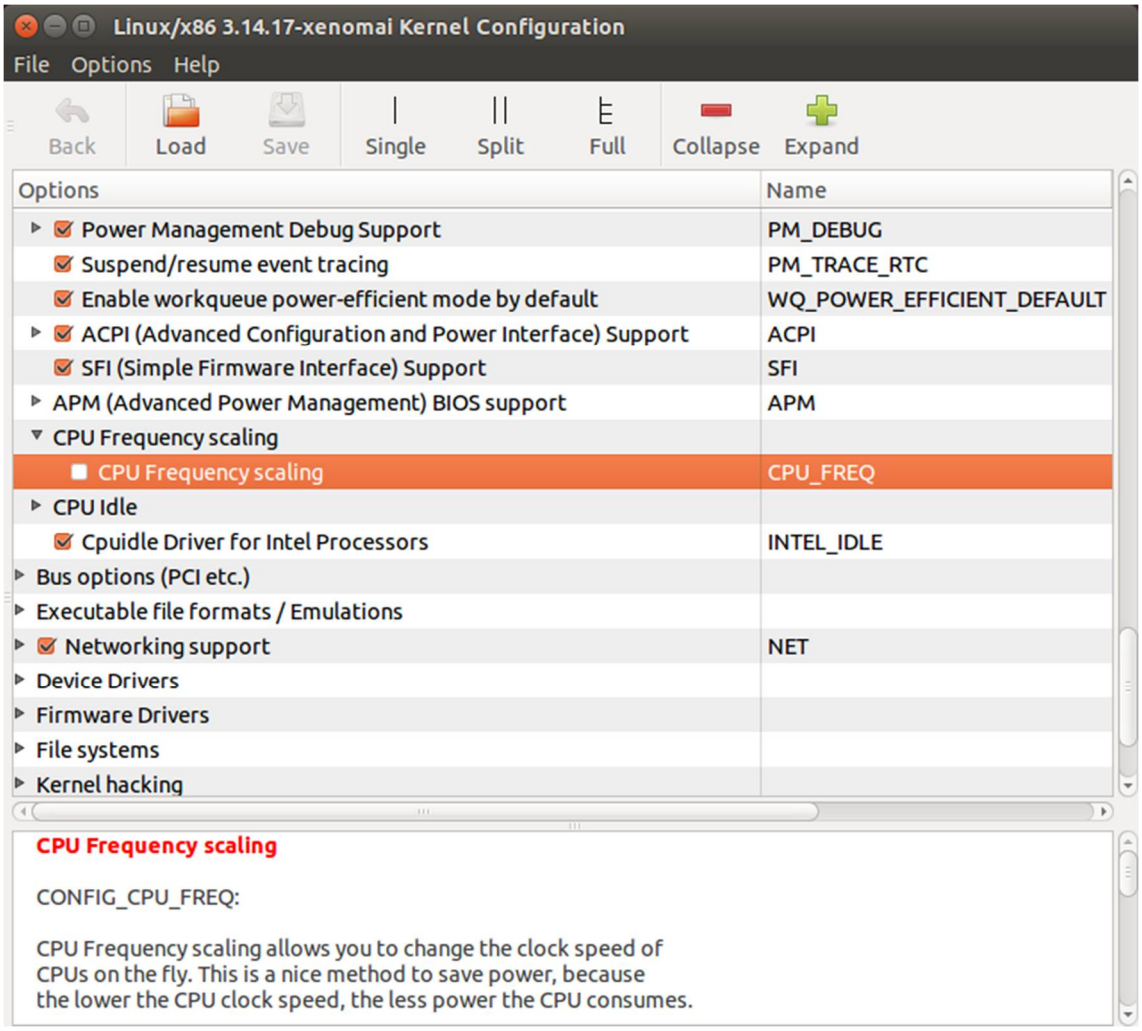


Figura 33. Configuración del kernel: CONFIG_CPU_FREQ

CONFIG_INTEL_IDLE (deshabilitado)

Dentro de *Power management and ACPI options > CPU Idle > Cpuidle Driver for Intel Processors:*

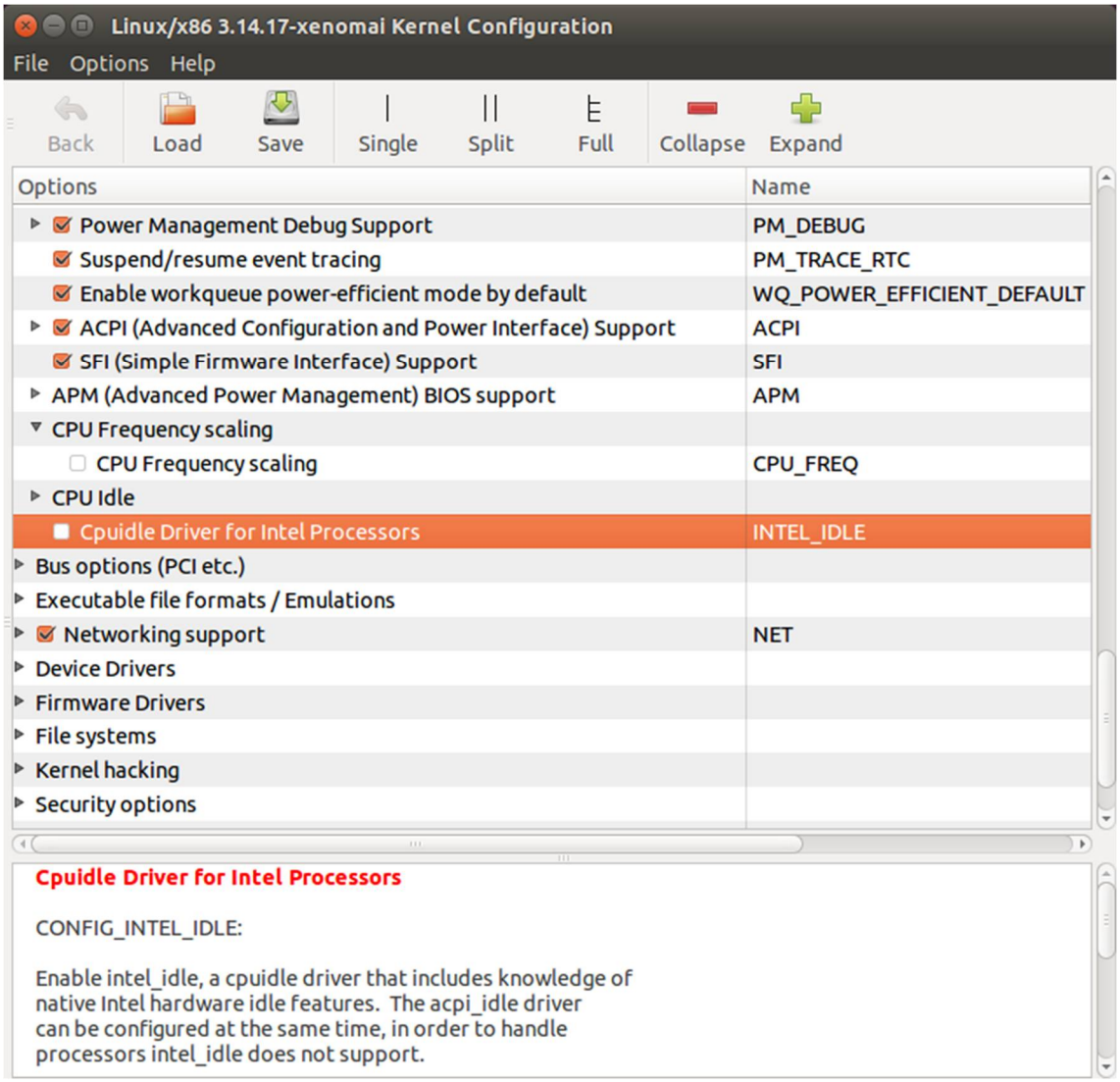


Figura 34. Configuración del kernel: CONFIG_INTEL_IDLE

CONFIG_APM (deshabilitado)

Dentro de *Power management and ACPI options > APM (Advanced Power Management) BIOS support:*

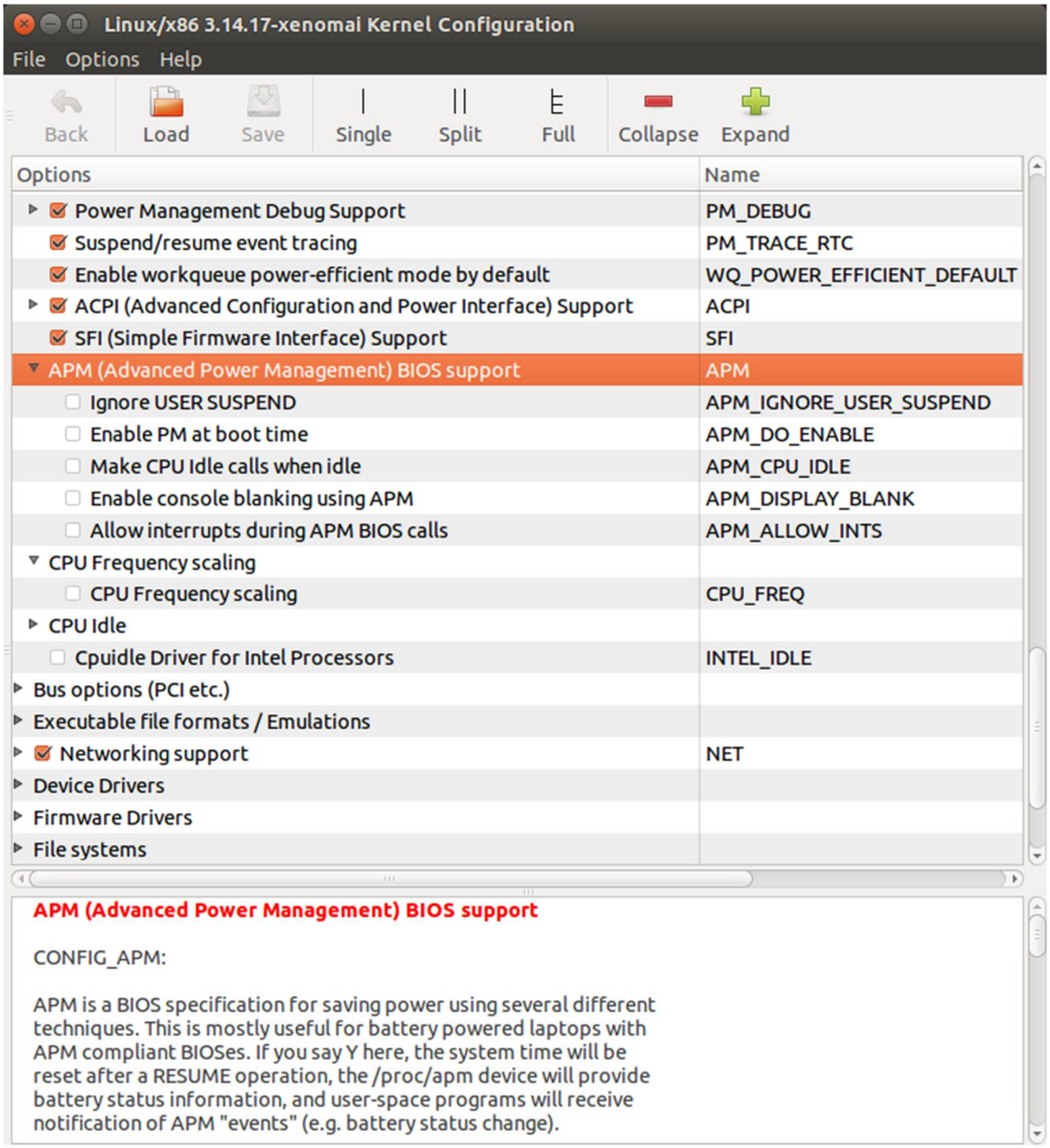


Figura 35. Configuración del kernel: CONFIG_APM

CONFIG_ACPI_PROCESSOR (deshabilitado)

Dentro de *Power management and ACPI options > ACPI (Advanced Configuration and Power Interface) Support > Processor*

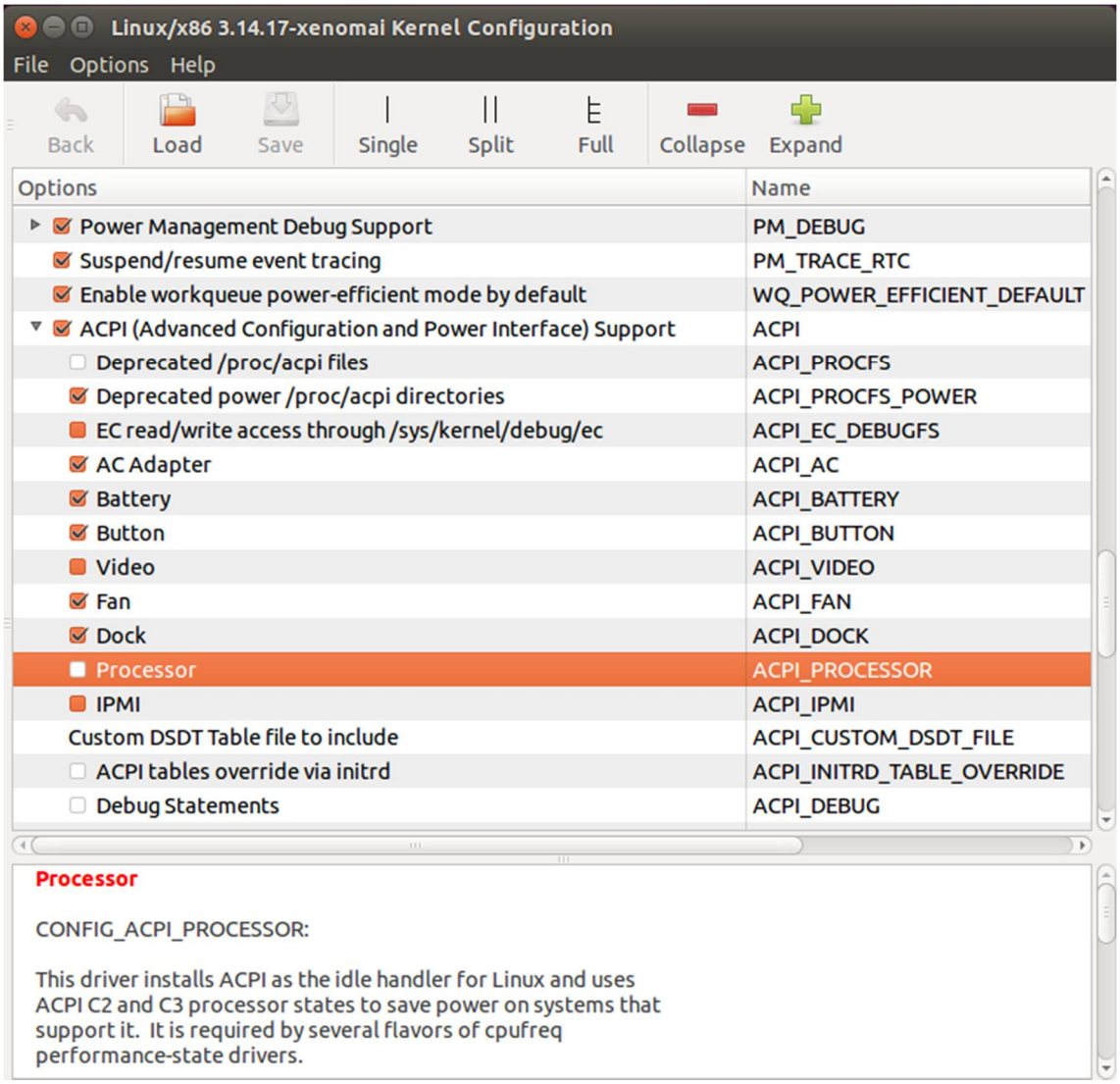


Figura 36. Configuración del kernel: CONFIG_ACPI_PROCESSOR

CONFIG_CPU_IDLE (deshabilitado)

Dentro de *Power management and ACPI options > CPU Idle > CPU idle PM support:*

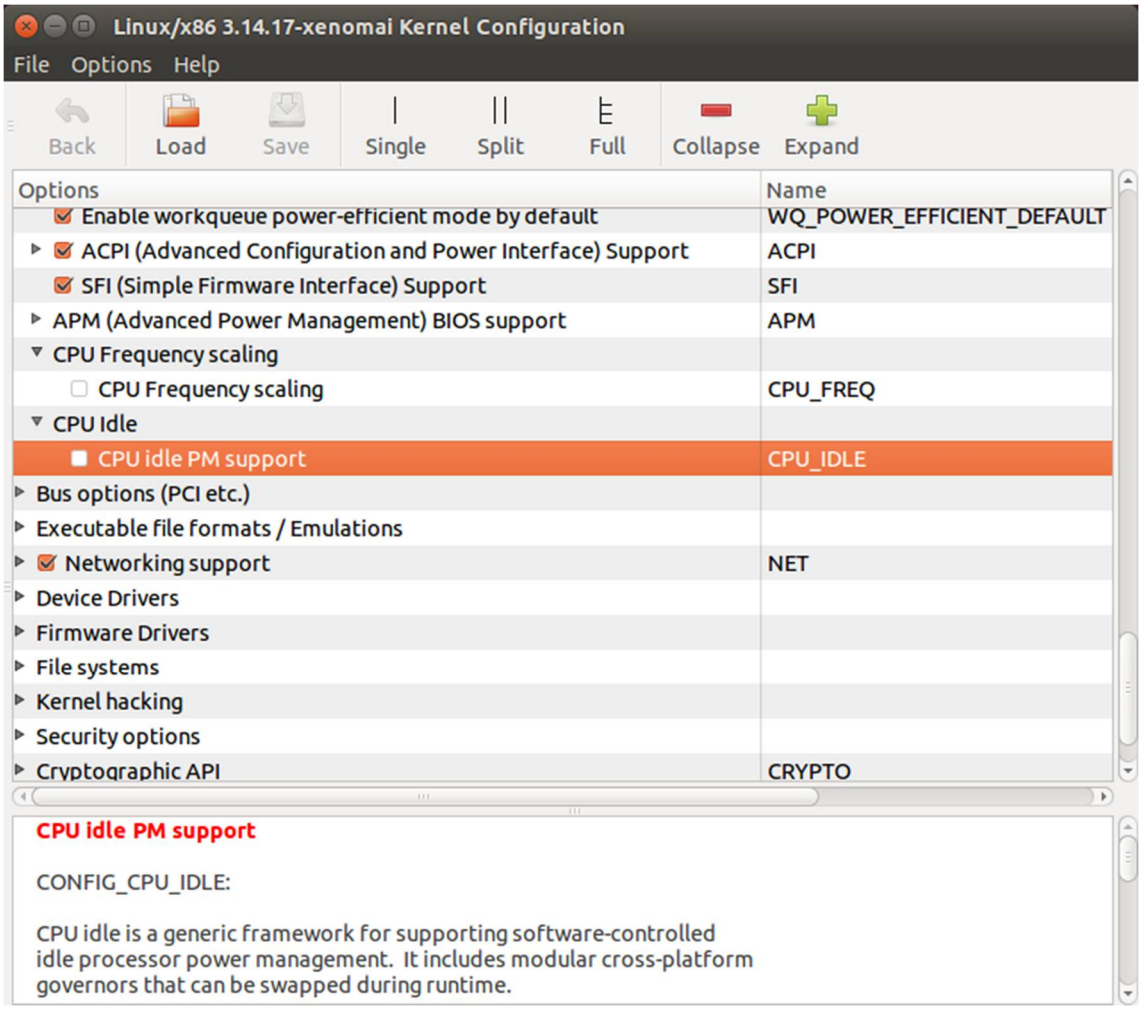


Figura 37. Configuración del kernel: CONFIG_CPU_IDLE

```
apt-get install kernel-package
make-kpkg clean
time CONCURRENCY_LEVEL=? fakeroot make-kpkg --initrd --append-to-
version=-sdiag kernel_image kernel_headers
```

El nivel de concurrencia del comando anterior lo podemos calcular en función del número de núcleos (*cores*) que tiene la máquina. En GNU/Linux esto se puede averiguar escribiendo:

```
cat /proc/cpuinfo | grep “model name”
cat /proc/cpuinfo | grep “cpu cores”
```

para generar los dos .deb (*image* y *headers*), en la carpeta inmediatamente superior (/usr/src). Este proceso puede tardar un tiempo considerable, en función de la máquina usada.

```
cd ..
dpkg -i linux-image-3.14.17-xenomai-sdiag_3.14.17-xenomai-sdiag-
10.00.Custom_i386.deb
dpkg -i linux-headers-3.14.17-xenomai-sdiag_3.14.17-xenomai-sdiag-
10.00.Custom_i386.deb
```

Paso 3: Reiniciar con el nuevo kernel Xenomai y configurar Xenomai

```
reboot
sudo -s
cd /usr/src/xenomai
```

```
./configure --enable-x86-tsc
make install
cd /usr/xenomai/bin
./xeno-test
```

Añadimos:

```
export PATH=$PATH:/usr/xenomai/bin/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/xenomai/lib
```

al final del fichero .bashrc, tanto del usuario *real-time* como del *root*. Están en:

```
/home/real-time/.bashrc
/root/.bashrc
```

B. Anexo: Implementar una aplicación en tiempo-real

En este anexo se estudiará el modo de implementación de aplicaciones en tiempo-real para PREEMPT_RT y Xenomai.

