https://doi.org/10.23850/2422068X.1177

Diseño de un kernel de tiempo real para el control de una impresora 3D[1]

Design of a real-time kernel for the control of a 3D printer

Marco León Mora Méndez[2]

RESUMEN
Este documento describe el proceso de diseño del software de un gestor de tareas adecuado para controlar una impresora 3D, e incluye el análisis de tiempo de cada subproceso, el control de ejecución (en intervalos de tiempo fijos y determinados) y la atención a comunicaciones —con características no determinísticas— entre el dispositivo a controlar y la interfaz humana, residente en un computador de escritorio.

Palabras clave: Algoritmos de planificación, impresión 3D, núcleo, simulación, Sistemas de tiempo real.

Recibido: 13/07/2017 Aceptado: 25/10/2017.

ABSTRACT
This document describes the process of designing the software of an addecuate task manager to control a 3D printer, and includes the time analysis of each subprocess, the execution control (in fixed and determined time intervals) and the attention to communications with non-deterministic characteristics- between the device to be controlled and the human interface, that resides in a desktop computer.

Keywords: Planning algorithms, 3D print, core, simulation, real-time systems.

[1] Artículo de investigación derivado del proyecto de investigación Diseño y Fabricación de una Impresora 3D por extrusión de termoplástico, financiado por SENNOVA, 2015.

[2] Ingeniero de Sistemas, especialista en Teleinformática. Centro de Industria y la Construcción, SENA Regional Tolima. Ibagué, Colombia. Correo electrónico: mlmoram@misena.edu.co

INTRODUCCIÓN

Se presenta el proceso de diseño de un gestor de tareas *kernel* adaptado a las necesidades de control de una impresora 3D (Torres, León & Torres, 2012); que es un tipo de dispositivo CNC (Control Numérico Computarizado) desarrollado en los últimos años, adecuado a un mercado muy amplio y versátil y de gran utilidad en el campo del aprendizaje.

Una alternativa para realizar la implementación de la gestión de tareas en sistemas de tiempo real periódicos lo constituye la utilización de planificadores cíclicos. Este método goza de un alto grado de determinismo, previsibilidad, fiabilidad y sencillez de implementación (Guevara, Valdez & Delgado, 2014). Es una técnica bien conocida y ampliamente utilizada en entornos industriales, aunque su principal inconveniente es la falta de flexibilidad, ya que cualquier cambio en el conjunto de tareas o en sus características temporales obliga a rehacer el plan de ejecución.

A partir de una revisión de la teoría de Sistemas Operativos de Tiempo Real (RTOS) y de la definición de los requerimientos funcionales de los mecanismos a controlar, se definieron las características de los objetos de software utilizados para cada elemento de control, que incluyen análisis de tiempos y del comportamiento de los algoritmos respectivos, hasta finalizar en la selección de ciertas características del planificador, que cumplieran con las condiciones necesarias para implementar un control embebido, basado en microcontrolador (μC) (Camargo & Andrade, 2013). El diseño general del sistema sigue la metodología Top-Down, que va de lo general a lo particular o específico, permitiendo un desarrollo incremental de las funcionalidades requeridas.

METODOLOGÍA

El diseño general sigue la metodología *Top-Down*, que facilita un desarrollo incremental, desde lo general hasta los detalles específicos.

Inicialmente, se define un diagrama de bloques general (modelo estático), para comprender la interrelación entre los módulos del software; posteriormente, se realiza el diagrama de estados (modelo dinámico); finalmente, se define con mayor detalle cada subsistema, se realizan diagramas de flujo de datos para determinar las variables a utilizar y, por último, se diseñan los algoritmos respectivos.

A partir de la revisión de los conceptos teóricos asociados a Sistemas Operativos en Tiempo Real y una vez definida las características funcionales de la impresora, se determinó usar el algoritmo de *Round-Robin*, para ejecutar los módulos software de manera equitativa (Guevara *et al.,* 2014; Uzcátegui, Dinarle & Delgado, 2009).

Para analizar las exigencias de tiempos de respuesta y determinar los valores críticos a considerar en el diseño del planificador, se definieron los componentes software, teniendo especial cuidado en los tiempos totales máximos de ejecución de cada módulo.

DESCRIPCIÓN DE LOS SISTEMAS OPERATIVOS DE TIEMPO REAL (RTOS)

Los Sistemas Operativos (S.O.) son módulos de software encargados de administrar la ejecución

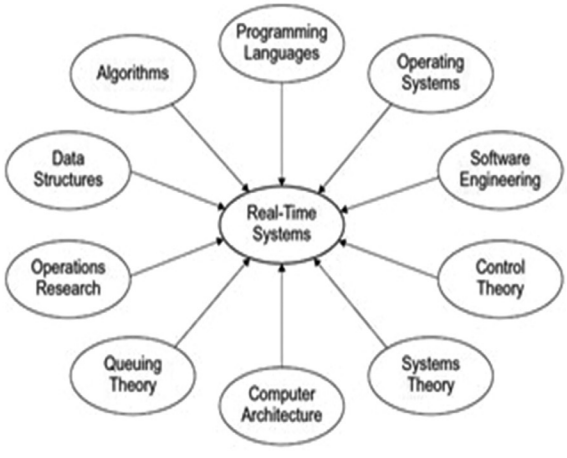
de diferentes tareas y de los recursos de un sistema computador (Crespo & Alonso, 2006). Existen dos tipos básicos; interactivos; utilizados en interfaces de usuario, con características no determinísticas y de tiempo real, que en general son determinísticos y en los que se deben considerar estrictas restricciones de tiempo (Aldea Rivas, 2002; Yépez, 2007).

El término *tiempo real* es utilizado en diversos contextos, tanto técnicos como convencionales. Para muchos significa *al tiempo* o *instantáneamente*. En este contexto se entenderá como el referido a aplicaciones en las que el computador debe responder con la rapidez requerida por el usuario o por el proceso que se controla (Drake, *et al.,* 2014; Laplante & Ovaska, 2012).

El estudio de los sistemas en tiempo real es una verdadera disciplina multidimensional de la ingeniería de sistemas que está fuertemente influida por la teoría del control, la investigación operativa y la ingeniería de software. La Figura 1 muestra algunas de las disciplinas que afectan al diseño y análisis de sistemas en tiempo real (Laplante & Ovaska, 2012). Debido a esto, se presentan múltiples desafíos de diseño. Aunque los fundamentos de los sistemas en tiempo real están bien establecidos desde prácticamente el inicio de la computación, los sistemas en tiempo real son un área de desarrollo muy activa, debido a la evolución de las arquitecturas de CPU, las estructuras de sistemas distribuidos, las redes inalámbricas y el advenimiento de nuevas tecnologías informáticas y de comunicaciones.