B.1. Solución propuesta 2: PREEMPT_RT

Las aplicaciones de tiempo-real escritas sobre PREEMPT_RT sólo requieren seguir unas buenas prácticas de programación [Scordino 16]. El objetivo es reducir la latencia aleatoria. Esta sección se divide en una serie de apartados (hardware, configuración de kernel y aplicación) que explican cómo podemos reducir latencias (si es posible).

- Hardware

Un buen comportamiento de tiempo-real del sistema depende mucho de un manejador de interrupción de baja latencia. La plataforma x86 no está optimizada para uso de tiempo-real. Diversos mecanismos causan latencias por rutinas de servicio de interrupción que pueden ejecutarse dentro de decenas o centenas de microsegundos. Conocerlas habilitaría a hacer las mejores opciones de diseño en esta plataforma.

- Interrupción de gestión del sistema (SMI) en chipset Intel x86 ICH

Las interrupciones de gestión del sistema, en adelante SMI, se generan por el hardware de gestión de alimentación que incluye cada plataforma. Éstas suponen un inconveniente si se requiere de tiempo-real. En primer lugar, puede prolongarse durante cientos de microsegundos, que para muchas aplicaciones de tiempo-real causan una variación de latencia inaceptable. Segundo, son las interrupciones de prioridad más altas en el sistema. Tercero, uno no puede interceptar el SMI porque no tiene un vector en la CPU. En vez de esto, cuando la CPU consigue un SMI pasa a un modo especial y salta a una dirección de memoria física predeterminada (que es probablemente en la BIOS ROM).

Consejos para conseguir librarse de las interrupciones SMI en x86.

1. Uso de ratón y teclado PS/2.
2. Deshabilitar el ratón USB y el teclado en BIOS.
3. Compilar un kernel con ACPI habilitado, que es la interfaz que se utiliza para la gestión de la alimentación.
4. Deshabilitar la generación de un temporizador de SMIs.

Nunca debemos deshabilitar las interrupciones SMI globalmente. Deshabilitar SMI puede causar serios daños al ordenador. En sistemas de P4 se puede quemar la CPU hasta dejarla inservible, cuando SMI está deshabilitado. SMIs se usan también para reparar errores de chips, así que ciertos componentes pueden no trabajar como lo esperado cuando SMI está deshabilitado. Así que, hay que asegurarse que se sabe lo que se está haciendo antes de deshabilitar cualquier interrupción SMI.

- DMA bus máster

Los eventos de bus máster pueden causar bloqueos de CPU de larga latencia de muchos microsegundos. Esto se puede generar por cualquier dispositivo que usa DMA, tales como dispositivos SATA/PATA/SCSI y a menudo adaptadores de red. También tarjetas de vídeo, que introducen ciclos de espera en el bus en respuesta a un acceso a CPU, pueden causar este tipo de latencia. En ocasiones, el comportamiento de tales periféricos se puede controlar desde el driver, comprometiendo el rendimiento para ganar en menor latencia. El impacto negativo del bus máster es independiente del sistema operativo elegido, entonces no es un problema único para Linux en tiempo-real. A menudo otros sistemas operativos de tiempo-real experimentan este tipo de latencia.

- Gestión de la alimentación

Muchas BIOS soportan gestión de la alimentación para diferentes tipos de hardware. Obviamente, habilitar gestión de la alimentación ahorra unos pocos wattios pero pierde mucho en latencia. Por consiguiente, se recomienda deshabilitar opciones de gestión de la alimentación o comparar el rendimiento del sistema entero cuidadosamente para cada opción y su impacto en la latencia.

- *Hyper threading*

El *hyper threading* (término usado en Inglés) introduce latencias aleatorias. Como se ha mencionado en el apartado de gestión de la alimentación, se recomienda deshabilitar estas características (si es posible) o evaluar cuidadosamente el rendimiento.

- Configuración de kernel
  - Escalado de CPU en demanda

Esta opción que se puede configurar en el kernel, crea eventos de larga latencia cuando la CPU se pone en un estado de bajo consumo después de un periodo de inactividad. Tales problemas son usualmente bastante fáciles de detectar.

- Aplicación.
  - Consola VGA

Cuando el sistema está cumpliendo sus requerimientos de tiempo-real, la consola de texto VGA no se debería usar. Esta consola de texto VGA causa muy grandes latencias, hasta más de cientos de microsegundos. Es mejor usar una consola serie y no tener opción de *login* en la consola de texto VGA. También se pueden usar conexiones SSH o sesiones Telnet. La opción *quiet* en la línea de comando kernel podría también ser útil para evitar que cualquier *printk* o traza de kernel se escriba en consola. Hay que tener en cuenta que usar una interfaz gráfica de usuario de X no tiene impacto sobre el tiempo-real. La consola de texto VGA es la única que causa latencia.

  - Latencias causadas por fallos de páginas

Los fallos de página son interrupciones que se generan cuando un programa solicita datos que no se encuentran en memoria real. Hay dos tipos de fallos de página, *major* y *minor*. Fallos de página *minor* se manejan sin accesos IO. Fallos de página *major* son fallos de página que se manejan por medio de actividad IO. El mecanismo de intercambio de páginas Linux puede intercambiar páginas de código de una aplicación a disco, y tomará un tiempo largo para intercambiar esas páginas de vuelta a RAM. Si dicha página pertenece al proceso de tiempo-real, las latencias se incrementan enormemente. Los fallos de página son por consiguiente peligrosos para aplicaciones de tiempo-real y se recomienda eviatarlas.

Si no se está utilizando espacio de intercambio y ninguna otra aplicación está estresando la memoria hasta los límites, hay probablemente suficiente RAM libre preparada para que la aplicación de tiempo-real sea usada. En este caso, la aplicación de tiempo-real se ejecutará probablemente en fallos de páginas *minor*, que causa relativamente pequeñas latencias. Se debe tener en cuenta que los fallos de páginas de una aplicación no pueden interferir en el comportamiento de tiempo-real de otra aplicación.

Durante el lanzamiento de una aplicación de tiempo-real siempre experimentará muchos fallos de página. Esto no se puede evitar. De hecho, este periodo de lanzamiento se debe usar para reservar suficiente memoria para el proceso de tiempo-real en RAM. Esto se debe hacer de tal manera que cuando una aplicación necesita exponer sus capacidades de tiempo-real, los fallos de página no ocurran más.

En la literatura hay varios ejemplos que muestran varios aspectos para prevenir fallos de página. Esto depende de los requerimientos que se ajustan mejor para cada propósito.

- Variables globales y *arrays*

Las variables globales y los *arrays* no son parte del binario, pero las aloja el sistema operativo en el proceso de lanzamiento. Las páginas de la memoria virtual asociada a este dato no se mapean inmediatamente a páginas físicas de RAM, por lo que los fallos de página ocurren durante el acceso. Resulta que la llamada `mlockall()` hace que todas las variables y arrays globales pasen a la RAM, asegurando que el acceso a esta memoria no resulta en fallos de página. Como tal, usar variables globales y *arrays* no introduce problemas adicionales para aplicaciones de tiempo-real. Se puede verificar este comportamiento usando el siguiente programa (ejecutar como *root* para permitir la operación `mlockall()`).

```
// This application checks whether mlockall() forces all pages of a
// global array into RAM. Normally, the OS maps a 'copy on write' MMU page
// to such arrays meaning that reading from the array returns only zeros and
// the first write results in a page fault so a physical page of RAM is
// mapped to it (and initialised to zero before the write occurs).

   #include <stdio.h>
   #include <string.h>
   #include <unistd.h>
   #include <sys/mman.h>
   #include <sys/resource.h>

   // Lock the application in memory to avoid page faults
   static void lockApplication(void)
   {
       if (mlockall(MCL_CURRENT | MCL_FUTURE) != 0 )
       {
           perror( "mlockall() failed" );
       }
   }

   // Dump minor and major page faults that occurred since the
   //previous call
   static bool dumpPageFaults(void)
   {
       bool l_PageFaultsDetected = false;
       static bool ls_Init = false;
       static struct rusage ls_RusagePrevious;
       struct rusage l_Rusage;

       getrusage(RUSAGE_SELF, &l_Rusage);
       int a_NewMinorPageFaults =
       l_Rusage.ru_minflt - ls_RusagePrevious.ru_minflt;
       int a_NewMajorPageFaults =
       l_Rusage.ru_majflt - ls_RusagePrevious.ru_majflt;
       ls_RusagePrevious.ru_minflt = l_Rusage.ru_minflt;
       ls_RusagePrevious.ru_majflt = l_Rusage.ru_majflt;

       if (ls_Init)
       {
              if ((a_NewMinorPageFaults > 0) || (a_NewMajorPageFaults > 0))
              {
                     printf ("New minor/major page faults: %d/%d\n",
                            a_NewMinorPageFaults,
                            a_NewMajorPageFaults);
                     l_PageFaultsDetected = true;
              }
       }
       ls_Init = true;
       return l_PageFaultsDetected;
   }

   // Define the global array
   const static int gs_BufferPages = 5000;
   const static int gs_PageSize = 1024*4; // Assume 4kb pages
```

```
   static char gs_Buffer[gs_BufferPages*gs_PageSize];

   int main(int argc, char *argv[])
   {
       bool l_LockMemory = true;
       int i;
       for (i=1; i<argc; i++)
       {
              if (strncmp (argv[i], "-nolockmem", 10) == 0)
              {
                     // Results in many page faults!
                     l_LockMemory = false;
              }
       }

       if (l_LockMemory)
       {
              lockApplication();
              printf ("Current and future memory locked in RAM\n");
       }
       (void)dumpPageFaults(); // Set the baseline

       const int l_PageSize = sysconf(_SC_PAGESIZE);
       printf ("Page size = %d\n", l_PageSize);

       (void)dumpPageFaults();

       // From this point onwards we no longer expect to have any page faults.
       bool l_UnexpectedPageFaultsDetected = false;
       for (i=0; i<gs_BufferPages; i++)
       {
              int l_Value = static_cast<int>(gs_Buffer[i*gs_PageSize]);
              if (l_Value != 0)
              {
       printf ("Reading unexpected value %d on page %d of static
buffer.\n",l_Value,i);
              }
              if (dumpPageFaults())
              {
                     l_UnexpectedPageFaultsDetected = true;
              }
              gs_Buffer[(i*gs_PageSize)+1] = 1;
              if (dumpPageFaults())
              {
       printf ("Writing to page %d of static buffer caused page fault(s)\n",
i);

                     l_UnexpectedPageFaultsDetected = true;
              }
       }

       printf ("Done, result: %s\n",
               l_UnexpectedPageFaultsDetected ? "failed":"success");
       return (l_UnexpectedPageFaultsDetected ? 1:0);
   }
```

Figura 38. Programa en C para evitar fallos de página

B.2. Solución propuesta 3: Xenomai

PREEMPT_RT no proporciona un API de programación a nivel de usuario tal y como lo hace Xenomai.

En Xenomai las librerías de espacio de usuario se compilan usando la tradicional herramienta *autotools* [Petazzoni -].

```
./configure --target=arm-linux && make && make DESTDIR=/your/rootfs/
install
```

El *script* `xeno-config` instalado cuando se instala el soporte para espacio de usuario Xenomai ayuda a compilar los propios programas. Para ello, es interesante ver el directorio de `examples` de Xenomai. Los detalles de la instalación se pueden encontrar en la guía README.INSTALL. En [XenomaiAPIPOSIX 14] hay disponible una introducción a la programación con el API POSIX. Mientras que, en [XenomaiAPINativa 06] hay disponible una introducción a la programación con el API nativa.

Por otro lado, el POSIX skin permite recompilar sin cambios una aplicación POSIX tradicional. Entonces, en lugar de usar servicios de Linux en tiempo-real, usa servicios Xenomai. Xenomai dispone de relojes y temporizadores, variables de condición, colas de mensajes, mutexes, semáforos, memoria compartida, señales y gestión de hilos. Esto es bueno para código existente o programadores familiarizados con el API POSIX. Si la aplicación usa cualquier servicio Linux que no está disponible en Xenomai, cambiará a modo secundario.

Para enlazar una aplicación contra el POSIX skin:

```
DESTDIR=/path/to/xenomai
export DESTDIR
CFL=`$DESTDIR/bin/xeno-config --posix-cflags`
LDF=`$DESTDIR/bin/xeno-config --posix-ldflags`
ARCH-gcc $CFL –o rttest rttest.c $LDF
```

Si una aplicación de tiempo-real Xenomai usando el POSIX skin desea comunicar con una aplicación separada de no tiempo-real, debe usar el mecanismo *rtipc*. En la aplicación Xenomai, crea un socket IPCPROTO_XDDP:

```
socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP);
setsockopt(s, SOL_RTIPC, XDDP_SETLOCALPOOL, &poolsz, sizeof(poolsz));
memset(&saddr, 0, sizeof(saddr));
saddr.sipc_family=AF_RTIPC;
saddr.sipc_port=MYAPPIDENTIFIER;
ret=bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
```

Y el API socket normal `sendto()` / `recvfrom()`.

En la aplicación Linux:

- Abrir `/dev/rtpx`, donde *x* es el puerto XDDP.
- Usar `read()` y `write()`.

B.2.1. API nativa

Existe un API específica de Xenomai para desarrollar tareas de tiempo-real. Se usa tanto en espacio de usuario como en el espacio de kernel. El desarrollo de tareas en el espacio de usuario es el modo preferido. Es un API más flexible y más coherente que el POSIX API. Más fácil de aprender y comprender. Las aplicaciones deberían incluir `<native/service.h>`, donde el servicio puede ser `alarm`, `buffer`, `cond`, `event`, `heap`, `intr`, `misc`, `mutex`, `pipe`, `queue`, `sem`, `task` y `timer`.

Para compilar aplicaciones:

```
DESTDIR=/path/to/xenomai/
export DESTDIR
CFL=`$DESTDIR/bin/xeno-config --xeno-cflags`
LDF=`$DESTDIR/bin/xeno-config --xeno-ldflags`
ARCH-gcc $CFL -o rttest rttest.c $LDF –lnative
```

Las funciones de servicios de gestión de tareas son:

`rt_task_create()`, `rt_task_start()`, `rt_task_suspend()`, `rt_task_resume()`, `rt_task_delete()`, `rt_task_join()`, etc.

Las funciones de servicios de semáforo de conteo:

`rt_sem_create()`, `rt_sem_delete()`, `rt_sem_p()`, `rt_sem_v()`, etc.

Las funciones de servicios de cola de mensaje:

`rt_queue_create()`, `rt_queue_delete()`, `rt_queue_alloc()`, `rt_queue_free()`, `rt_queue_send()`, `rt_queue_receive()`, etc.

Las funciones de servicios mutex:

`rt_mutex_create()`, `rt_mutex_delete()`, `rt_mutex_delete()`, `rt_mutex_acquire()`, `rt_mutex_release()`, etc.
Las funciones de servicios de alarma:

`rt_alarm_create()`, `rt_alarm_delete()`, `rt_alarm_start()`, `rt_alarm_stop()`, `rt_alarm_wait()`, etc.

Las funciones de servicios de pila de memoria:

Permite compartir memoria entre procesos y/o reservar un espacio en memoria.

`rt_heap_create()`, `rt_heap_delete()`, `rt_heap_alloc()`, `rt_heap_alloc()` y `rt_heap_bind()`.

La funciones de servicios de variables de condición:
`rt_cond_create()`, `rt_cond_delete()`, `rt_cond_signal()`, `rt_cond_broadcast()` y `rt_cond_wait()`.

En [XenomaiCódigoFuente 16] hay código fuente para Linux en tiempo-real sobre Xenomai. Del mismo modo, en [FREE 16] existe el fichero `rttest.c` que usa la librería POSIX Threads que soporta Xenomai a través de un skin. El siguiente programa se puede compilar para Xenomai.

```
/*
 * Small program to test high-resolution timers and scheduling latency
 * in Unix /Linux
 *
 * Copyright (c) 2007-2008, Free Electrons
 * http://free-electrons.com/labs/solutions/cypfig/rttest.c
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License version
 * 2 as published by the Free Software Foundation.
 */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
```

```
#include <errno.h>
#include <sys/mman.h>

#define MAX(a,b)    (((a) > (b)) ? (a) : (b))
#define MIN(a,b)    (((a) < (b)) ? (a) : (b))

unsigned long long int timespec_diff(struct timespec *t2, struct timespec *t1)
{
       /* Computes the time difference between 2 timespecs */
       /* Assumes that t2 > t1!                    */

       return (t2->tv_sec - t1->tv_sec) * 1000000000ULL + t2->tv_nsec - t1-
>tv_nsec;
}

void timespec_add_ns(struct timespec *ts, unsigned ns)
{
       ts->tv_nsec += ns;
       if (ts->tv_nsec >= 1000000000) {
              ts->tv_nsec -= 1000000000;
              ts->tv_sec++;
       }
}

int main (void)
{
       struct timespec start_time, time1, time2;
       unsigned long long int jitter;
       unsigned long long int min_jit = 999999999999999ULL;
       unsigned long long int max_jit = 0ULL;
       unsigned long long sum_jit = 0ULL;
       unsigned samples = 0;

       mlockall(MCL_CURRENT | MCL_FUTURE);

       /* Display clock resolution */
       clock_getres(CLOCK_MONOTONIC, &time1);
       printf("Clock resolution (ns): %lu\n", time1.tv_nsec);

       /* Initialize the timer that will be used in nanosleep(),  */
       /* to a value of 100 us                                    */

       printf("Measurement, please wait 1 minute...\n");
       fflush(stdout);
       clock_gettime(CLOCK_MONOTONIC, &start_time);

       do {
              /* Get the date before sleeping */
              clock_gettime(CLOCK_MONOTONIC, &time1);

              /* Compute the wake-up date */
              timespec_add_ns(&time1, 100000);

              /* Sleep */
              clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &time1, NULL);
              /* Get the wake-up date */
              clock_gettime(CLOCK_MONOTONIC, &time2);

              /* skip the first second for warmup */
              if (samples >= 1) {
                     /* Compute the sleep time */
                     jitter = timespec_diff(&time2, &time1);
                     min_jit = MIN(min_jit, jitter);
                     max_jit = MAX(max_jit, jitter);
                     sum_jit += jitter;
              }
              ++samples;
       } while (timespec_diff(&time2, &start_time) < 60000000000ULL);
```

```
        /* Display sleeping statistics */
        printf ("Samples: %u\n", --samples);
        printf ("Min latency: %llu us\n", min_jit / 1000);
        printf ("Max latency: %llu us\n", max_jit / 1000);
        printf ("Average latency: %llu us\n", (sum_jit / samples) / 1000);
        exit(EXIT_SUCCESS);
}
```

Figura 39. Programa en C [FREE 16] de Free Electrons para obtener tiempos de latencia en Xenomai

C. Anexo: Resultados obtenidos con procesos (timon_emb de HMI)

Muestra	Error en s	Señal leída
573,585	0,017	0,00E+00
573,6	0,015	0,00E+00
1697,937	0,017	0,00E+00
1697,952	0,015	0,00E+00
1771,153	0,017	0,00E+00
1771,168	0,015	0,00E+00
1826,45	0,018	0,00E+00
1826,464	0,014	0,00E+00
1840,785	0,017	0,00E+00
1840,8	0,015	0,00E+00
1907,974	0,022	0,00E+00
1907,985	0,011	0,00E+00
1908	0,015	0,00E+00
2155,153	0,017	0,00E+00
2155,168	0,015	0,00E+00
2949,266	0,018	0,00E+00
2949,28	0,014	0,00E+00
3126,931	0,019	0,00E+00
3126,945	0,014	0,00E+00
3126,96	0,015	0,00E+00
3183,249	0,017	0,00E+00
3183,264	0,015	0,00E+00
4691,089	0,017	0,00E+00
4691,104	0,015	0,00E+00
4847,25	0,018	0,00E+00
4847,264	0,014	0,00E+00
5422,738	0,018	0,00E+00
5422,752	0,014	0,00E+00
5461,65	0,018	0,00E+00
5461,664	0,014	0,00E+00
5553,811	0,019	0,00E+00
5553,824	0,013	0,00E+00
5555,859	0,019	0,00E+00
5555,872	0,013	0,00E+00
5611,153	0,017	0,00E+00
5611,168	0,015	0,00E+00
5611,251	0,019	0,00E+00
5611,264	0,013	0,00E+00
5934,225	0,017	0,00E+00
5934,24	0,015	0,00E+00
5998,225	0,017	0,00E+00

5998,24	0,015	0,00E+00
6290,578	0,018	0,00E+00
6290,592	0,014	0,00E+00
7211,154	0,018	0,00E+00
7211,168	0,014	0,00E+00
7386,465	0,017	0,00E+00
7386,48	0,015	0,00E+00
7444,113	0,017	0,00E+00
7444,128	0,015	0,00E+00
7767,186	0,018	0,00E+00
7767,2	0,014	0,00E+00
8460,947	0,019	0,00E+00
8460,96	0,013	0,00E+00
8497,089	0,017	0,00E+00
8497,104	0,015	0,00E+00
8538,259	0,019	0,00E+00
8538,272	0,013	0,00E+00
8770,193	0,017	0,00E+00
8770,208	0,015	0,00E+00
8772,754	0,018	0,00E+00
8772,768	0,014	0,00E+00
8877,201	0,017	0,00E+00
8877,216	0,015	0,00E+00
8903,825	0,017	0,00E+00
8903,84	0,015	0,00E+00
8950,93	0,018	0,00E+00
8950,944	0,014	0,00E+00
8956,947	0,019	0,00E+00
8956,96	0,013	0,00E+00
9002,13	0,018	0,00E+00
9002,144	0,014	0,00E+00
9129,619	0,019	0,00E+00
9129,632	0,013	0,00E+00
9289,875	0,019	0,00E+00
9289,888	0,013	0,00E+00
9292,435	0,019	0,00E+00
9292,448	0,013	0,00E+00
9310,354	0,018	0,00E+00
9310,368	0,014	0,00E+00
9397,397	0,021	0,00E+00
9397,408	0,011	0,00E+00
9398,418	0,018	0,00E+00
9398,432	0,014	0,00E+00
9529,491	0,019	0,00E+00
9529,504	0,013	0,00E+00

9622,674	0,018	0,00E+00
9622,688	0,014	0,00E+00
9658,514	0,018	0,00E+00
9658,528	0,014	0,00E+00
9667,217	0,017	0,00E+00
9667,232	0,015	0,00E+00
9730,181	0,021	0,00E+00
9730,192	0,011	0,00E+00
9794,193	0,017	0,00E+00
9794,208	0,015	0,00E+00
9858,193	0,017	0,00E+00
9858,208	0,015	0,00E+00
9903,762	0,018	0,00E+00
9903,776	0,014	0,00E+00
10411,155	0,019	0,00E+00
10411,168	0,013	0,00E+00
10662,546	0,018	0,00E+00
10662,56	0,014	0,00E+00
10841,346	0,018	0,00E+00
10841,36	0,014	0,00E+00
11148,946	0,018	0,00E+00
11148,96	0,014	0,00E+00
11149,969	0,017	0,00E+00
11149,984	0,015	0,00E+00
11178,129	0,017	0,00E+00
11178,144	0,015	0,00E+00
11368,595	0,019	0,00E+00
11368,608	0,013	0,00E+00
11410,578	0,018	0,00E+00
11410,592	0,014	0,00E+00
11553,937	0,017	0,00E+00
11553,952	0,015	0,00E+00
11644,163	0,019	0,00E+00
11644,176	0,013	0,00E+00
12547,217	0,017	0,00E+00
12547,232	0,015	0,00E+00
13943,442	0,018	0,00E+00
13943,456	0,014	0,00E+00
14156,435	0,019	0,00E+00
14156,448	0,013	0,00E+00
14162,578	0,018	0,00E+00
14162,592	0,014	0,00E+00
14499,477	0,021	0,00E+00
14499,488	0,011	0,00E+00
14783,634	0,018	0,00E+00

14783,648	0,014	0,00E+00
14930,578	0,018	0,00E+00
14930,592	0,014	0,00E+00
15379,602	0,018	0,00E+00
15379,616	0,014	0,00E+00
15442,578	0,018	0,00E+00
15442,592	0,014	0,00E+00
15922,002	0,018	0,00E+00
15922,016	0,014	0,00E+00

Tabla 6. Errores de lecturas distintas a 16 ms con kernel vanilla (3.13.0-32-generic)

Muestra	**Error en s**	Señal leída
192,753	**0,017**	0,00E+00
192,768	**0,015**	0,00E+00
3968,689	**0,017**	0,00E+00
3968,704	**0,015**	0,00E+00
3976,705	**0,017**	0,00E+00
3976,72	**0,015**	0,00E+00
5085,985	**0,017**	0,00E+00
5086	**0,015**	0,00E+00
7379,281	**0,017**	0,00E+00
7379,296	**0,015**	0,00E+00
7869,697	**0,017**	0,00E+00
7869,712	**0,015**	0,00E+00

Tabla 7. Errores de lecturas distintas a 16 ms con PREEMPT_RT (3.14.2)

D. Anexo: Optimización del proceso de actualización de Linux a través de dispositivo USB autoarrancable.

Para añadir esta funcionalidad hacemos:

Descargamos `ubuntu-14.04.4-server-i386.iso` al escritorio (`/home/real-time/Escritorio`).

Lo podemos descargar desde esta URL al escritorio:

http://releases.ubuntu.com/14.04/ubuntu-14.04.4-server-i386.iso

Abrimos una terminal y ejecutamos:

```
sudo –s
cd /home/real-time/Escritorio
```

Comprobamos con md5 el fichero `ubuntu-14.04.4-server-i386.iso`.

MD5 proporciona la seguridad de que un archivo descargado de Internet no se ha alterado. Podemos descargar el resumen MD5 desde esta URL al escritorio:

http://releases.ubuntu.com/14.04/MD5SUMS

Conectamos USB 1.

Iniciamos la máquina virtual y, sobre USB 1, seguimos los pasos de instalación de la misma forma que las siguientes figuras:

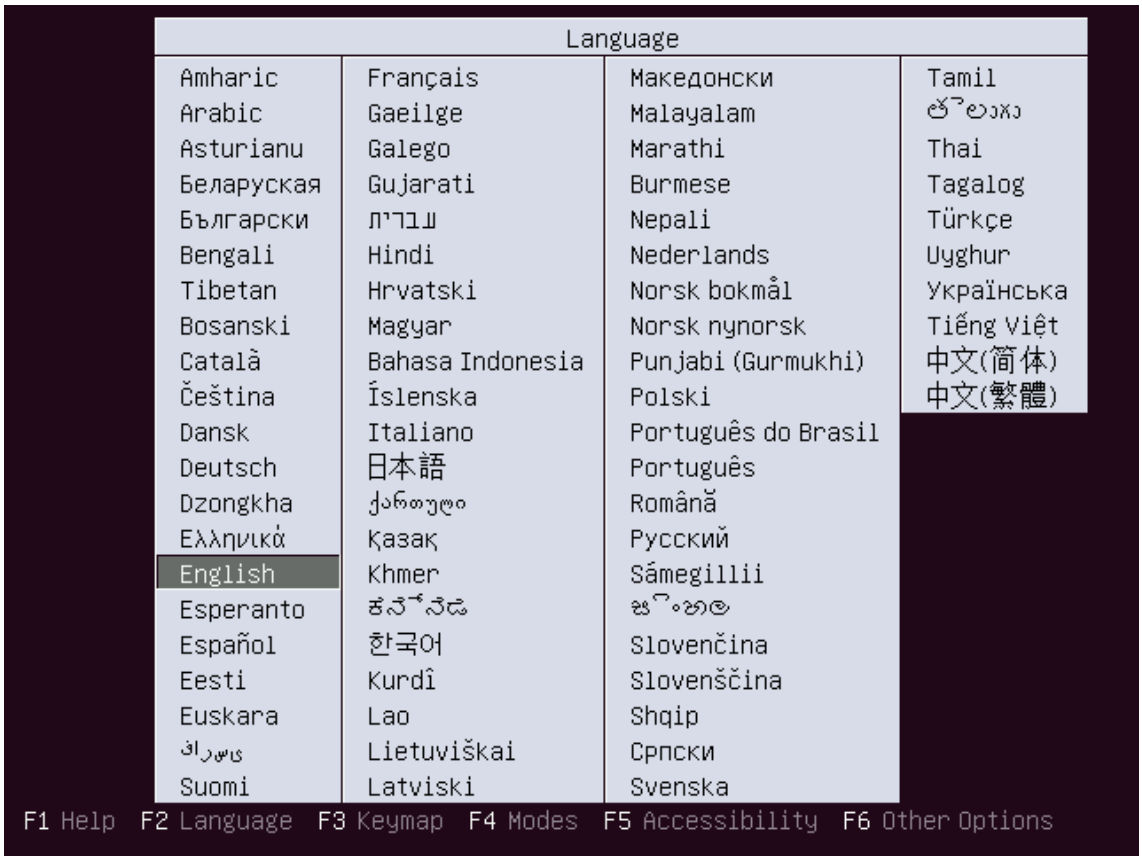


Figura 40. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 1

Pulsamos *F4 Modes*. Elegimos *Install a minimal system*.