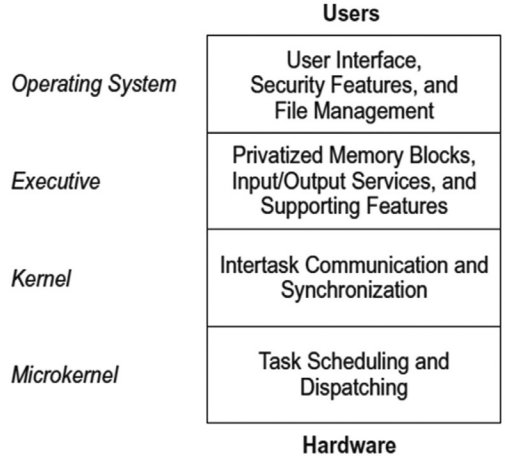
Los diferentes sistemas de control en tiempo real se pueden clasificar como se ejemplifica en la Figura 2, al considerar la cercanía relativa al usuario o a la máquina. Esta clasificación otorga los primeros elementos a considerar en el posterior proceso de diseño.

Figura 1. Variedad de disciplinas que afectan el diseño de Sistemas en Tiempo Real



Fuente: (Laplante & Ovaska, 2012)

Figura 2. Taxonomía de los niveles de complejidad de los RTOS



Fuente: (Laplante & Ovaska, 2012)

Los algoritmos que implementan estos sistemas de control se clasifican en estáticos y dinámicos. En los estáticos todas las decisiones de control se toman a priori, construyendo una tabla que indica los tiempos de inicio y finalización, asociados a cada tarea, que permiten controlar que ninguna pierda su tiempo límite de ejecución y, presentan como resultado, un enfoque altamente predecible. En este caso, si algún parámetro cambia, es necesario recalcular la tabla y reiniciar el sistema (Guevara, *et al.,* 2014).

Un S.O. se compone fundamentalmente de un Planificador y un Despachador. El Planificador puede ser Cooperativo, en el que las diferentes tareas ceden el control una a una, o Expropiativo, controlado por interrupciones; en el que un núcleo —kernel— asigna las tareas a ejecutar. El Despachador puede usar un mecanismo de turno rotativo (Round-Robin) o por prioridad, con estrategias primero en llegar, primero en salir, (FIFO); primero en llegar, último en salir (LIFO), etc.

Los métodos de planificación (planificadores) en tiempo real se clasifican en: Basados en reloj (por tiempo), en Round-Robin con prioridades, Basados en prioridades, Basados en la reserva del tiempo, Heurísticos y Flexibles (Guevara, *et al.,* 2014).

Para determinar el método a utilizar, es fundamental entender las características del sistema desde el punto de vista del nivel de aleatoriedad de las mismas. Un sistema es determinístico si para cada posible estado y para cada grupo de entradas, puede ser determinado un solo conjunto de salidas y conocido el siguiente estado del sistema (Laplante & Ovaska, 2012). Cuando no se conocen con precisión estos valores, se dice que el sistema es no determinístico.

En la clasificación anterior se han resaltado las que más se adaptan a los requerimientos de sistemas embebidos (o empotrados), diseñados para realizar una o algunas pocas funciones dedicadas, frecuentemente en un sistema de computación en tiempo real.

La planificación de tareas concurrentes puede ser realizada antes de iniciar la ejecución (off-line) o por un mecanismo de prioridades en tiempo de ejecución (on-line), como se muestra en la tabla 1.

Conceptos asociados al diseño de un RTOS

Se presenta, a continuación, una relación de elementos conceptuales útiles para el diseño propuesto (Pastor, 2005; Aldea Rivas, 2002), y se incluye una breve descripción de cada uno.

Planificación de la ejecución de las tareas. Dado que el objetivo es ejecutar varios procesos a la vez y que esto es imposible, ya que el μC solo puede realizar un proceso al mismo tiempo, se debe repartir el tiempo del μC entre las diferentes tareas concurrentes. De acuerdo con las características del sistema a implementar, se usará el algoritmo de *Round-Robin*, que es un método útil para desarrollar todos los elementos en un grupo de manera equitativa y en un orden racional. Normalmente se comienza por el primer elemento de la lista hasta llegar al último y se vuelve a hacer lo mismo (Morales, 2014; Yépez, 2007).
Se trata de ejecutar las tareas sucesivamente turnadas por un tiempo o quantum. Mientras

Tabla 1. Clasificación general de los planificadores para tareas en tiempo real concurrentes

Planificación	**1. Planificación off-line (ejecución cíclica)**		
	2. Planificación on-line (Por prioridades)	Prioridades estáticas	Rate Monotonic
			Deadline Monotonic
		Prioridades dinámicas	Earliest deadline First
			Least Laxity First
			Shortest Slack Time First

Fuente: (Guevara, *et al.,* 2014). Adaptado

más pequeño sea el quantum en cuestión, más simultánea parecerá la ejecución; y mientras más procesos se produzcan a la vez, menor será la velocidad con que se origine cada proceso y más lento será el sistema (Díaz, Ramírez, Mejía Álvarez & Leyva del Foyo, 2013).

Planificador Ejecutivo Cíclico. Es un procedimiento iterativo que permite planificar la ejecución de un conjunto de procesos periódicos en un procesador de forma determinista, tal que los tiempos de ejecución sean predecibles (Alfonsi, 2012). Los ejecutivos cíclicos o planificadores cíclicos ejecutan las tareas en secuencia según unas tablas de planificación o planes de ejecución, su construcción depende de los requisitos temporales y del conjunto de tareas. El plan define la secuencia de actividades que deben ejecutarse durante un período fijo de tiempo llamado ciclo principal o híperperiodo. Este plan principal se divide en uno o más planes secundarios, los cuales describen la secuencia de tareas que deben ejecutarse durante un período fijo de tiempo llamado ciclo secundario (Alfonsi, 2012).

Bloque de Control de Proceso (PCB). Es un área de memoria donde se agrupa toda la información necesaria para un proceso particular. Cada vez que se crea un proceso, el sistema operativo crea el BCP correspondiente, para que sirva como descripción en tiempo de ejecución durante su duración (Morales, 2014).

Sección Crítica. Los sistemas multitarea suelen estar relacionados con el uso compartido de recursos. En la mayoría de los casos, solo pueden ser utilizados por una sola tarea a la vez y su proceso no puede ser interrumpido. Se dice que dichos elementos son reutilizables, e incluyen periféricos, memoria compartida y también la Unidad Central de Procesamiento (CPU). Cada segmento de código que interactúa con estos medios compartidos se denomina *sección crítica.* Si dos tareas entran simultáneamente en la misma sección crítica, pueden producirse errores catastróficos (Laplante & Ovaska, 2012) (dos o más tareas que intentan acceder al mismo recurso simultáneamente).