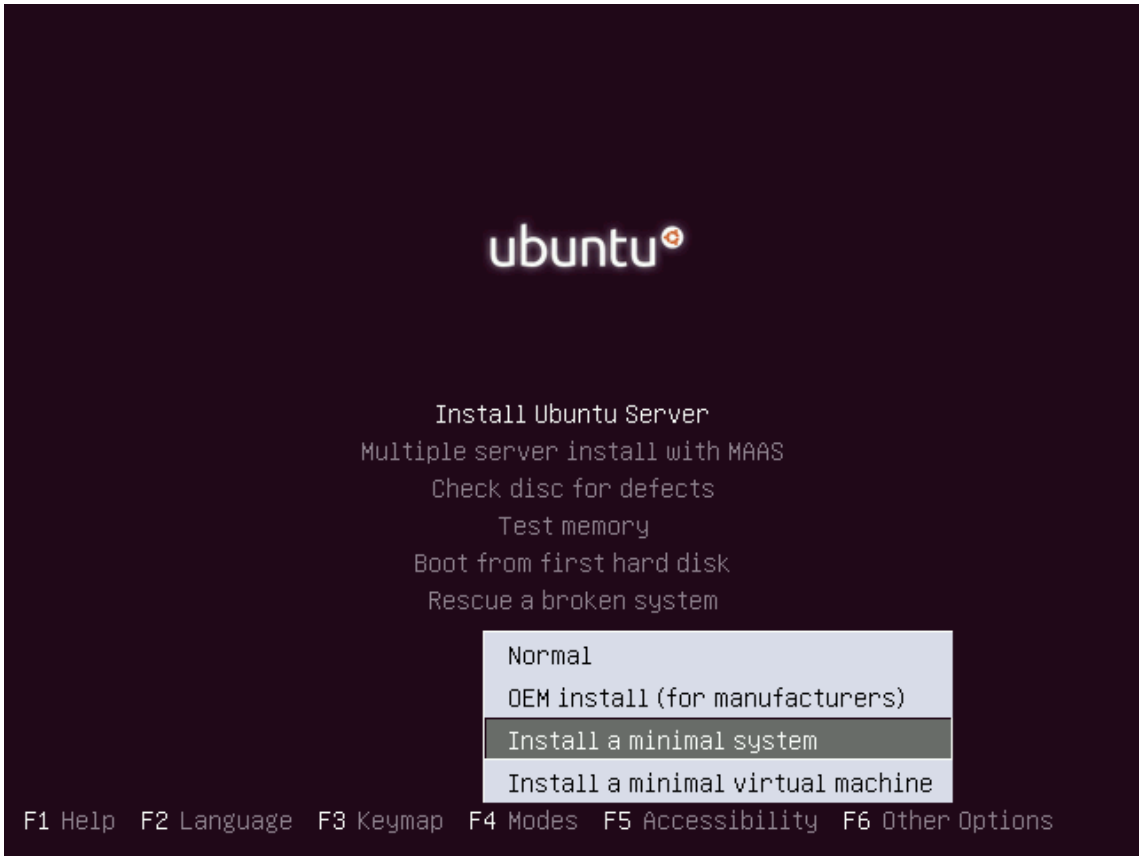


Figura 41. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 2

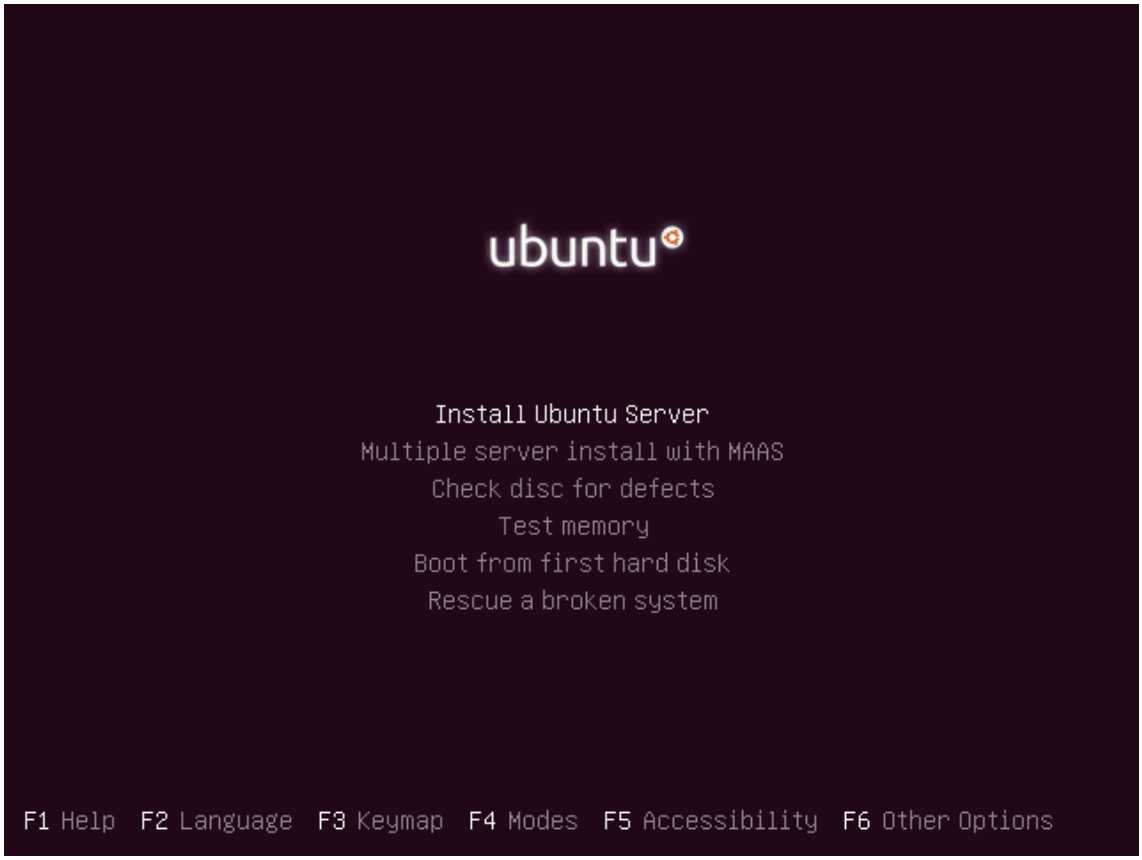


Figura 42. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 3

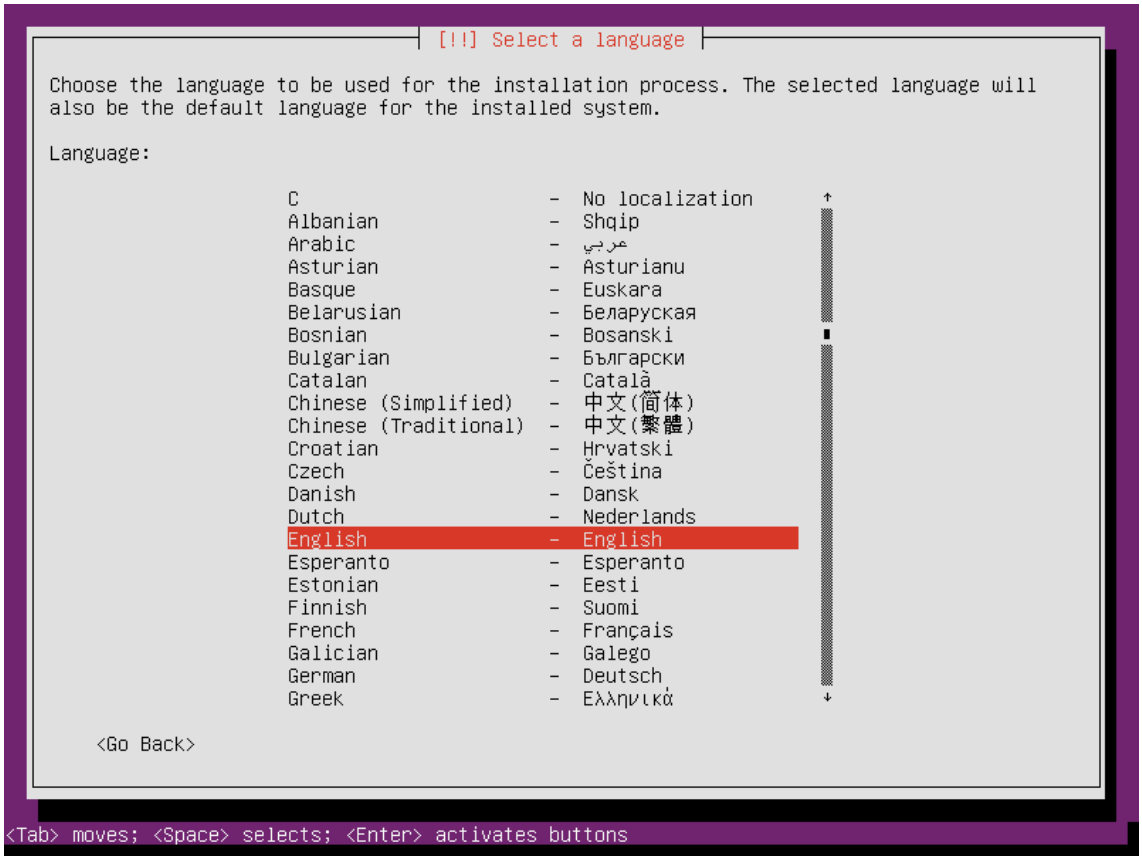


Figura 43. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 4

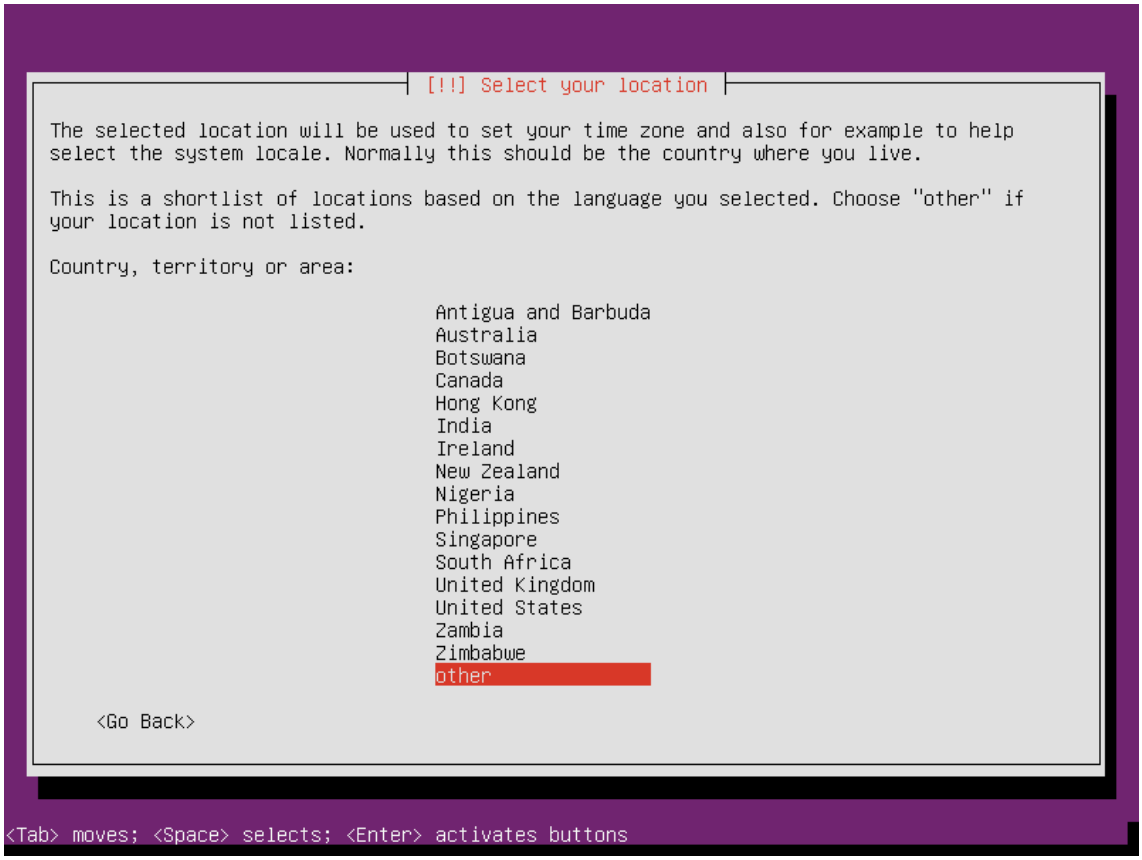


Figura 44. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 5

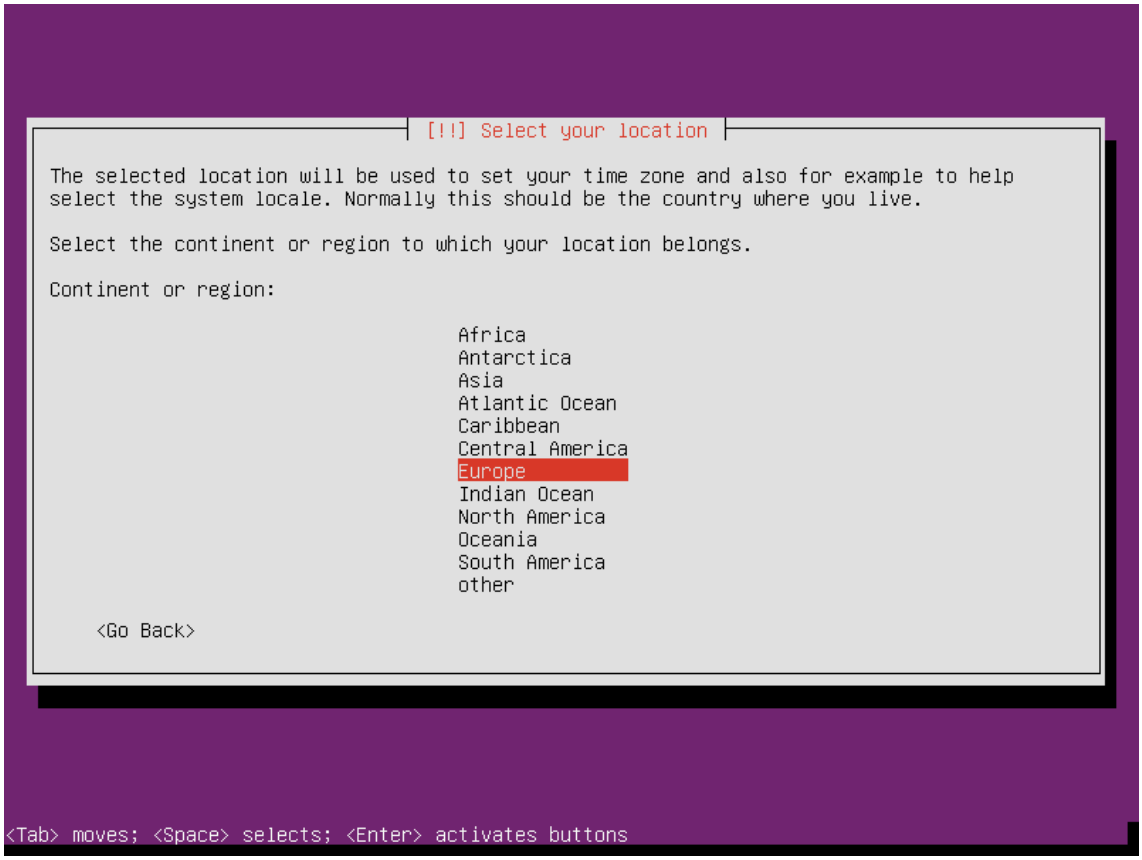


Figura 45. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 6

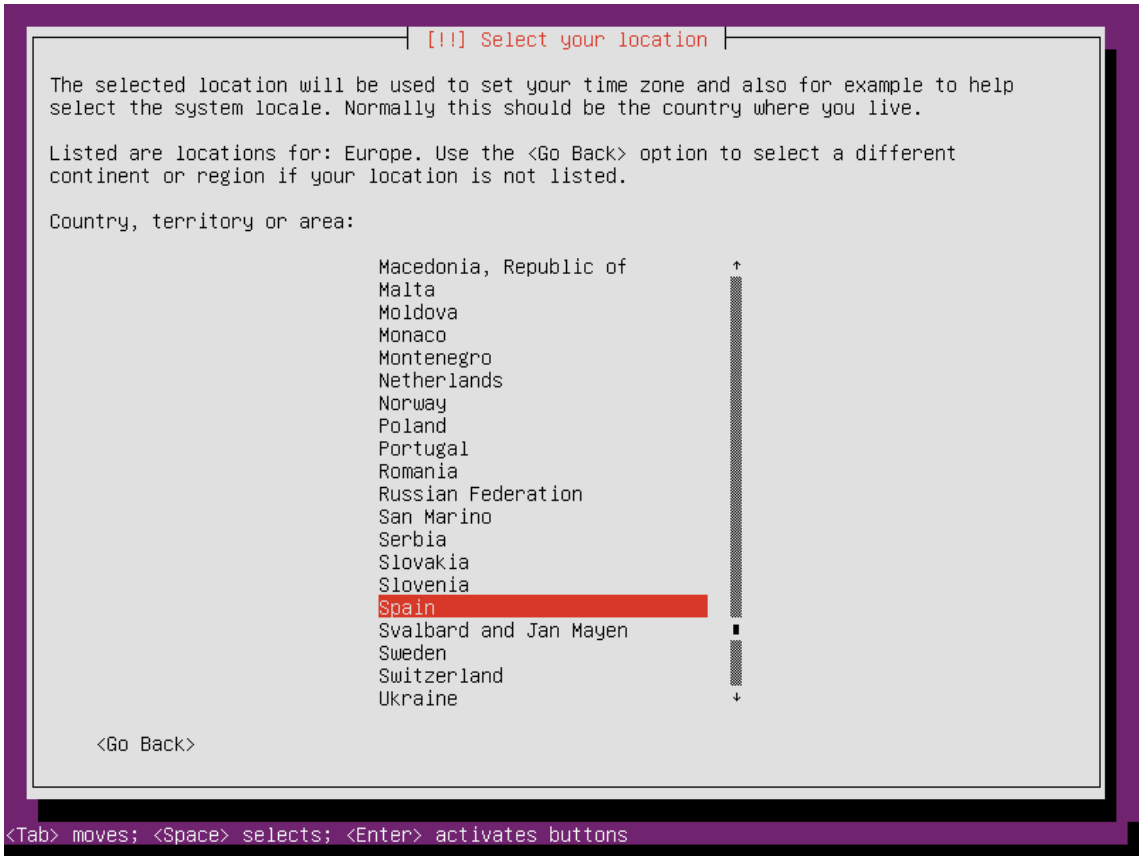


Figura 46. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 7

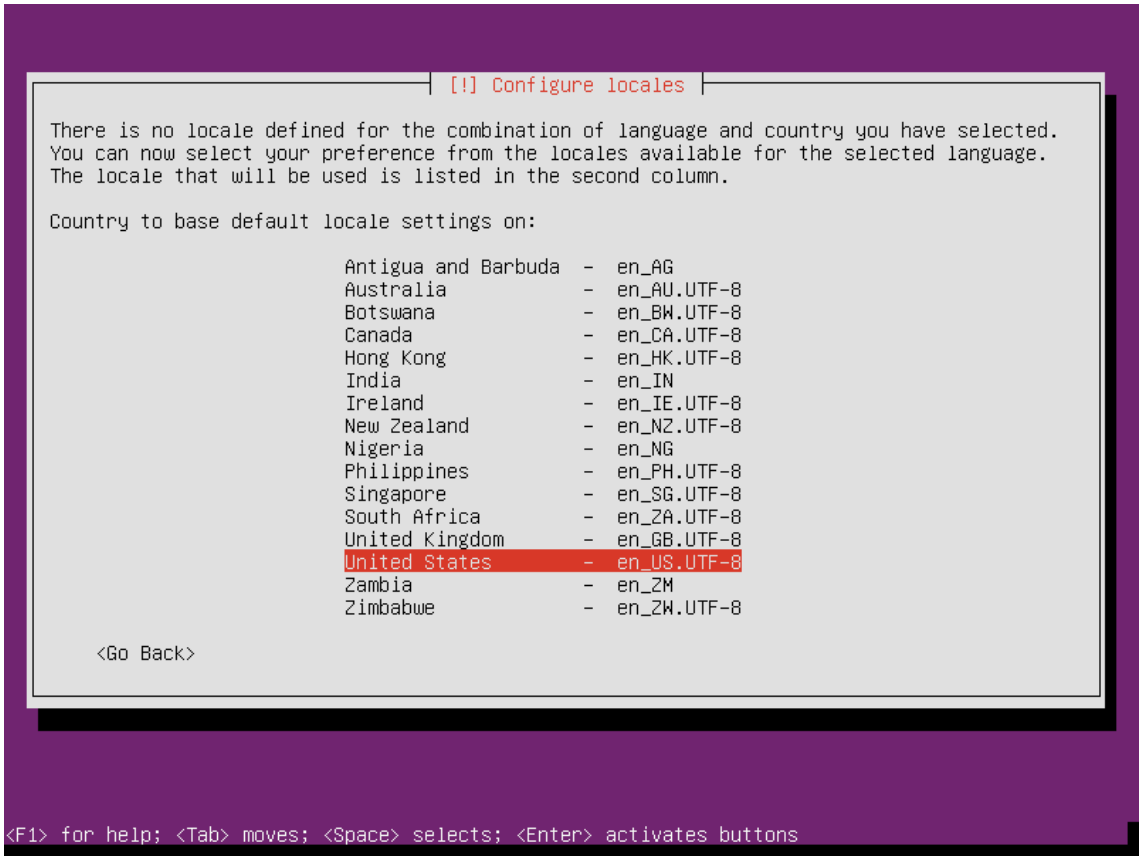


Figura 47. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 8

Figura 48. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 9

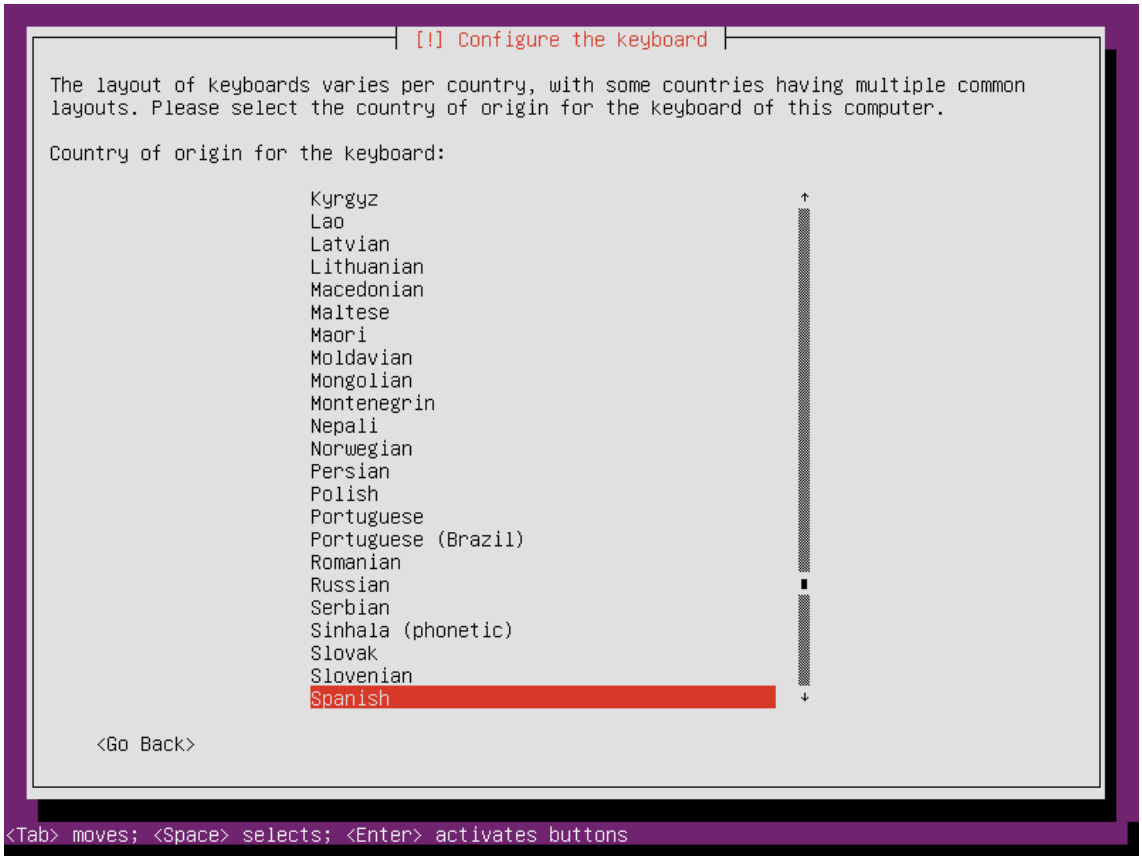


Figura 49. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 10

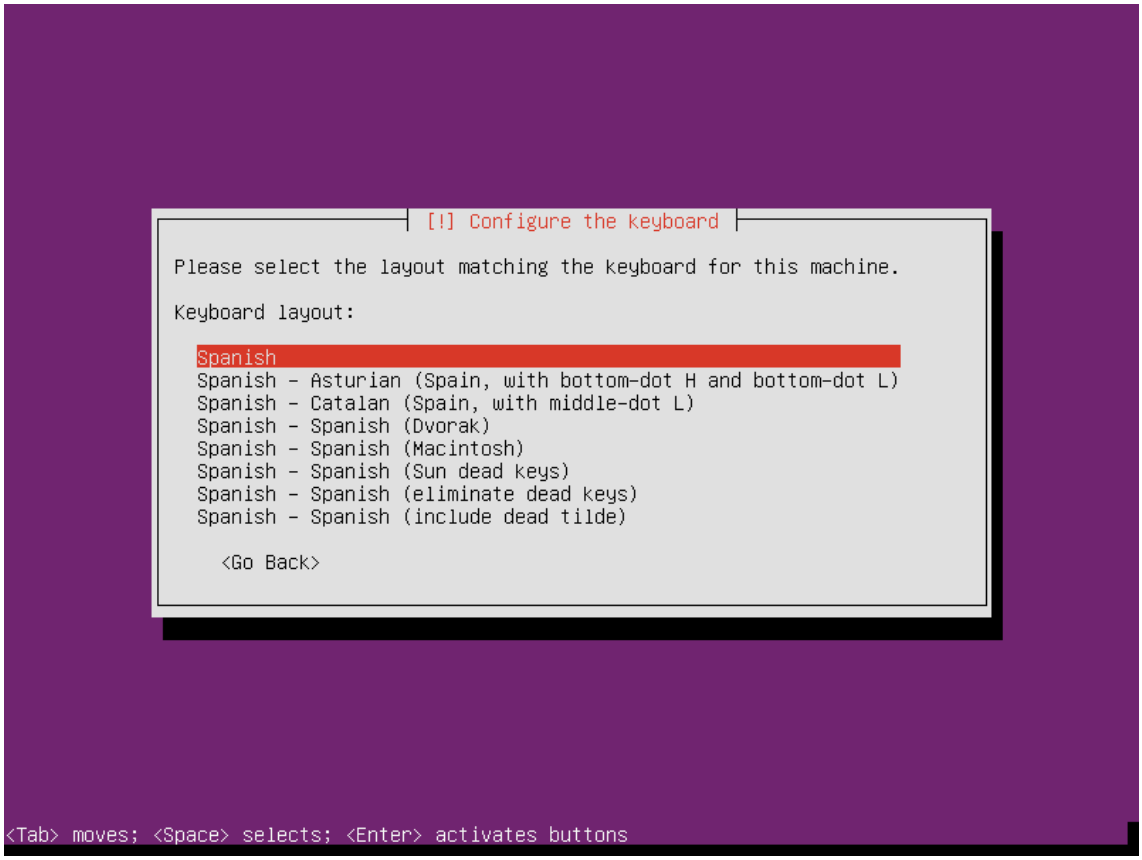


Figura 50. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 11

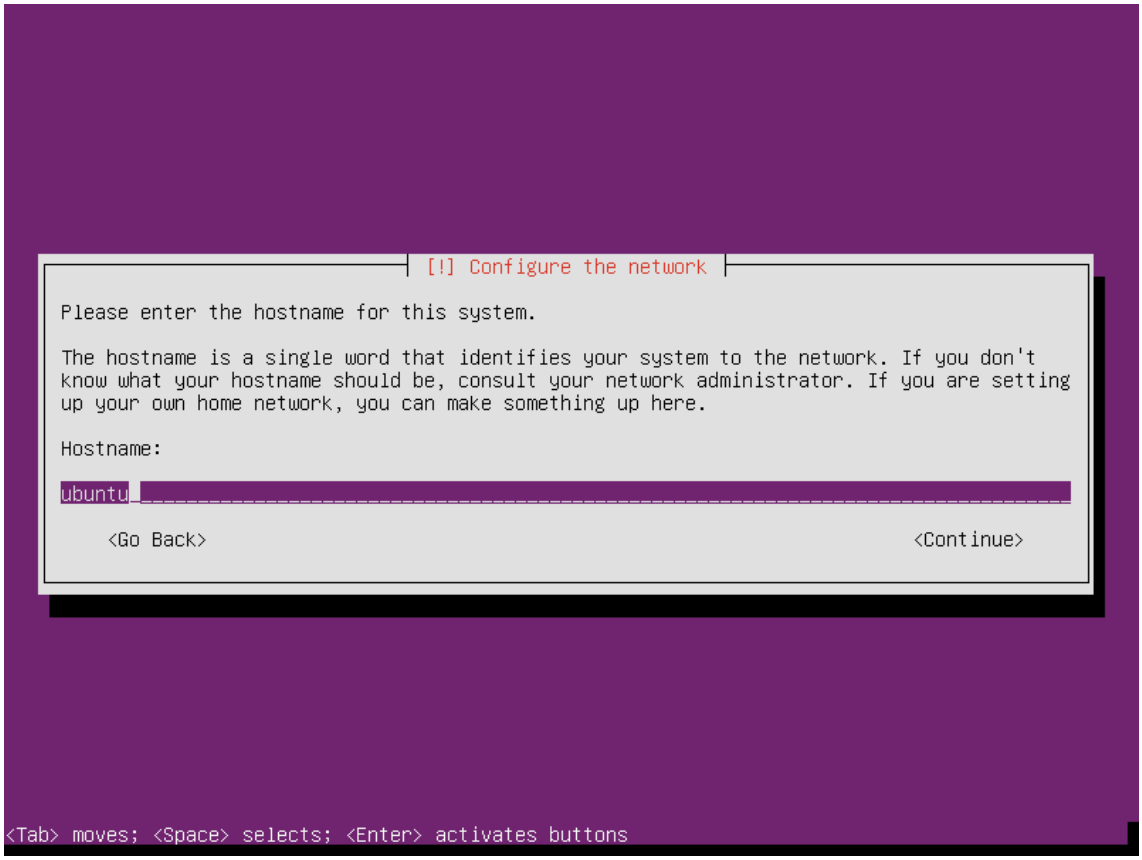


Figura 51. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 12

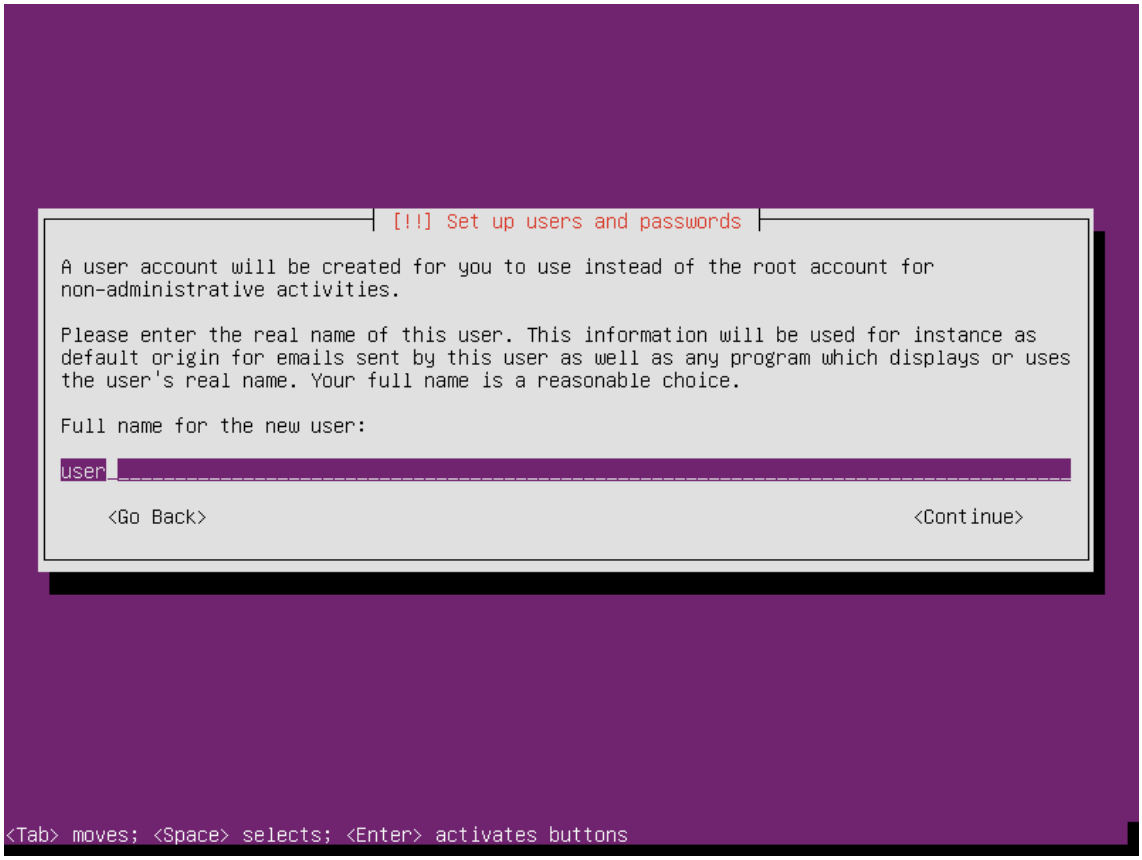


Figura 52. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 13

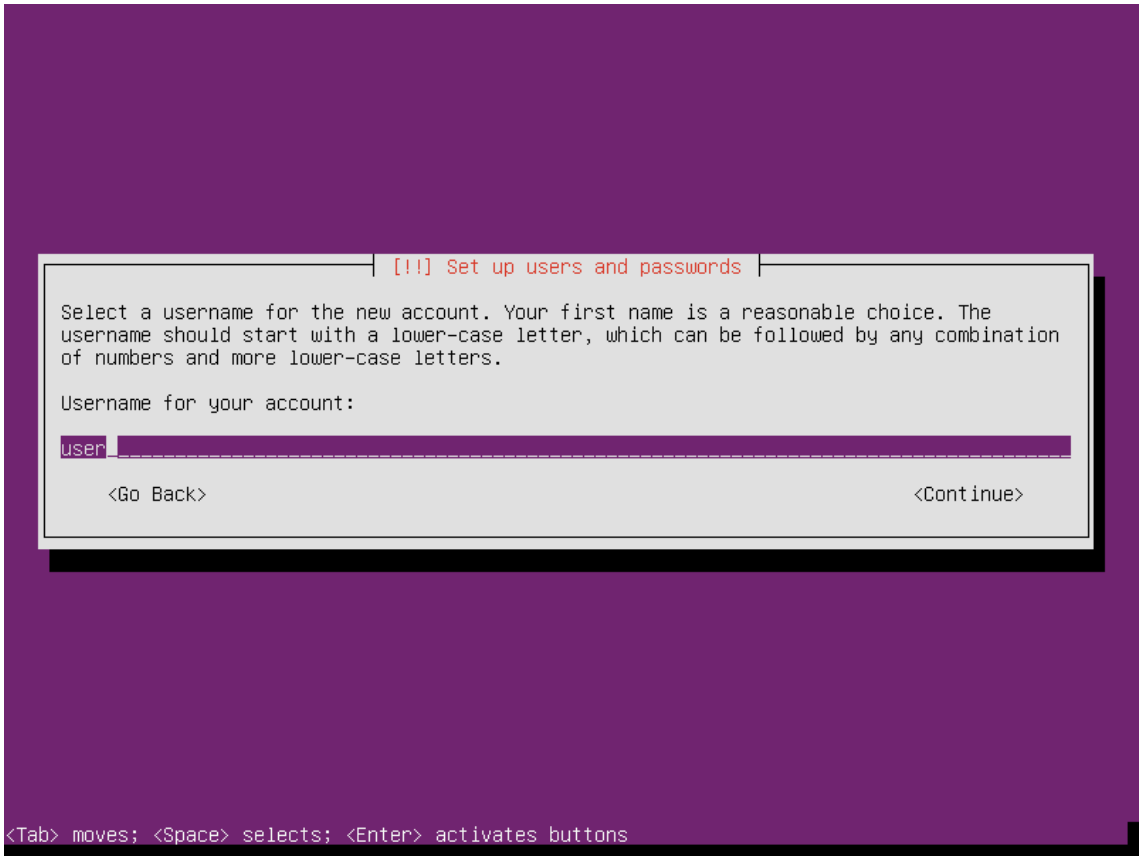


Figura 53. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 14

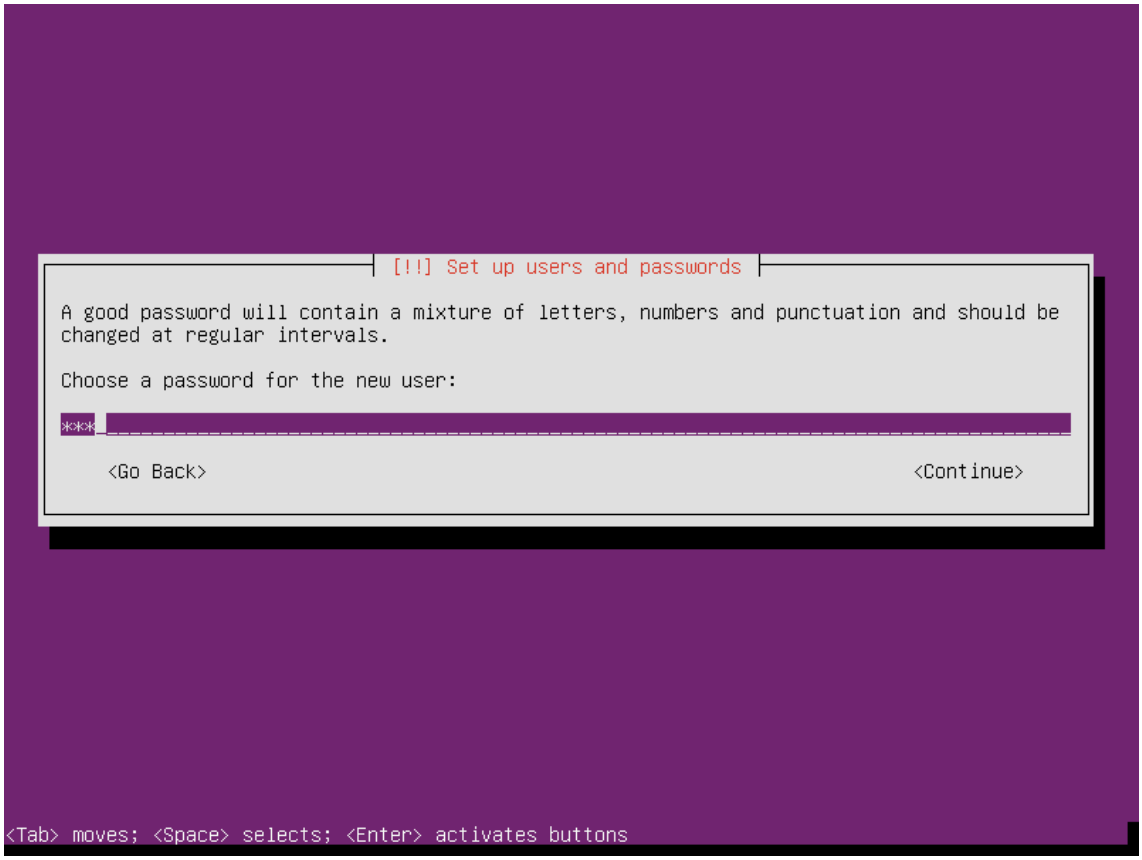


Figura 54. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 15

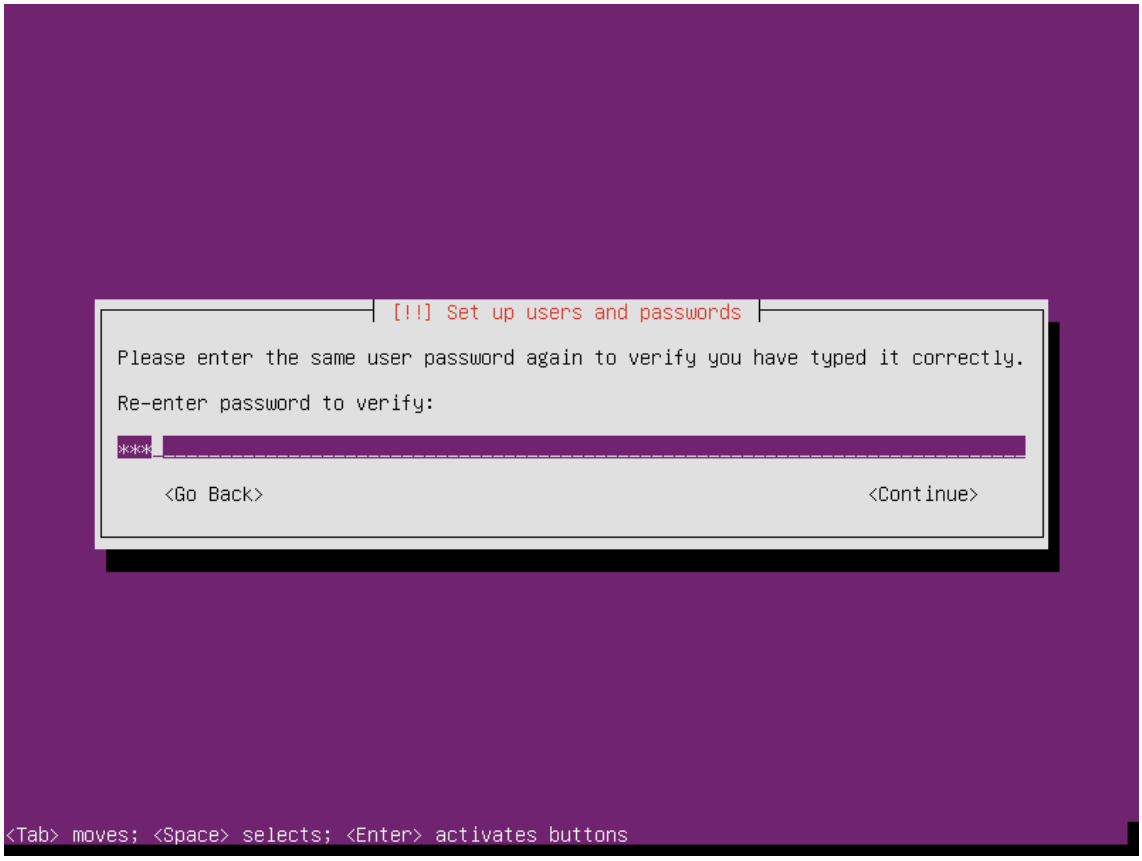


Figura 55. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 16

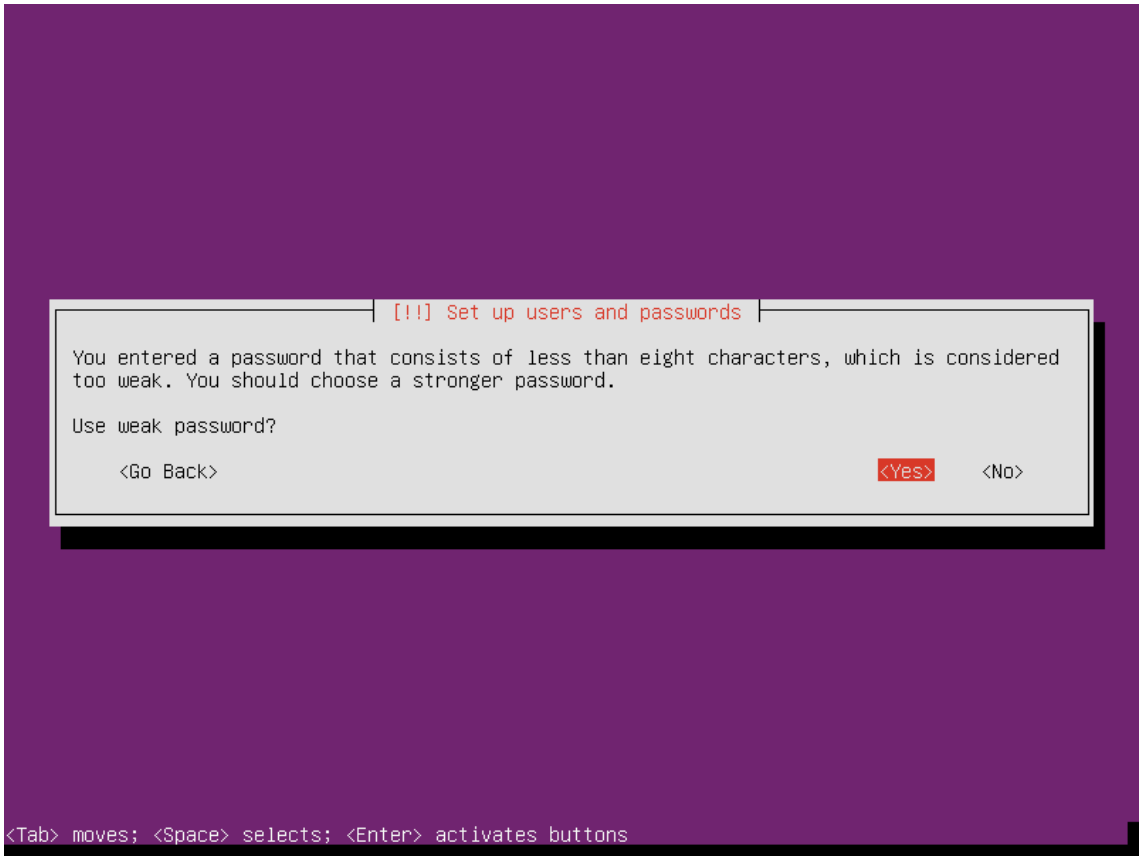


Figura 56. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 17

Figura 57. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 18

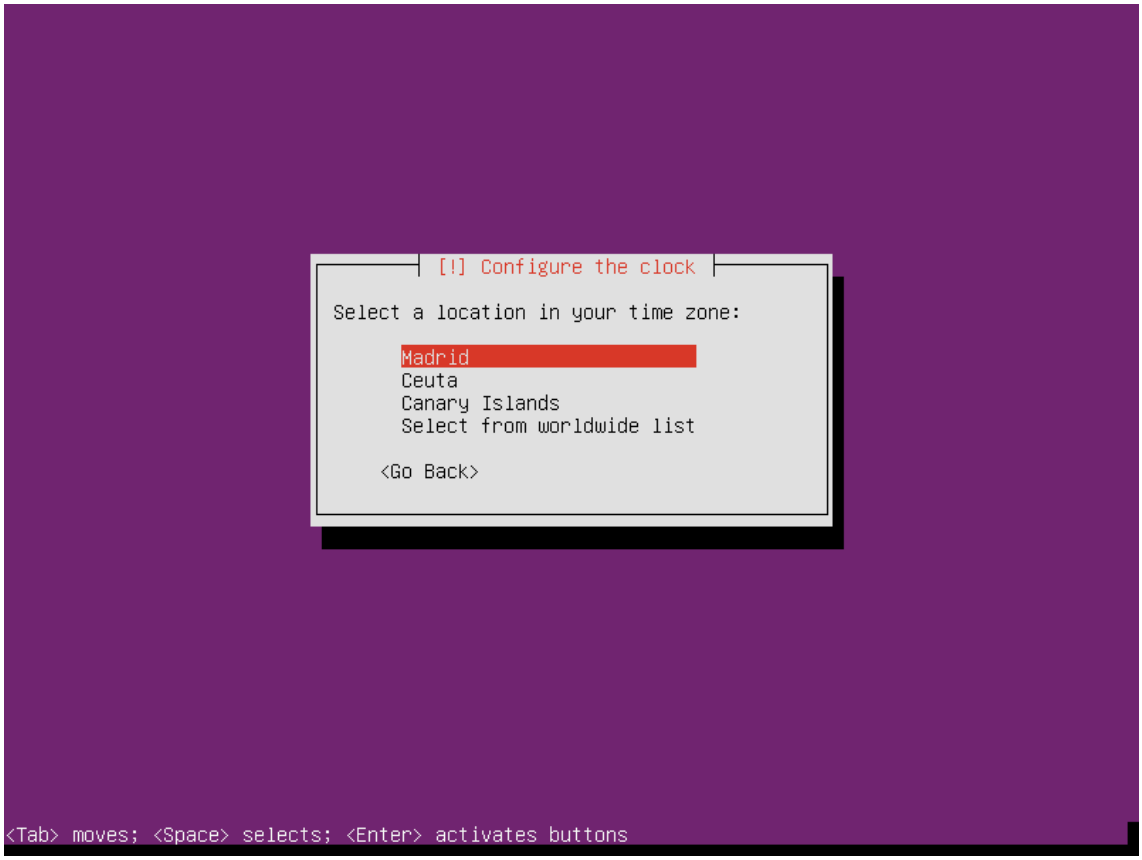


Figura 58. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 19

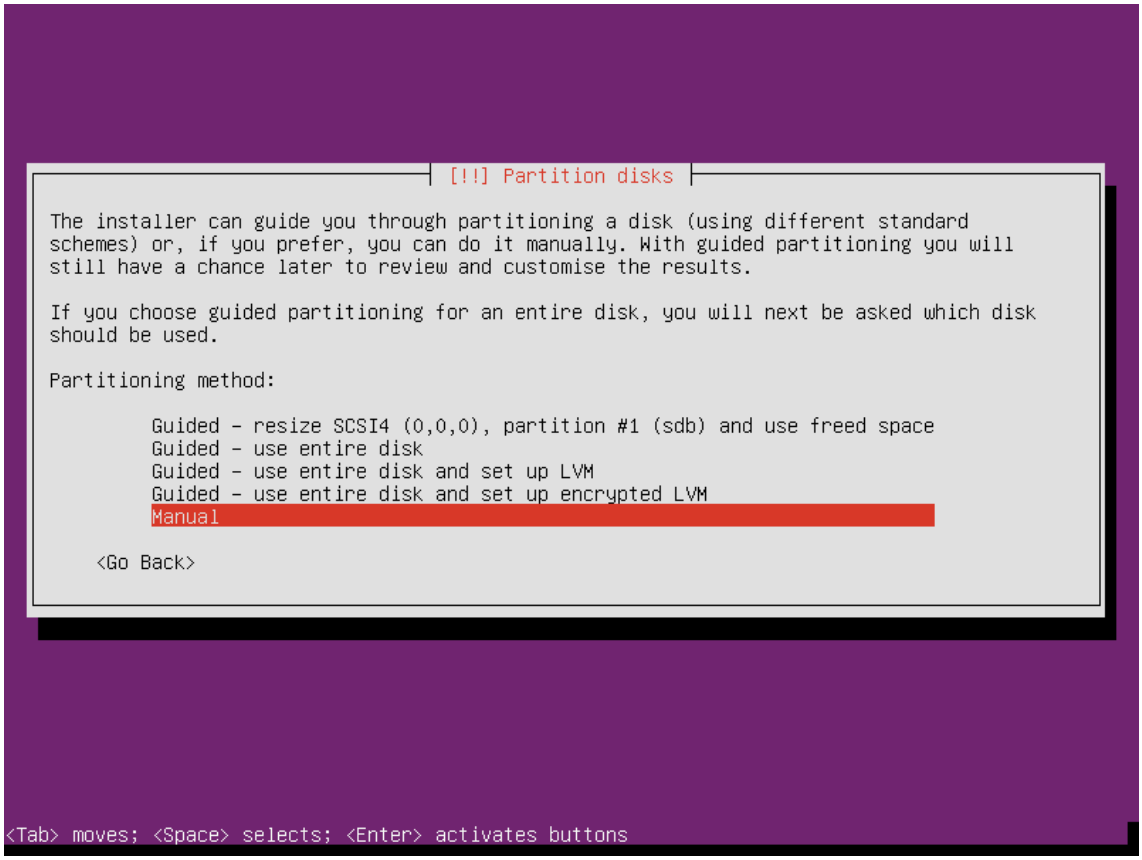


Figura 59. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 20

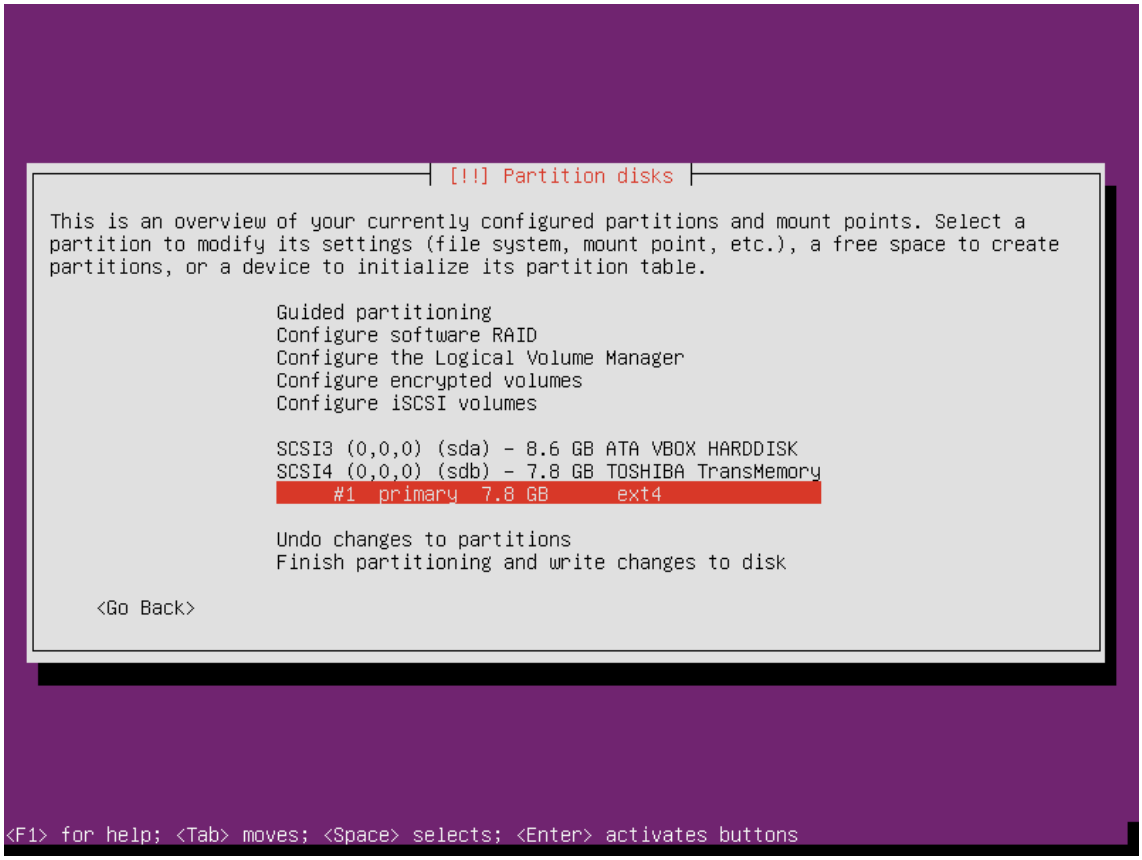


Figura 60. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 21

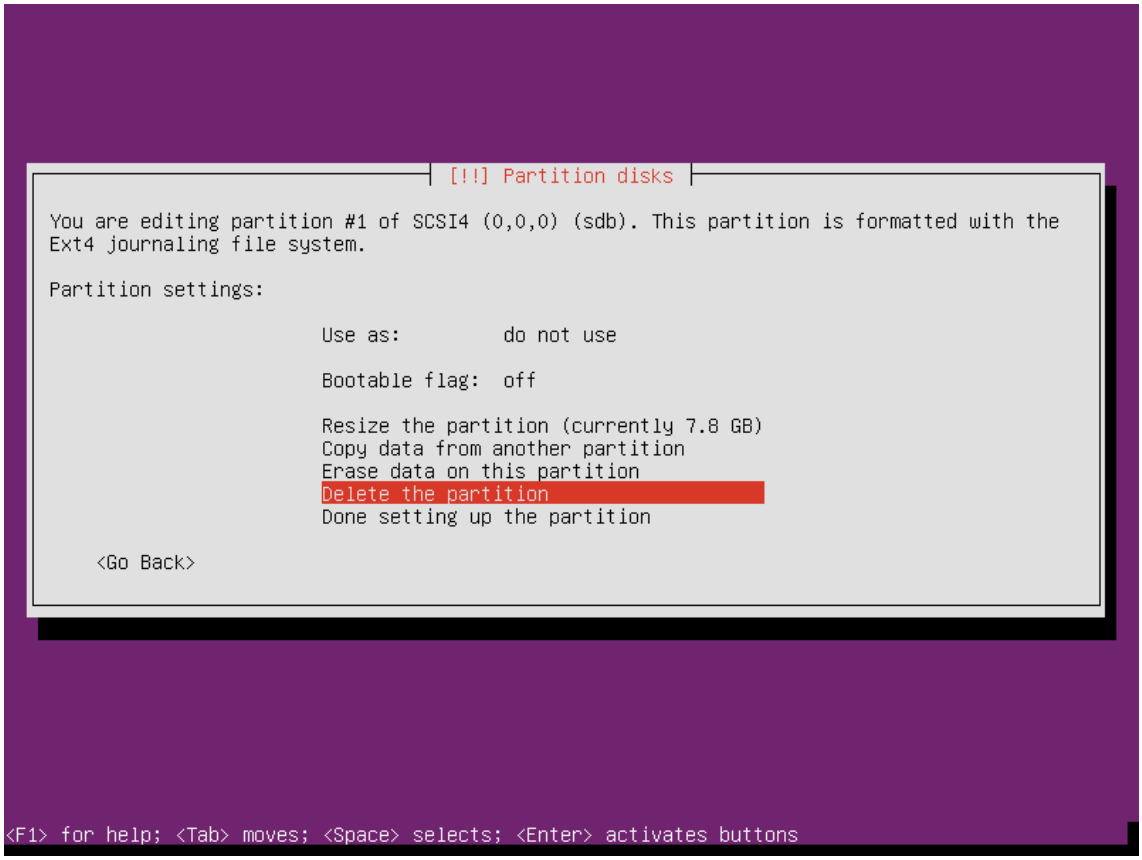


Figura 61. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 22

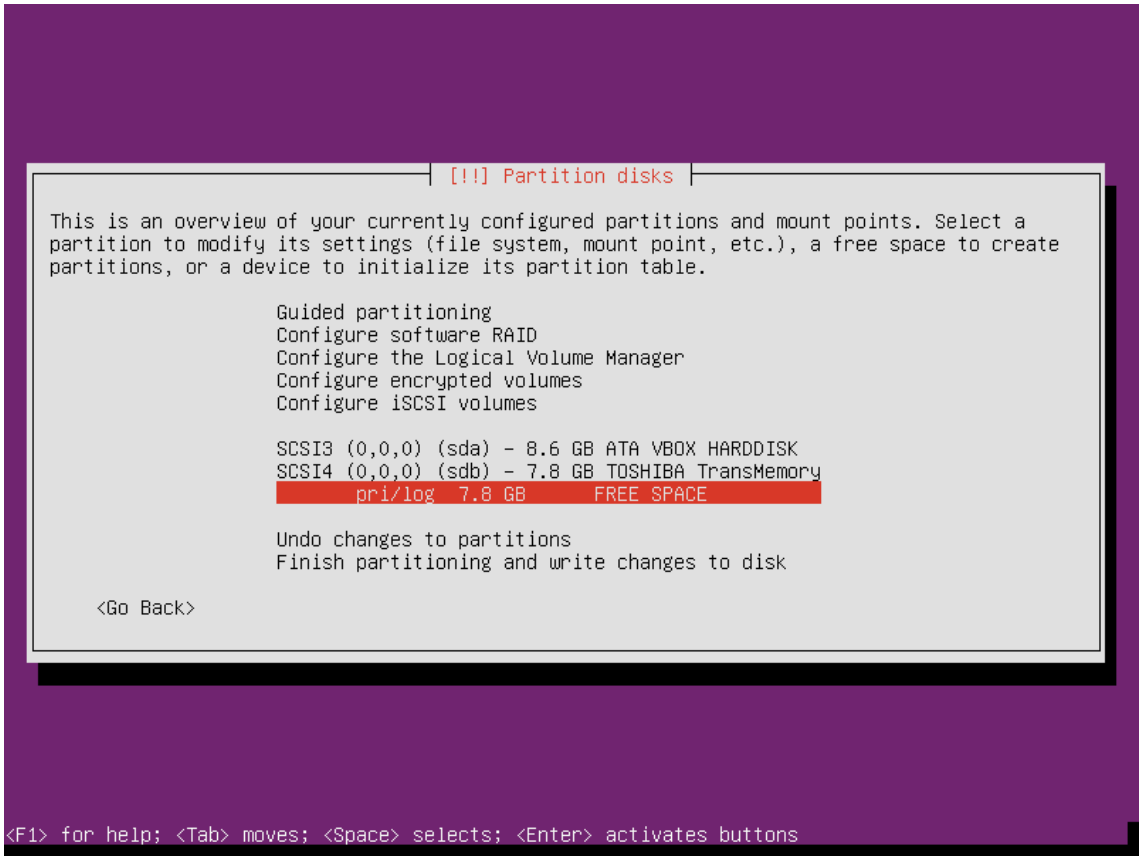


Figura 62. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 23

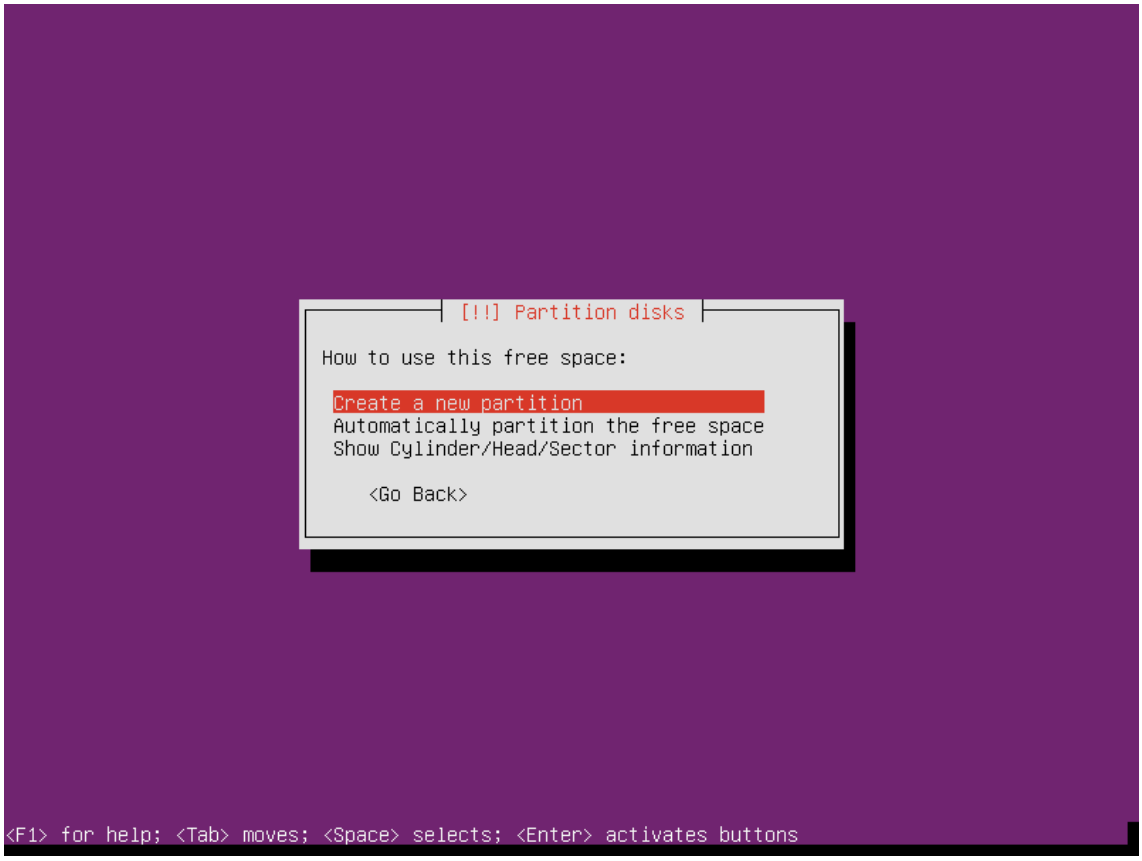


Figura 63. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 24

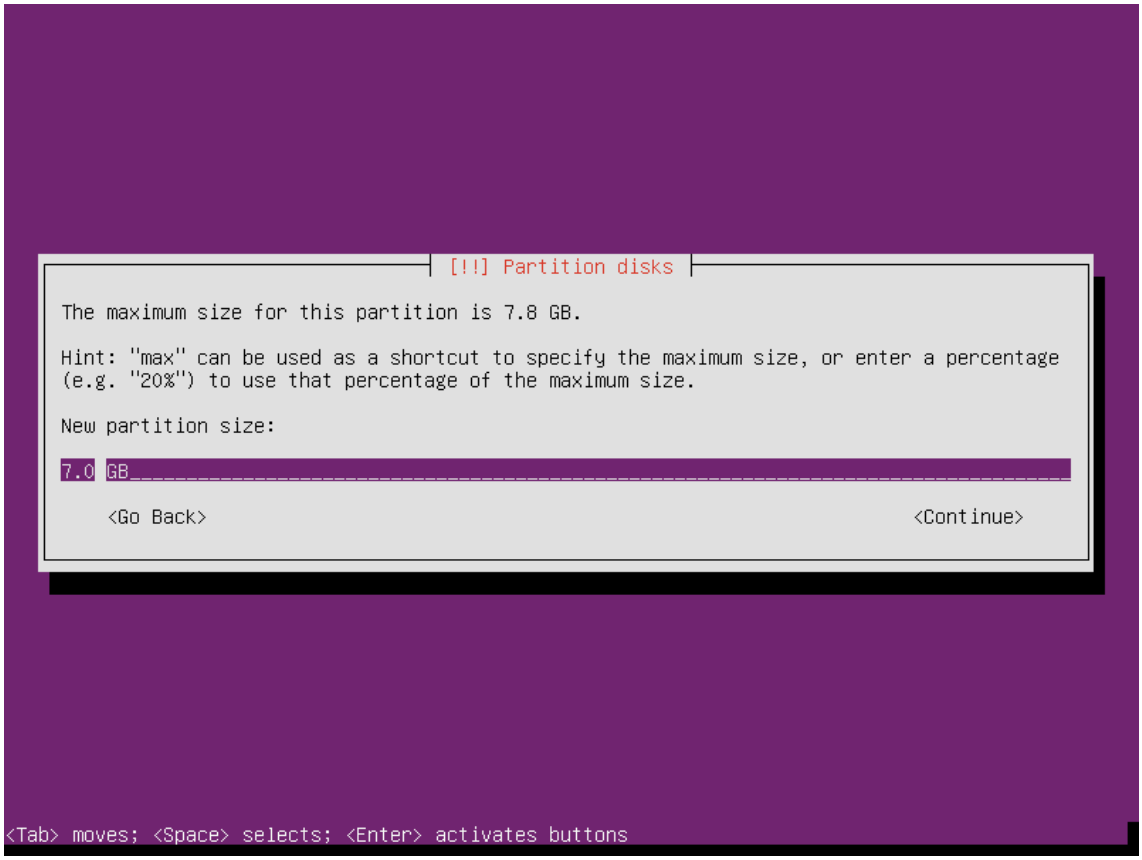


Figura 64. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 25

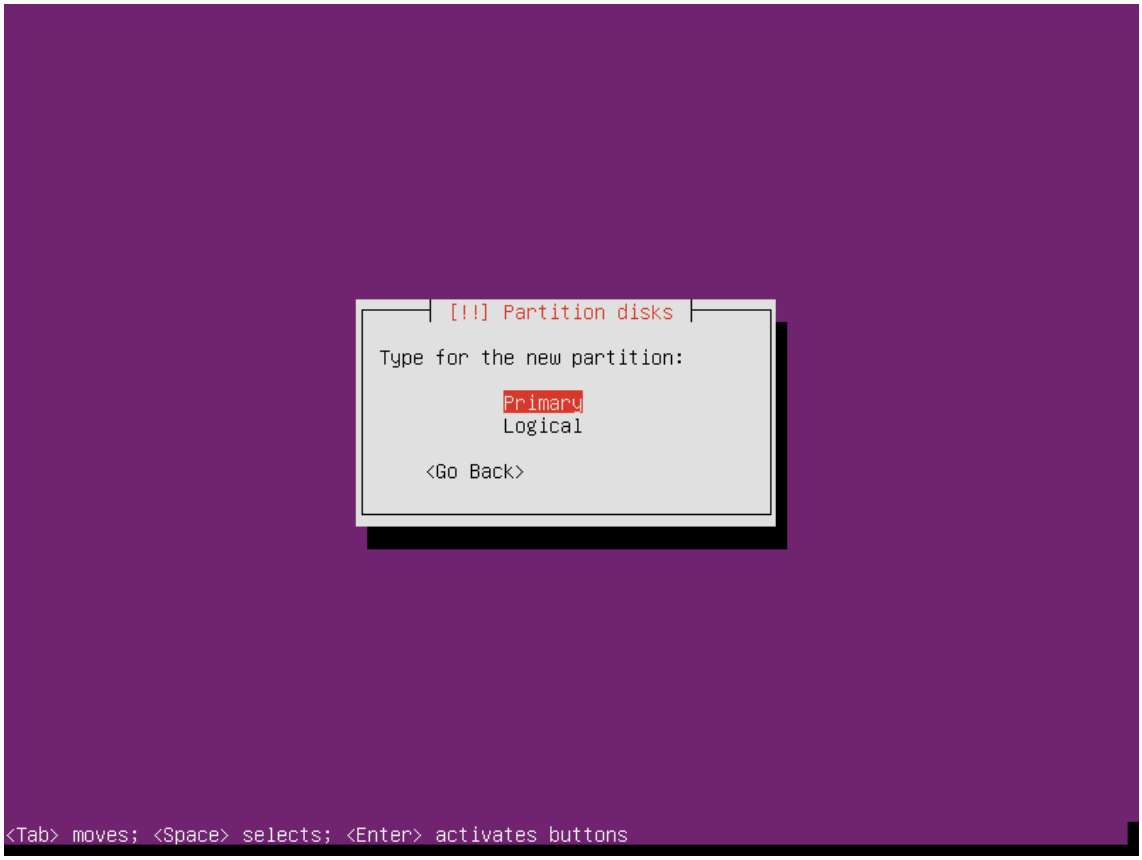


Figura 65. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 26

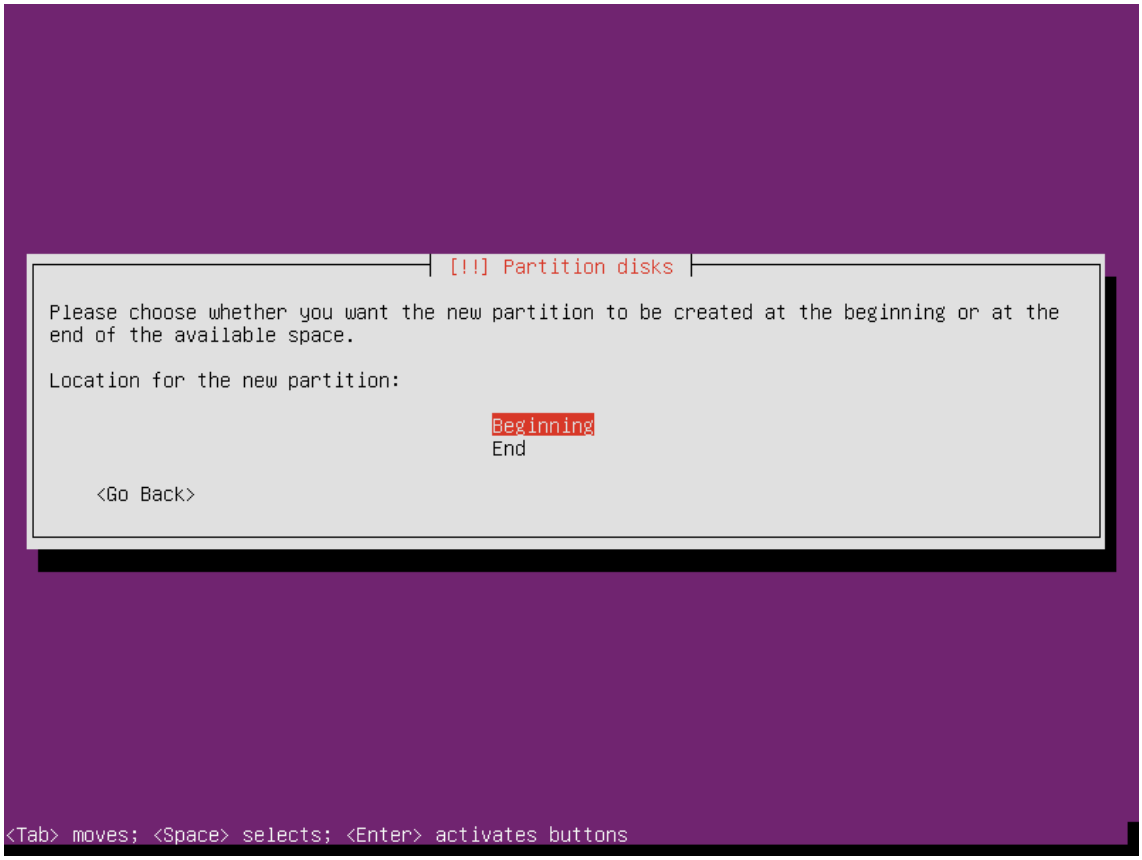


Figura 66. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 27

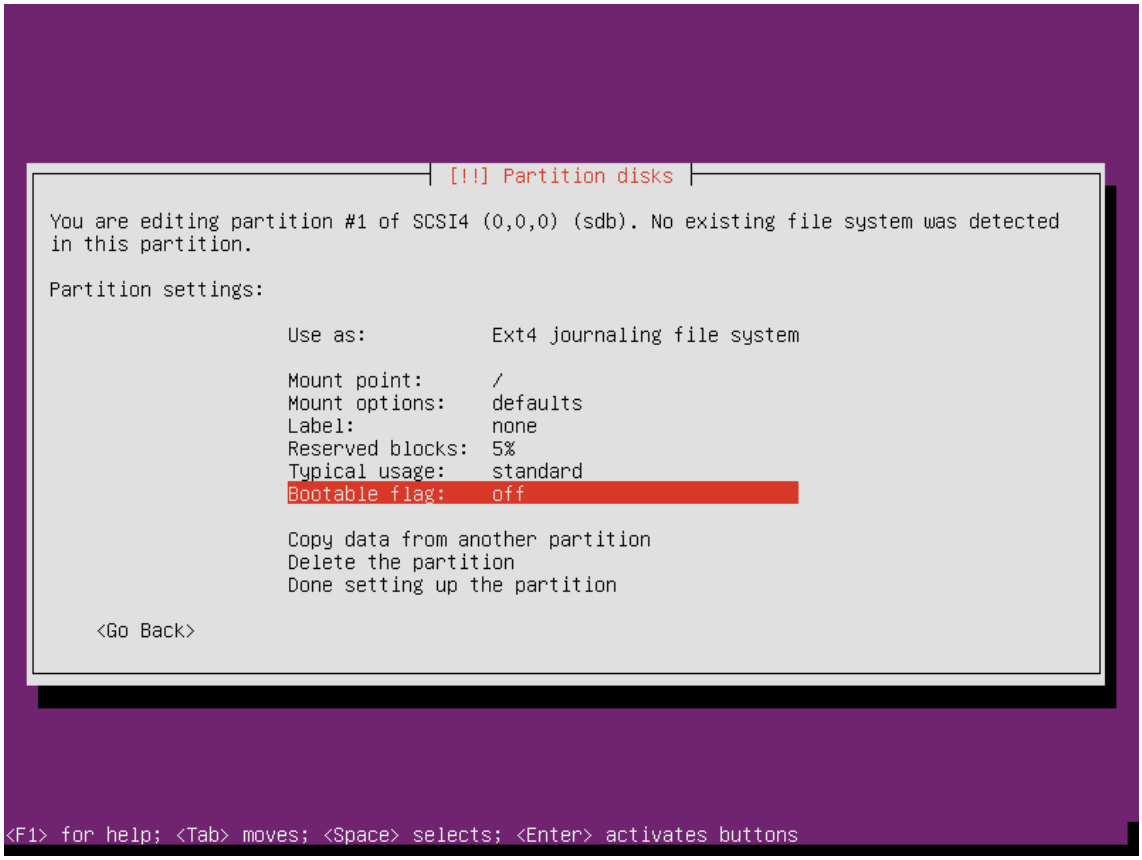