Programación Concurrente. Cuando dos o más tareas se ejecutan simultáneamente. También se denomina Programación Multihilo —*Multithreading*—. Un lenguaje de programación concurrente permite expresar una colección de procesos secuenciales autónomos, que se ejecutan, lógicamente, en paralelo. La ejecución de una colección de procesos, normalmente

se puede realizar de tres formas distintas. De acuerdo con Pastor (2005):

- Con ejecución multiplexada en un único procesador.
- Con ejecución multiplexada en un sistema multiprocesador con acceso a memoria compartida.
- Con ejecución multiplexada en varios procesadores que carecen de memoria compartida. Este es el caso de los sistemas distribuidos.
- *Mecanismos de comunicación y sincronización*. De acuerdo con Pastor (2015), es útil distinguir, con relación a la comunicación y sincronización entre los procesos, tres tipos de comportamiento:
- Independiente: Los procesos no se comunican o sincronizan entre ellos y se ejecutan de forma independiente.
- Cooperativo: Se comunican o sincronizan entre ellos regularmente y realizan conjuntamente una determinada operación.
- Competitivo: Varios procesos utilizan unos recursos del sistema que son limitados y, por tanto, la utilización por parte de unos procesos provoca la espera de otros que estén libres.

El tipo de relación entre los procesos es importante al momento de estudiar el comportamiento temporal del sistema y asegurar que se van a cumplir las restricciones temporales.

Los conceptos de comunicación y sincronización están relacionados. La comunicación es el paso de información de un proceso a otro y la sincronización es una forma de comunicación entre los procesos.

La comunicación de datos, usualmente está basada en dos mecanismos, las variables compartidas y el paso de mensajes. Las primeras son objetos a los que puede acceder más de un proceso. El paso de mensajes es un mecanismo de comunicación que se lleva a cabo depositando información de un determinado proceso, que será leída por otro proceso.

Semáforos y Banderas. El paso de mensajes se refiere al intercambio de datos que se envían de alguna manera entre dos procesos; y una forma de evitar el acceso simultáneo al mensaje (intentar leerlo mientras no se ha terminado de escribir; por ejemplo), es usando registros o áreas de memoria especiales que señalan la disponibilidad de dicho recurso y que se denominan semáforos (también se pueden usar para otro tipo de controles, por ejemplo, señalar el estado de una tarea). Cuando la señalización es suficiente con un bit (SÍ/NO) cada bit se denomina bandera (*flag*).

Procesos y Objetos. La programación orientada a Objetos ofrece una visión del sistema como una colección de objetos que colaboran entre ellos (Pastor, 2005). Este paradigma permite enfocar el diseño hacia la modularidad, el encapsulamiento (independencia entre objetos) y la reutilización de recursos de software, que facilitan la implementación del sistema global.

Interrupciones. Cuando ocurre un suceso externo, un dispositivo hardware lo detecta y genera una señal al procesador, que provoca una solicitud de interrupción. Es importante el tiempo que tarda el sistema en reaccionar a la interrupción, es decir, el período que transcurre desde que esta se solicita hasta que se activa su manejador (Pastor, 2005).

Existen tres factores importantes que influyen en el tiempo de respuesta a las interrupciones:

1) Niveles de interrupción. La disponibilidad de distintos niveles de interrupción, es decir, la capacidad de que una solicitud de nivel superior detenga a un manejador de interrupción de nivel inferior, permite organizar estos mecanismos en jerarquía, de forma que una solicitud de atención importante no pueda ser bloqueada por otra de menos importancia, aunque esta segunda precise de un manejador de larga duración.

2) Duración del tratamiento de las interrupciones. Es decir, el tiempo máximo que el sistema está tratando una interrupción. Durante su desarrollo, se inhibe la atención de otras interrupciones para evitar inconsistencias en el manejo del hardware asociado a la primera. La duración máxima determinará la máxima frecuencia a la que pueden atenderse estos mecanismos.

3) Tiempo de latencia de interrupción. Es el tiempo que el sistema inhibe las interrupciones en zonas críticas en que no puede ser detenido. Los tiempos de latencia de este mecanismo, provocarán retrasos en la activación de los manejadores, produciendo, no solo el retraso de la respuesta, sino la pérdida de interrupciones.

Contexto de una tarea. Hace referencia al valor de todos los registros del procesador relacionados con un proceso en ejecución, en el instante en que inicia una interrupción. Para que al terminarla, la tarea detenida pueda continuar sin sufrir alteraciones en su estado, es necesario almacenar el valor actual de los registros (salvar el contexto) en una zona de memoria que no sea alterada por el programa manejador de la interrupción y, al final, recuperar sus valores (cargar el contexto) y continuar con el proceso inicial.

Parámetros para describir las tareas de tiempo real

Un planificador cíclico ejecuta las tareas en secuencia según un plan de ejecución que se construye antes de poner en marcha el sistema (Uzcátegui, Dinarle & Delgado, 2009). Para construir el plan se debe determinar la duración de los períodos de los ciclos (ciclo principal, ciclos secundarios) y se asignan las ejecuciones de tareas hasta obtener una planificación que asegure el cumplimiento de sus restricciones temporales (Yépez, *et al.,* 2005). Para el efecto, se considerarán los siguientes parámetros, que son relevantes en el diseño propuesto (Yépez, *et al.,* 2005; Yépez, 2007):

- *Período T:* Es el tiempo entre cada inicio de ejecución de todas las tareas.
- *Plazo de Ejecución Di:* Es el máximo tiempo permitido para una nueva ejecución de la tarea.
- *Tiempo de Computo Ci:* Es el tiempo medio de ejecución de la tarea i.
- *Tiempo de Cómputo Máximo Cimax:* Es el tiempo de ejecución máximo de la tarea i.
- *Tiempo de Cómputo Mínimo Cimin:* Es el tiempo de ejecución mínimo de la tarea i.
- *Ciclo principal M:* Es la duración del ciclo principal (híperperiodo). La duración del ciclo principal debe cumplir ciertas condiciones:

Para que sea un conjunto de *n* tareas periódicas t_i: $\{(C_i, T_i, D_i)\}$, se debe cumplir con lo siguiente:

$$C_i \leq T_i \leq D_i \qquad (1)$$

Para que todas las tareas se ejecuten dentro de un ciclo M, se debe cumplir con lo siguiente:

$$\sum C_i \leq M \qquad (2)$$

En la escogencia del período T y para asegurar que, desde una perspectiva del hardware, todos los actuadores reciban atención a la frecuencia mínima necesaria, logrando el efecto de concurrencia, es necesario que:

$$\min (D_i) \geq T \qquad (3)$$

Se debe tener en cuenta que el plazo para una tarea periódica P_i, es un factor de diseño crítico que está limitado por:

$$C_{imax} \leq P_i \qquad (4)$$

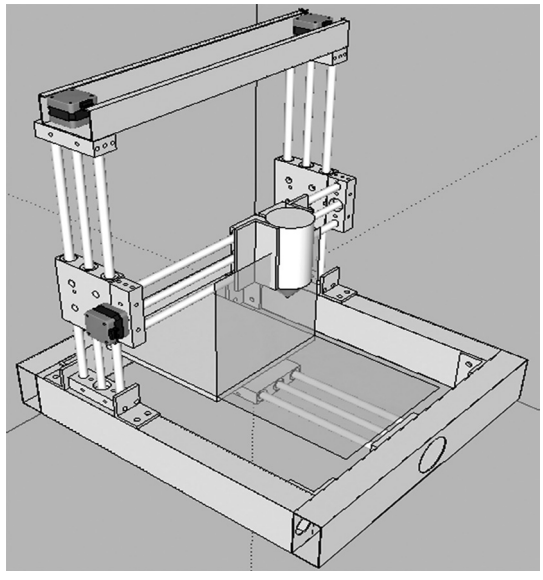
Estas pruebas confirman las condiciones matemáticas para verificar si el grupo de tareas cumple con las restricciones de tiempo del algoritmo de planificación seleccionado (Díaz-Ramírez, Mejía Álvarez & Leyva del Foyo, 2013; Yépez, Guardia, Velasco, Ayza, Marti & Fuertes, 2005). Si no existe ningún valor que satisfaga el conjunto de condiciones anteriores, entonces el conjunto de tareas no puede ser planificado en un planificador cíclico.

REQUERIMIENTOS FUNCIONALES DE LA IMPRESORA 3D

En la Figura 3 se presenta la estructura mecánica de la impresora 3D diseñada. Está compuesta por tres ejes (X, Y, Z) accionados por sendos motores paso a paso (PaP) que permiten el desplazamiento de un mecanismo extrusor con calentador y un motor PaP para aplicar un hilo de plástico fundido que formará la figura tridimensional. La base sobre la que se construye la figura se denomina *cama* y debe mantener una temperatura media para evitar distorsiones de la figura 3D al enfriarse parcialmente.

El sistema a diseñar controlará cuatro motores (MX, MY, MZ, ME); dos resistencias calefactoras (RE, RC); leerá sus temperaturas (TE, TC); vigilará constantemente seis (6) límites de carrera para evitar que los motores intenten moverse más allá de los límites mecánicos de desplazamiento (límite izquierdo y derecho para cada eje).

Figura 3. Estructura de la extrusora 3D



Fuente: El Autor

Para el control de la impresora se requiere un sistema que interprete los datos enviados desde el PC e interactúe con el subsistema electromecánico (actuadores y sensores) en forma precisa y en tiempo real. Dicho sistema debe estar embebido en el micro-controlador (μC) y debe interpretar

los comandos recibidos, administrando y controlando los diferentes actuadores y sensores y transmitiendo el estado de las variables de control, por medio de una comunicación permanente, de doble vía con el PC.

Los requerimientos generales se pueden subdividir en los presentados a continuación:

- Recibir cada comando enviado desde el PC y almacenarlo en una cola para su posterior atención.
- Interpretar cada comando y ejecutarlo.
- Leer las temperaturas y finales de carrera y transmitirlos al PC.

Análisis inicial del sistema embebido

El RTOS requerido exige la asignación periódica de tareas y la comunicación y sincronización entre ellas, razón por la cual, taxonómicamente, es de tipo *kernel* (Laplante & Ovaska, 2012).

Diagrama E-P-S del sistema de control

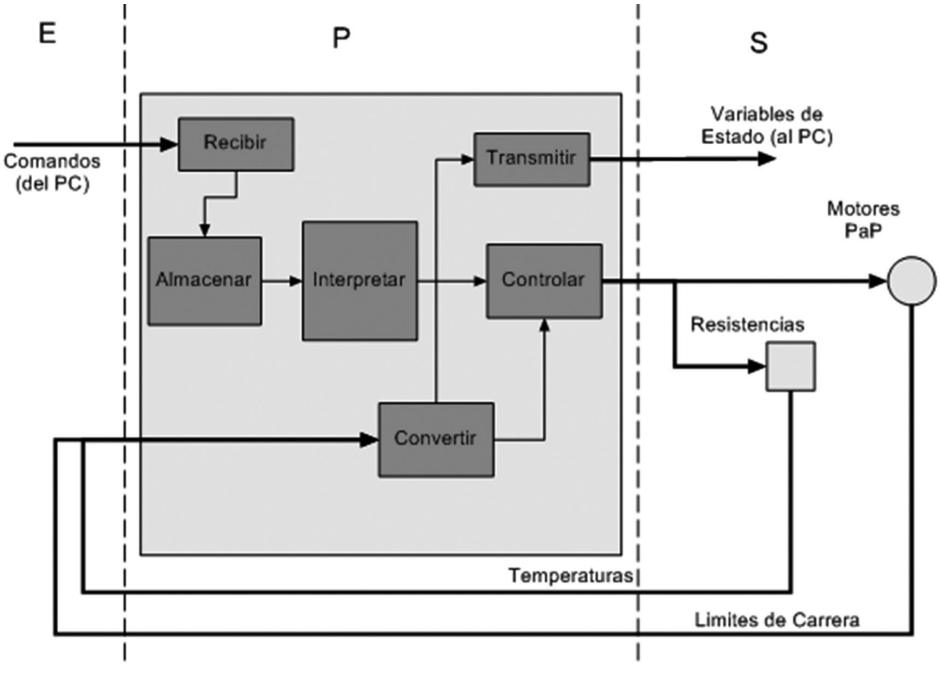
En la Figura 4 se realiza una primera aproximación al sistema, con un diagrama Entrada – Proceso – Salida (E-P-S).

Las entradas iniciales son los comandos recibidos desde el PC, que requieren un módulo encargado de su recepción y almacenamiento. Otras entradas son los valores de temperaturas (TE, TC) y los estados de los finales de carrera, recibidos por un módulo que se encarga de su conversión análogo a digital (ADC) para las temperaturas y lectura de los límites de carrera (On-Off).

El proceso principal consiste en interpretar cada comando recibido y transmitirlo al módulo de control del actuador respectivo (motor PaP o resistencia de calentamiento).

Como salidas se definen las señales eléctricas a los motores (un pulso por paso y dirección) y las

Figura 4. Diagrama de *caja gris* del sistema de control



Fuente: El Autor

señales eléctricas para las resistencias utilizando la técnica de modulación por ancho de pulso (PWM) para convertir señales digitales que salen del µC en valores análogos a las resistencias de calentamiento. También se considera salida del sistema la transmisión de las variables de estado (temperaturas y límites de carrera) al PC.

Módulos de software

Al considerar los planteamientos anteriores, seguidamente se define el diagrama de bloques que permite comprender la interrelación entre los módulos del software (ver Figura 5). En los extremos, los elementos hardware de interface: A la izquierda la conexión al PC por un puerto USB; a la derecha los puertos de salida del µC para los actuadores (motores y resistencias) y de entrada (temperaturas y límites).