Figura 67. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 28

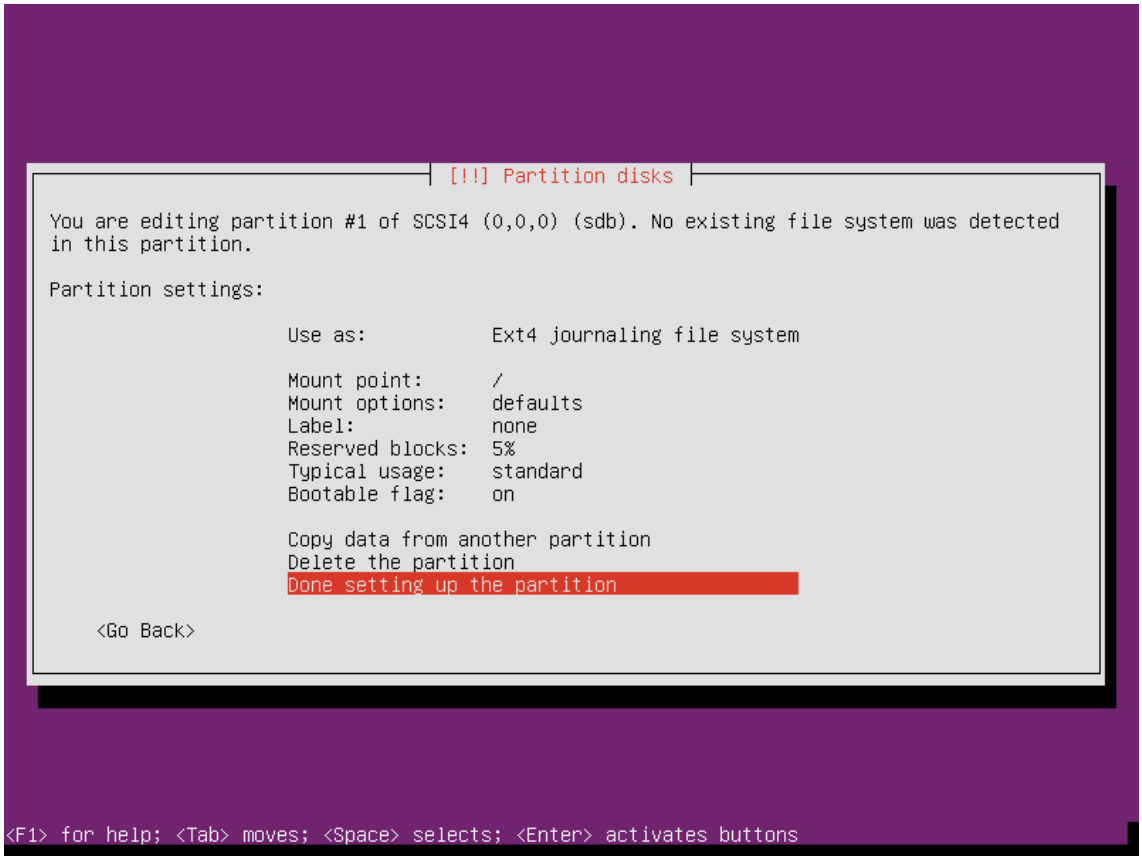


Figura 68. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 29

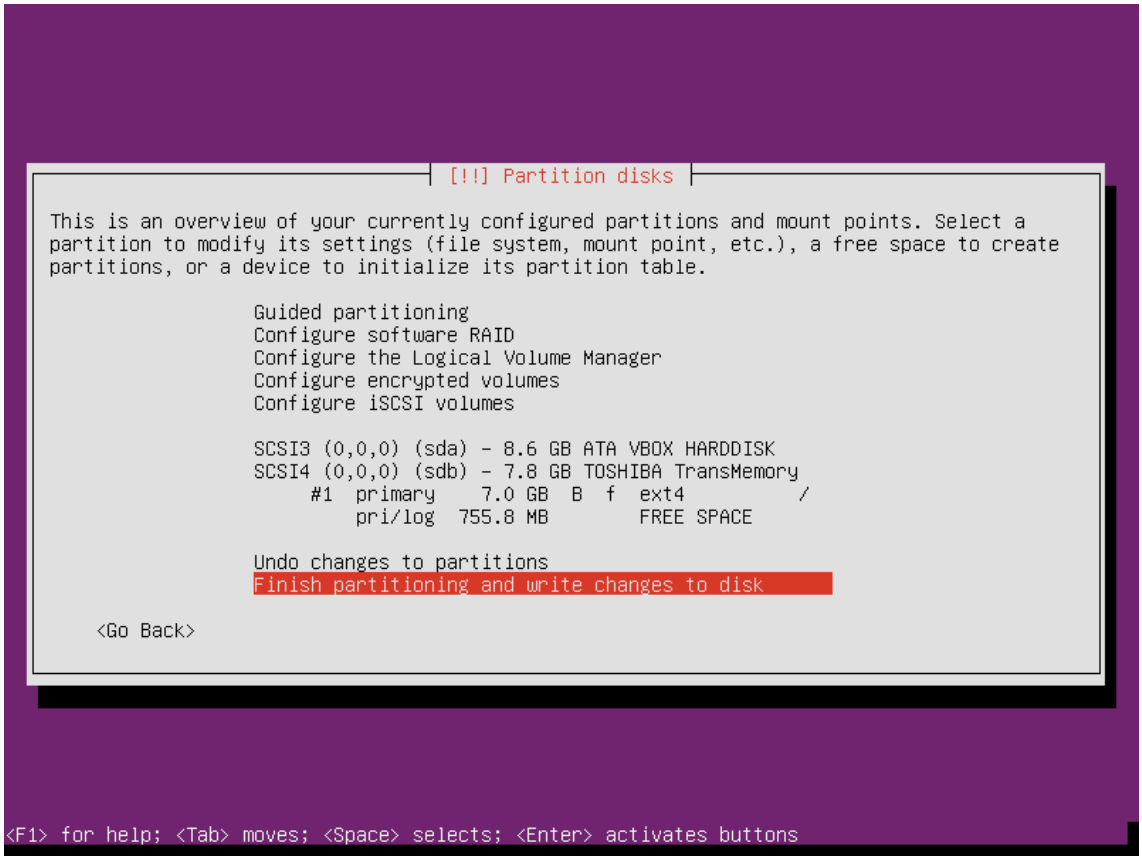


Figura 69. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 30

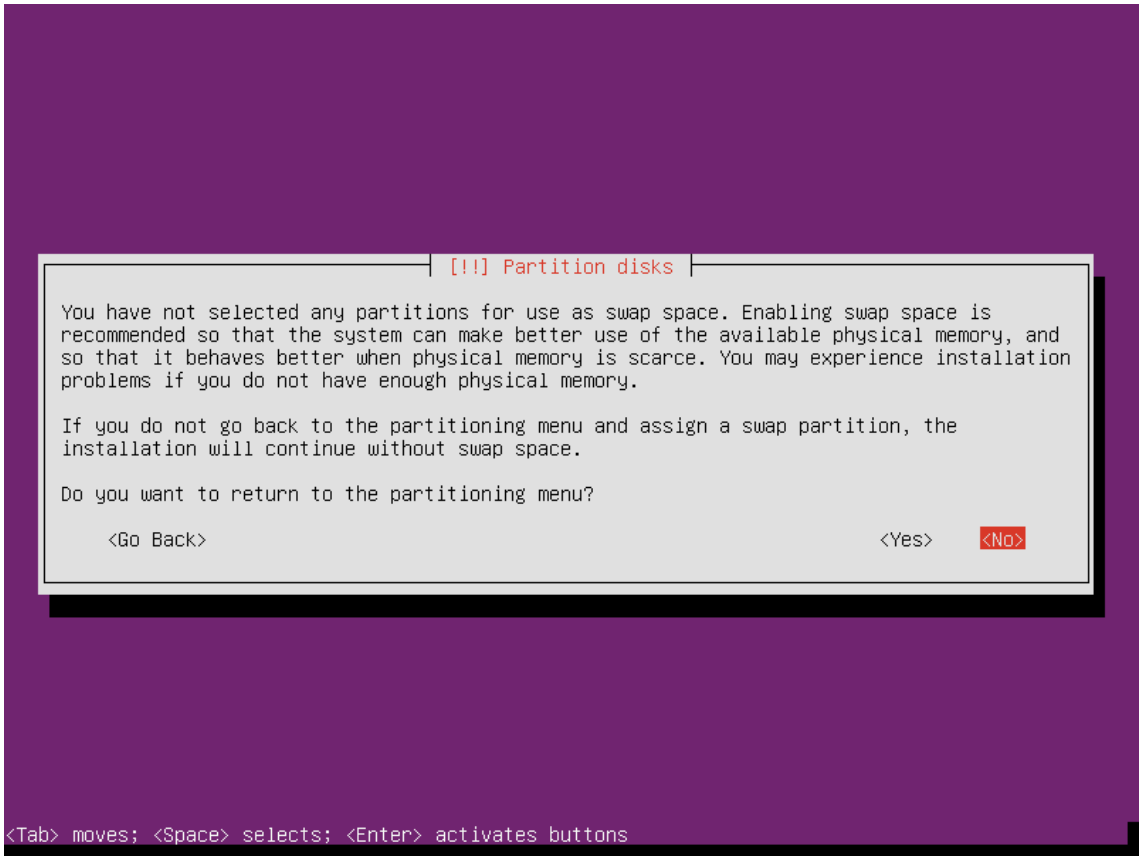


Figura 70. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 31

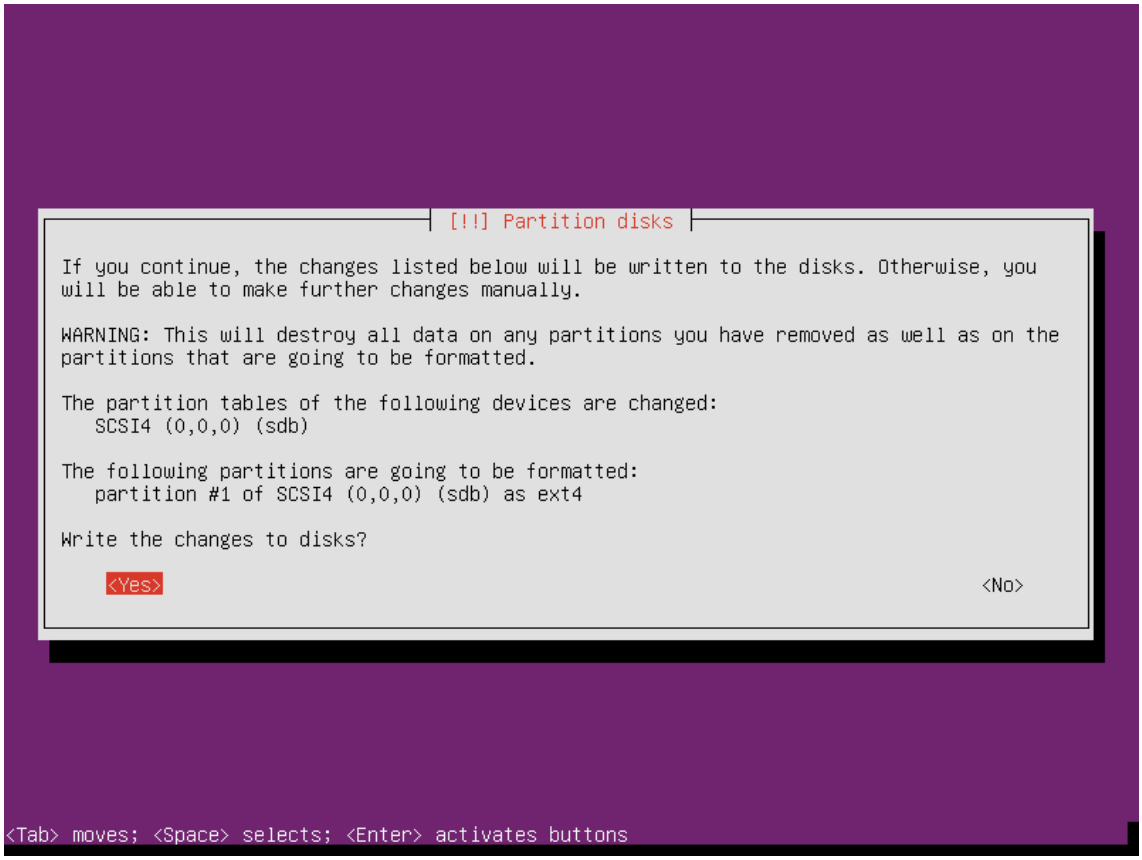


Figura 71. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 32

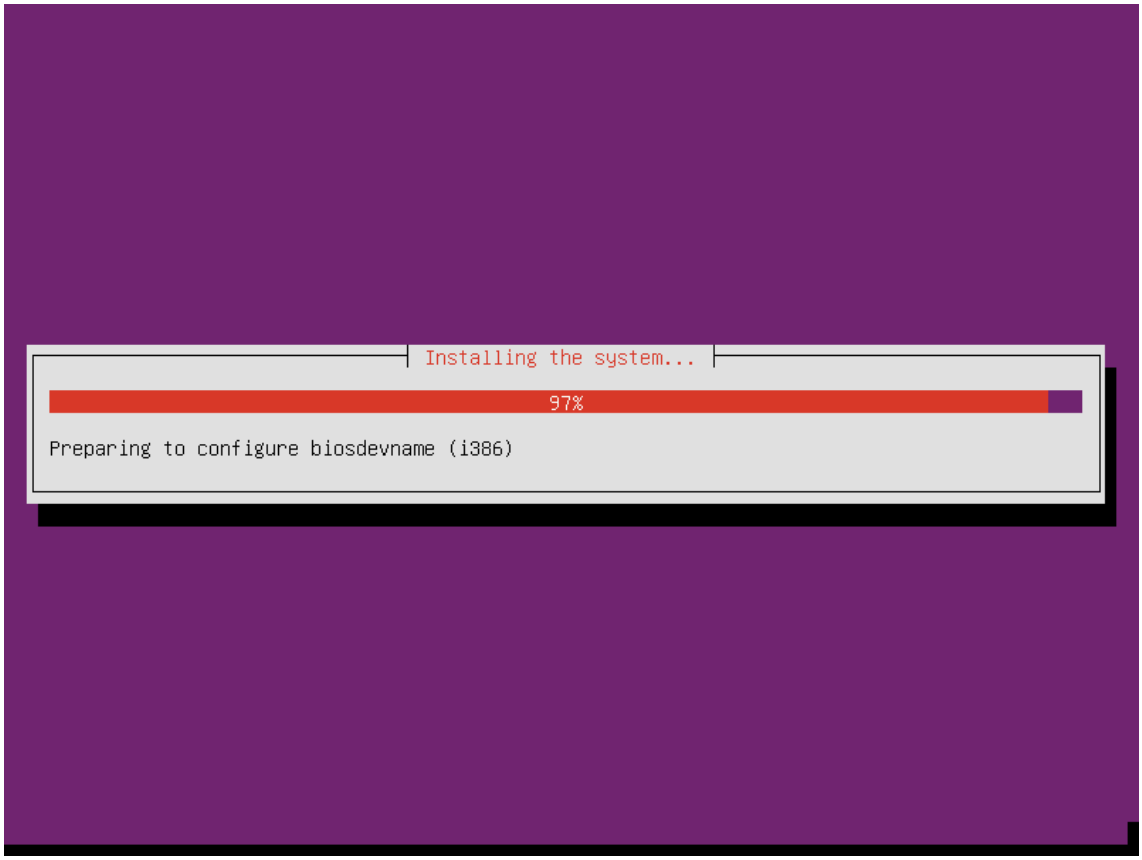


Figura 72. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 33

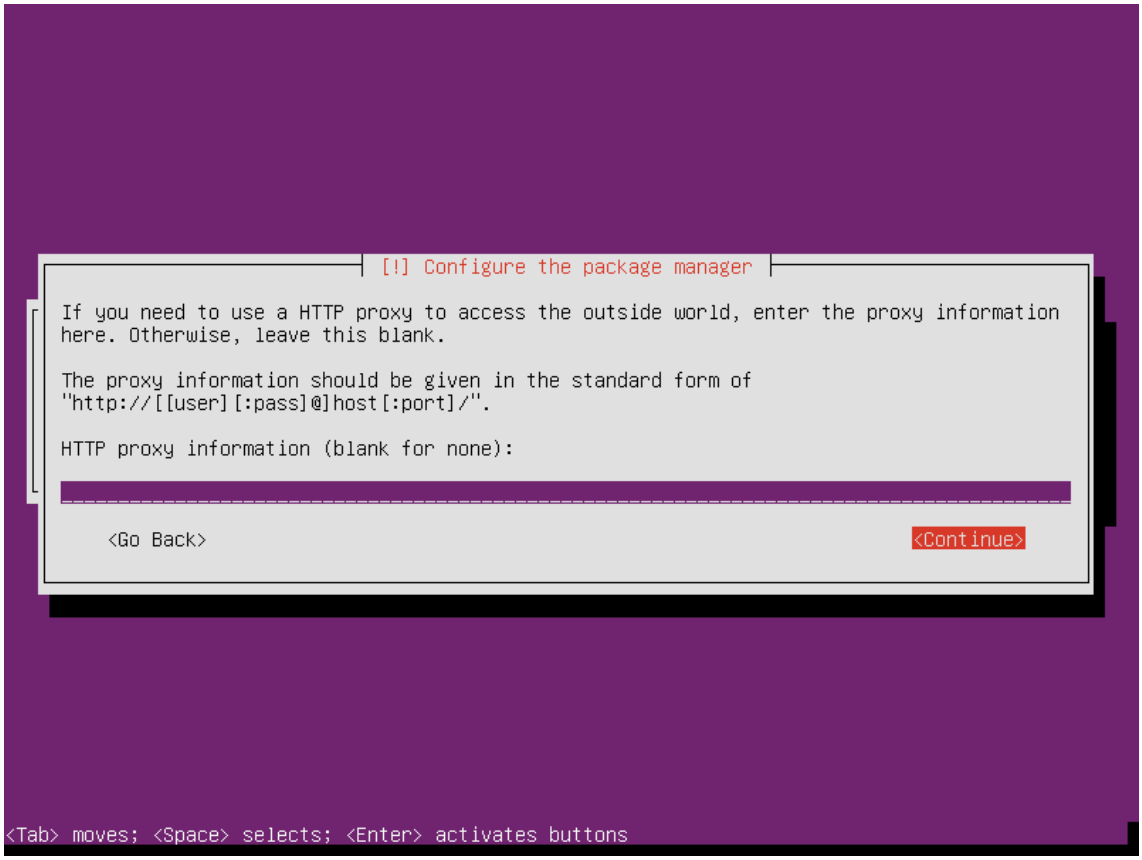


Figura 73. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 34

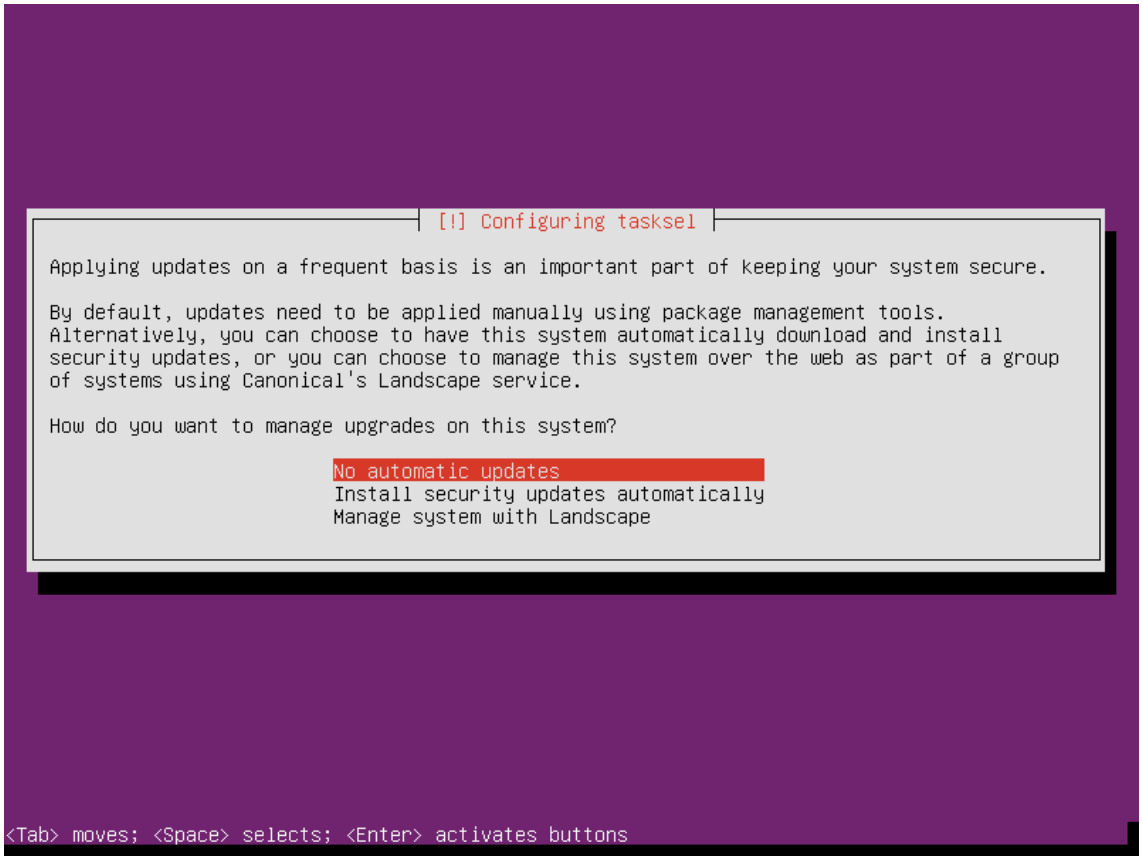


Figura 74. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 35

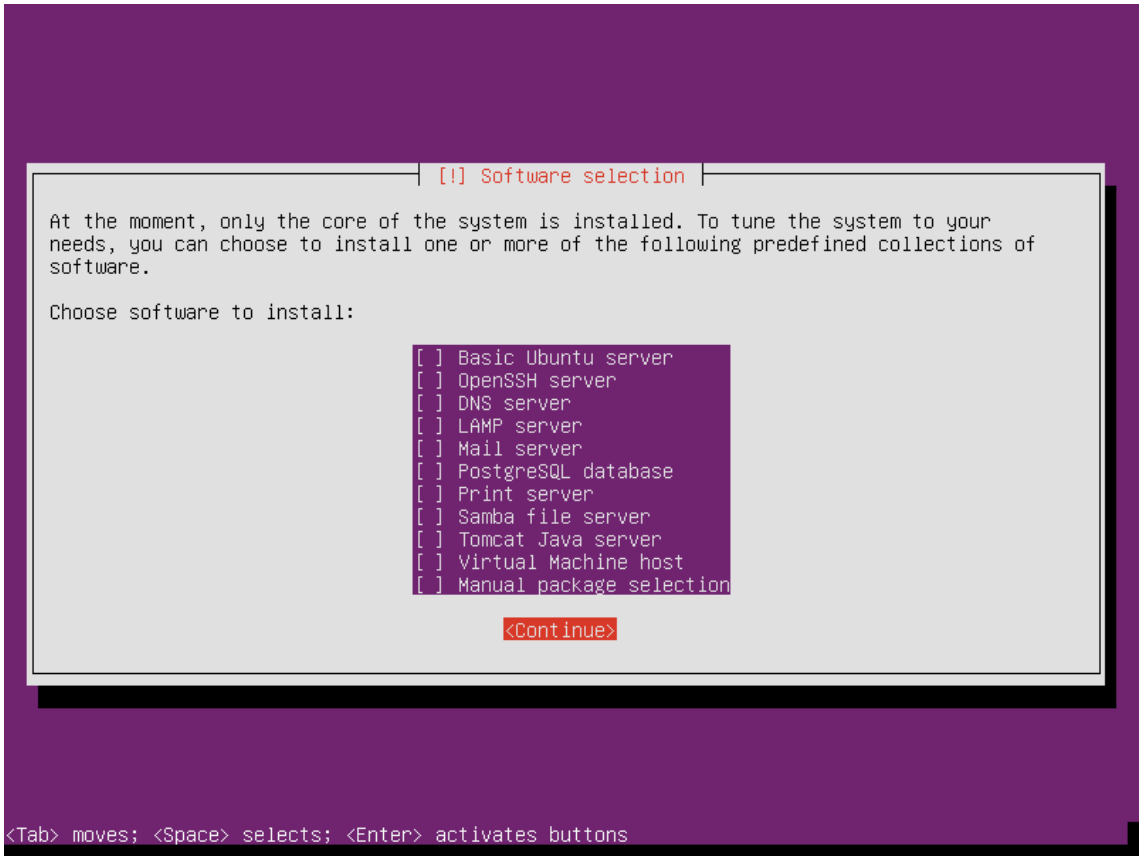


Figura 75. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 36

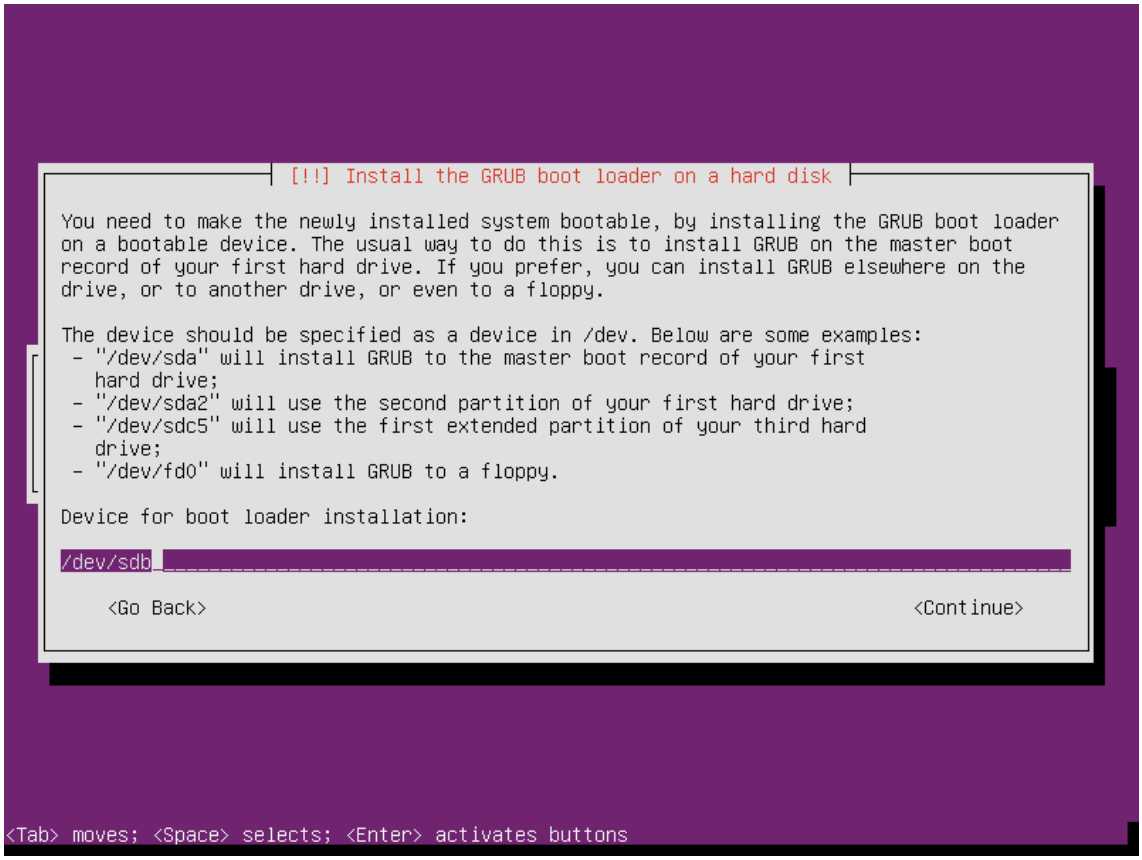


Figura 76. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 37

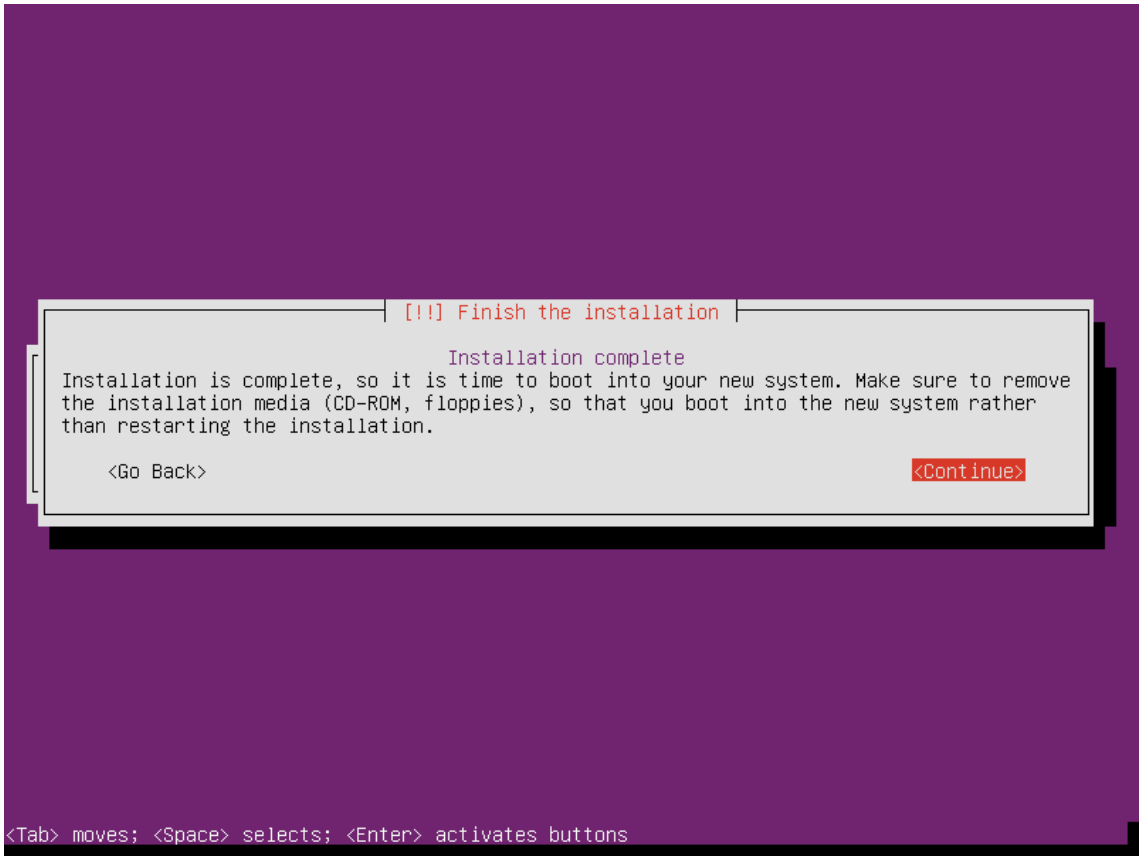


Figura 77. Instalación de Ubuntu Server 14.04.4 sobre USB 1: paso 38

Después de esto, abrimos una terminal (Linux) y ejecutamos con el dispositivo USB 1 conectado:

```
sudo -s
fdisk -l
mkdir /media/usb1
mount /dev/sdb1 /media/usb1
apt-get update
cd /var/cache/apt/archives
rm *.deb
apt-get install --download-only pv
cp *.deb /media/usb1/usr/src
umount /media/usb1
```

Arrancamos desde el dispositivo USB 1.

Instalamos el paquete pv:

```
sudo -s
cd /usr/src
dpkg –i *.deb
```

Creamos un programa *script* de nombre *booting.sh*.

```
nano booting.sh
```

- *booting.sh*

```
#!/bin/sh
clear
cd /usr/src/
echo Installing Xenomai kernel to Compact Flash /dev/sda
dd if=sDIAG_21032016_xenomai_configured_with_ro_mode.iso |pv|dd
of=/dev/sda bs=1M
```

Figura 78. Programa *script* para añadir al arranque del dispositivo USB

Damos permiso de ejecución al programa *script booting.sh*:

```
chmod +x booting.sh
```

Hasta aquí lo tenemos ya en el siguiente fichero `booting_15062016_cleaned_with_pv`

Este fichero lo podemos descargar desde esta URL de OneDrive:

https://1drv.ms/u/s!AhkSKZxSDxNheK71cpYn4kZMiq4

Podemos grabarlo en un dispositivo USB (por ejemplo, en `/dev/sdb`).

```
fdisk -l
dd if=booting_15062016_cleaned_with_pv |pv|dd of=/dev/sdb bs=1M
```

Encendemos la placa de evaluación sDIAG con el dispositivo USB.

Copiamos el script `booting.sh` a la carpeta `/etc/init.d/`

```
cp booting.sh /etc/init.d/
```

Lo convertimos en *daemon*, es decir que se ejecute al inicio, a partir de la próxima sesión.

```
update-rc.d booting.sh defaults
```

Para desactivarlo:

```
update-rc.d -f booting.sh remove
```

Copiamos la imagen a grabar (**`/home/real-time/Escritorio/sDIAG_21032016_xenomai_configured_with_ro_mode.iso`**) a la Compact Flash al dispositivo USB.