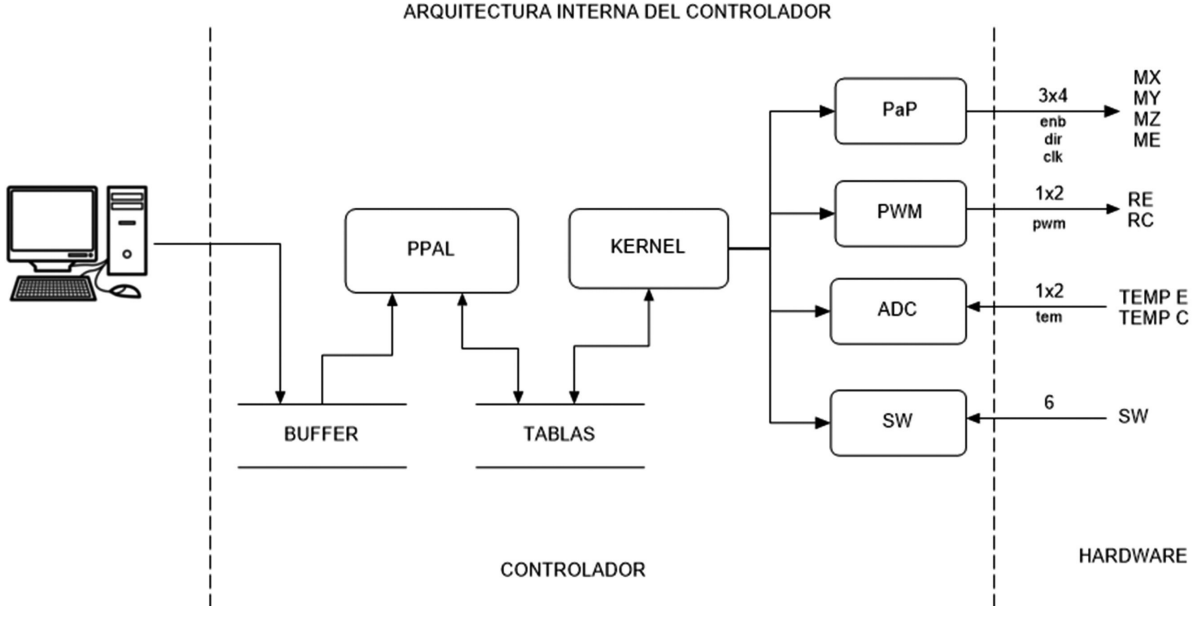
Se define un módulo de comunicaciones que, al recibir un comando del PC, lo almacenará en un *buffer* (memoria) circular y le transmitirá las variables de estado enviadas desde el control principal. Este se encarga de interpretar los comandos recibidos y de comunicar los parámetros a cada tarea.

Las tareas se administran desde un *kernel* (planificador) disparado por interrupciones periódicas para realizar la ejecución concurrente de los módulos PaP, PWM, etc. Es de resaltar que para cada uno de los cuatro motores se utilizará el mismo módulo de software, pero con sus parámetros particulares; lo mismo para los controles PWM de los calentadores y el conversor ADC de las temperaturas.

Diagrama general de estados

El módulo que inicia y que estará en ejecución permanente es el principal, encargado de las tareas de leer e interpretar los comandos recibidos y transmitir las variables de estado: Inicia en espera de un byte desde el PC, que interpreta para conocer si es un comando a un motor, a una resistencia o una petición del estado. Si no se interpreta correctamente el

Figura 5. Diagrama de bloques de los módulos del sistema



Fuente: El Autor

comando, se transmite un mensaje de error. Por último, se comunica el nivel de ocupación del buffer, para repetir todo de nuevo (no existe un punto de parada para este módulo).

El *kernel*, que contiene el planificador cíclico, se ejecutará con un período fijo, activado por una interrupción hardware con prioridad de primer nivel (bloquea cualquier otra interrupción), que detendrá al módulo principal mientras atiende todas las tareas definidas.

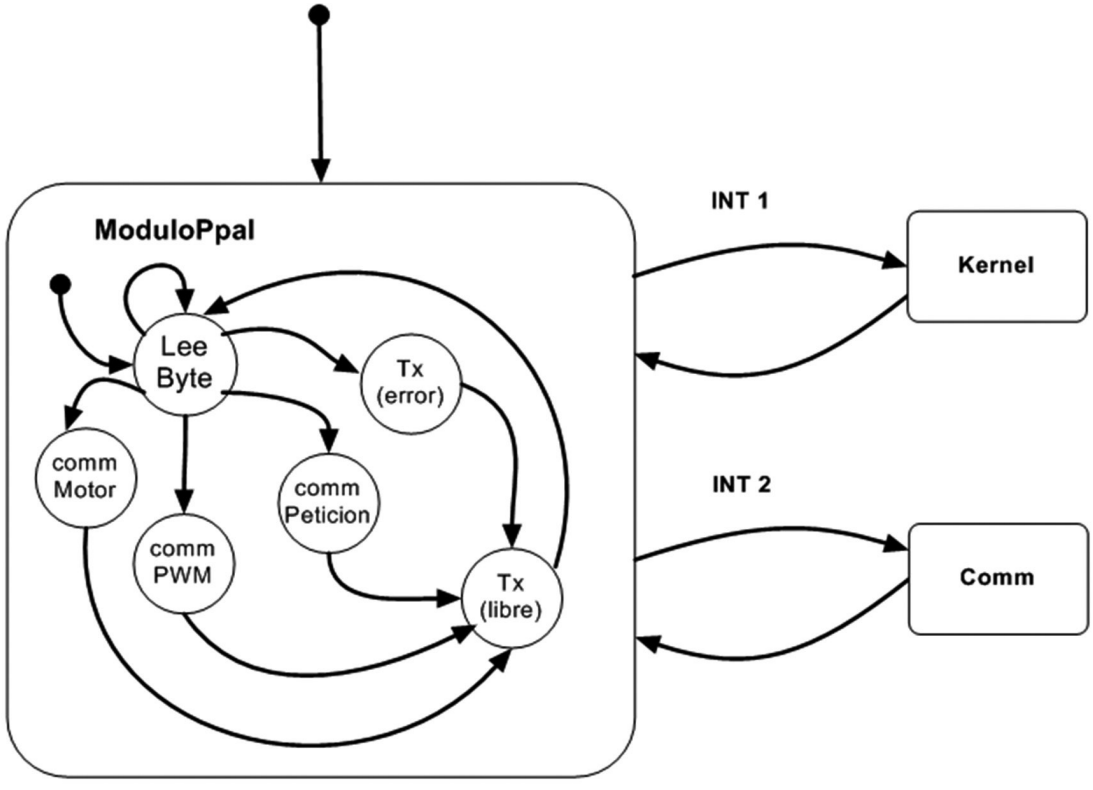
El módulo de comunicaciones se ejecutará por interrupción asincrónica, producida al recibir un byte desde el puerto USB.

El módulo ADC, que lee las temperaturas y las convierte a digital, se realiza utilizando el hardware del μC desde el módulo principal, razón por la cual no se considera para el diseño del planificador.

Análisis de tiempos de respuesta

Para analizar las exigencias de tiempos de respuesta y determinar los valores críticos a considerar en el diseño del planificador, se estudian los componentes software de cada módulo y algunas restricciones de tipo hardware. Para este estudio se contempla que la ejecución se realizará sobre un μC con arquitectura Harvard (marcas PIC o AVR), en la que, en general, cada instrucción requiere un ciclo (cuatro ciclos de reloj), excepto las de salto directo y a subrutina que necesitan dos. El análisis se realiza

Figura 6. Diagrama de estados del sistema



Fuente: El Autor

considerando que la codificación del *kernel* se efectuará en el lenguaje ensamblador del μC (*assembly*) (Barry, 2004).

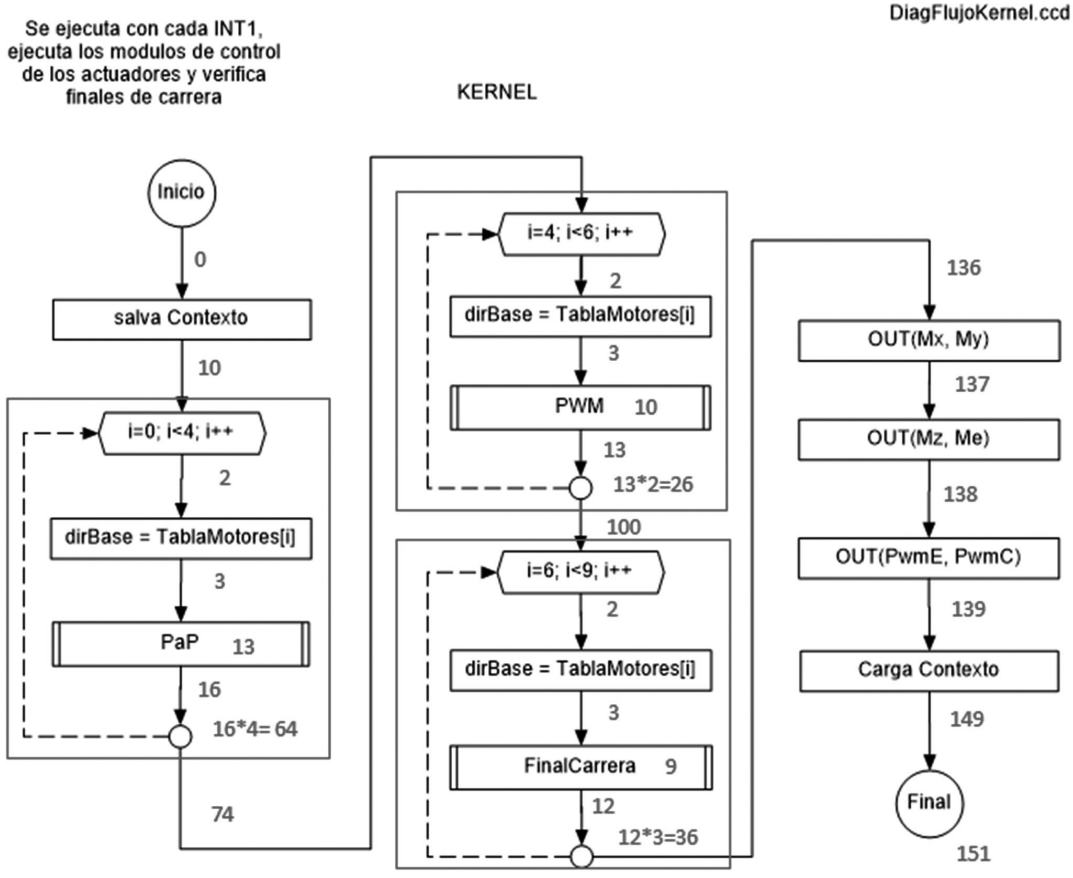
Estructura del kernel

Una vez definida la tarea a ejecutar en primer plano (módulo principal) y la atención a comunicaciones por interrupción no prioritaria, las tareas que deben cumplir con exigencias de tiempo real (control de actuadores) se planificarán consecutivamente usando un mecanismo de turno rotativo (*Round-Robin*) según el diagrama de flujo (Figura 7).

Al generarse una interrupción de nivel 1, se salva el contexto (principales registros asociados al proceso recién interrumpido), luego se repite el módulo PaP para los tres motores, el módulo PWM para las dos resistencias, el de finales de carrera para los tres ejes, se escriben las salidas resultantes en los puertos correspondientes y, por último, se recupera el contexto para continuar con el proceso de primer nivel.

Un cálculo sumario de los tiempos utilizados muestra que la ejecución acumulada más lenta es de 151 ciclos de instrucción (mínimo valor del período T).