```
rsync –-progress /home/real-time/Escritorio/sDIAG_21032016_xenomai_configured_with_ro_mode.iso
/media/usb1/usr/src
```

E. Anexo: Información adicional

En repositorio OneDrive (servidor remoto)

- `solutions.zip`: Kernel precompilados de soluciones para Linux en tiempo-real generados durante este proyecto.
  Disponible en: https://1drv.ms/u/s!AhkSKZxSDxNhenjMtI1bk56KaOw
- Mejoras:
  - `caf_pa.zip`: Tema `caf_pa` creado para el servicio de plymouth de Linux.
    Disponible en: https://1drv.ms/u/s!AhkSKZxSDxNhexV0C2Mlp1R4UDs

El resto de ficheros generados y usados son confidenciales de la empresa y no se han dejado disponibles una vez presentado el proyecto.

Agradecimientos

A Oihana Azpitarte (ingeniera de software empotrado) por ser mi responsable directo en CAF Power & Automation. A Juan Fernández Bustamante (gestor de producto TCMS) de la misma empresa por asignarme este proyecto.

A buena parte de la comunidad científica de sistemas operativos de tiempo-real por enseñarme desde lo que son los *spinlocks*, las interrupciones multihilo, o el API nativa de Xenomai, etc. Y en especial al Dr. Claudio Scordino, antiguo trabajador de Evidence (Italia), Dr. Xavier Martorell, de la *Universitat Politècnica de Catalunya*, Felipe Cerqueira, de *Max-Planck-Institut für Softwaresysteme* (Alemania) o Dr. Alberto Lafuente, de la Universidad del País Vasco/*Euskal Herriko Unibertsitatea*.