Figura 7. Diagrama de flujo del *kernel*



Fuente: El Autor

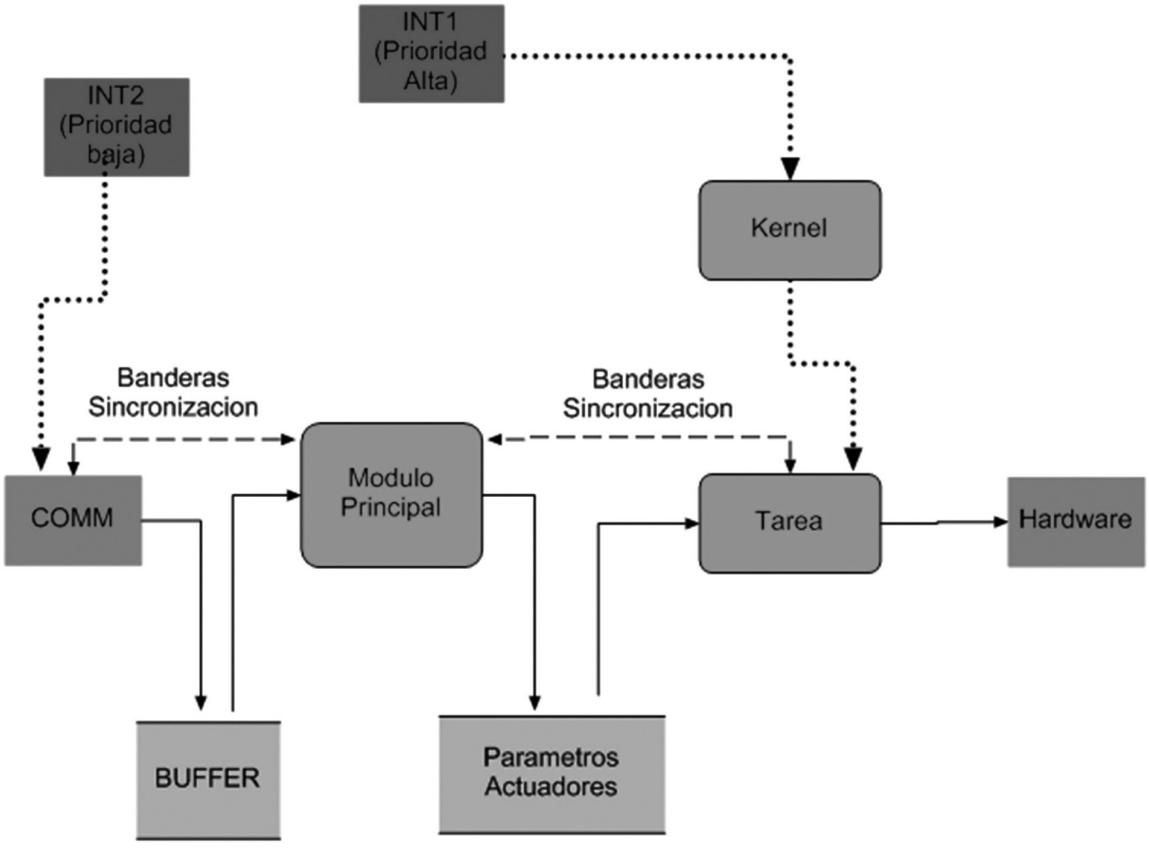
Flujo de datos del kernel

Dado el enfoque de programación de las tareas (pasadas rápidas) para mantenerlas en un tiempo reducido de ejecución y para lograr la sincronización entre el módulo principal y las controladas por el *kernel*, se utiliza una técnica de almacenes de datos y banderas de sincronización, como se muestra en la Figura 8. El valor de las variables es el que determina cuándo se deben realizar cambios en las señales de salida al hardware.

La interrupción de baja prioridad (INT2) se genera cada vez que se recibe un Byte por el puerto de comunicaciones, dando paso a la ejecución del módulo COMM, que almacena el comando recibido en un buffer circular y activa una bandera (*flag*) para indicar al módulo principal la llegada de un comando. Este mecanismo de interrupción se utiliza para evitar la técnica de *polling* (operación de consulta constante al hardware de recepción), debido al carácter no determinístico de los mensajes enviados desde el PC.

El módulo principal está atendiendo los comandos recibidos y trasladando los parámetros a las áreas de datos de los actuadores (motores PaP y resistencias PWM); este proceso de registro de parámetros también está sincronizado por banderas.

Figura 8. Diagrama de flujo de datos y sincronización de tareas



Fuente: El Autor

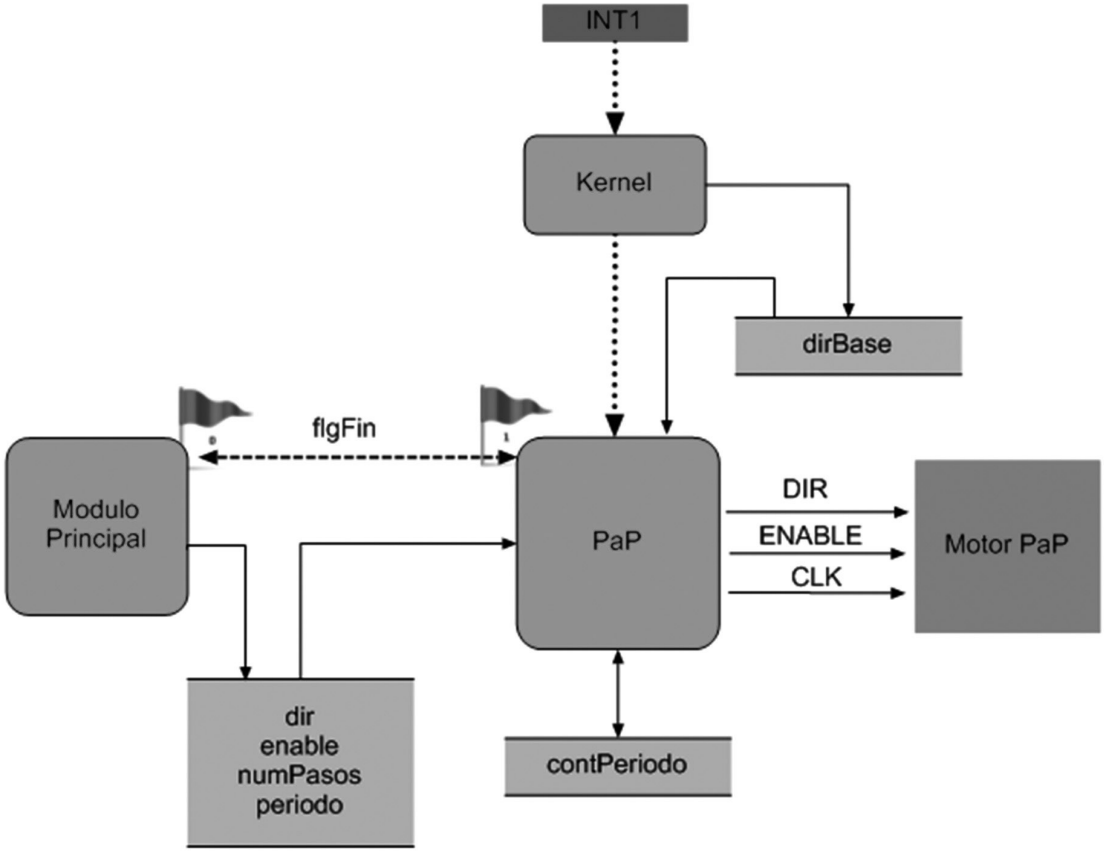
La interrupción de prioridad alta INT1 con frecuencia fija, inicia la ejecución del *kernel* quien lanza las diferentes tareas como se explicó en el apartado anterior.

Flujo de datos para un motor PaP

A un nivel más detallado (Figura 9), la sincronización entre el módulo principal y el de cada motor se logra de manera similar, con la característica adicional de que en cada llamada al módulo PaP, el k*ernel* actualiza un vector que apunta al inicio de los datos correspondientes al motor (dirBase), dichos datos son: sentido de giro (dir), señal de habilitación (enable), desplazamiento (numPasos) y un valor que determina la velocidad del movimiento (período). El hardware de cada driver PaP requiere las señales DIR, ENABLE y CLK.

La bandera flagFin es puesta en 1 (SET) por el módulo PaP para indicar que ya terminó la ejecución de la orden recibida. El módulo principal extraerá una nueva orden del buffer y la pasará al área de datos solo cuando encuentre 1 en la bandera (esperará hasta encontrar 1) y la colocará en 0 (RESET) para indicar al módulo PaP que puede iniciar un nuevo movimiento. Si en una nueva entrada a la tarea PaP para un motor determinado, se encuentra en 1, el módulo no hace nada y continuará así hasta encontrar un 0. En general, este es el mecanismo de sincronización entre tareas utilizado en el presente diseño.

Figura 9. Diagrama de flujo de datos del módulo PaP



Fuente: El Autor

Flujo de datos para PWM. Este mecanismo es más sencillo que el anterior (Figura 10), solo se requiere controlar el duty-cycle de la salida, a una frecuencia fija determinada por la constante KCICLO. Para ello, el módulo principal escribe el valor deseado en valor Ciclo; no se requiere sincronización con la tarea PWM.

Al igual que con PaP, se cuenta con un área de datos particular para cada dispositivo y un vector que apunta a su inicio y es actualizado por el *kernel*.

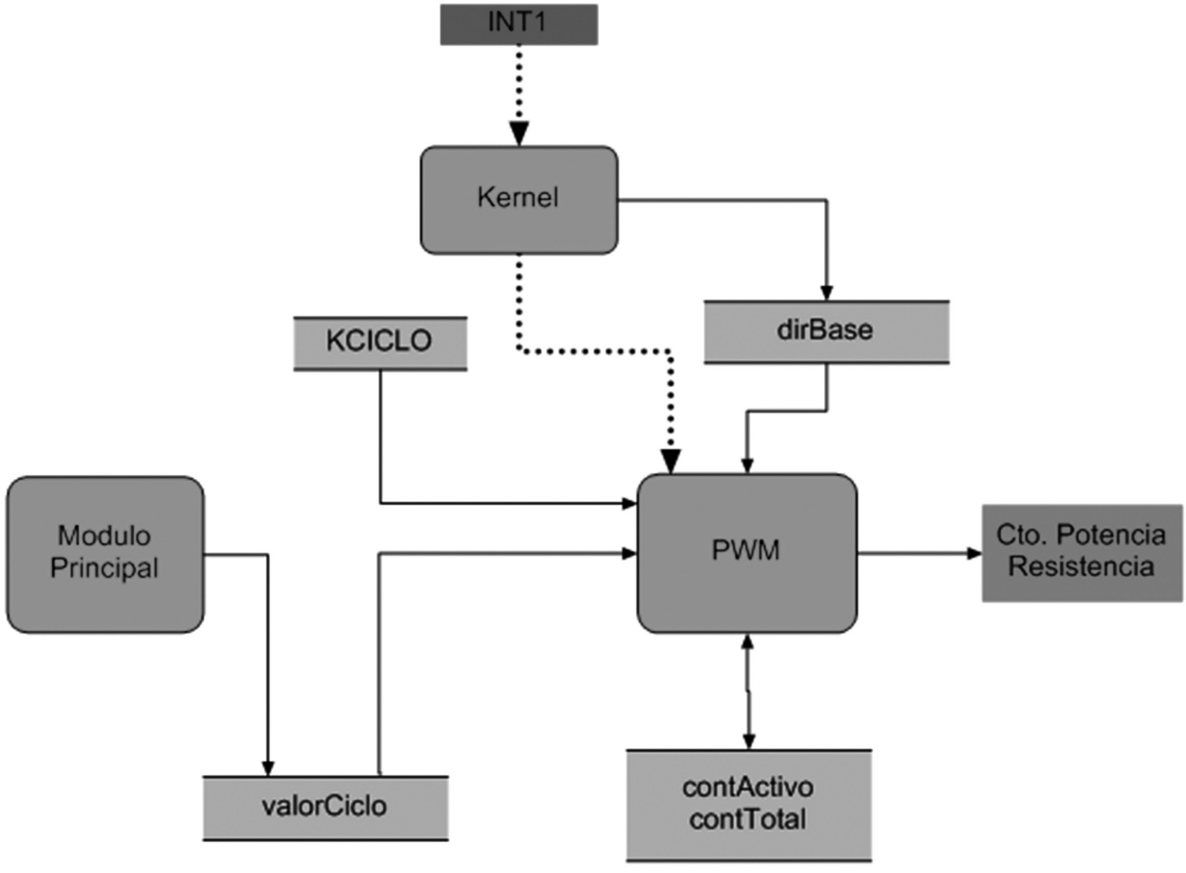
Mecanismo de interrupciones

El módulo principal siempre se ejecuta en primer plano y es detenido por las interrupciones.

La interrupción INT1, con prioridad alta es producida por el hardware (timer) del µC con un período fijo T. Cada vez que se produce esta interrupción, el kernel toma el control y ejecuta secuencialmente las tareas definidas previamente (en rojo en la Figura 11).

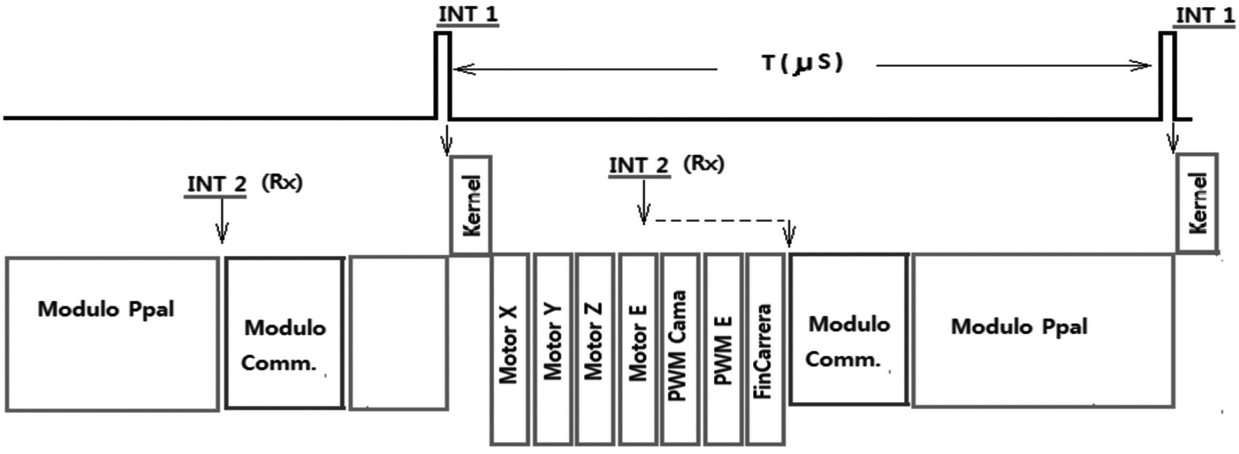
La interrupción INT2 (recepción de un dato) se puede presentar en dos momentos diferentes: mientras se está en primer plano, como se presenta a la izquierda en la gráfica 17, en cuyo caso COMM se ejecuta inmediatamente, devolviendo el control al módulo principal. La otra posibilidad es que INT2 se presenta mientras se está ejecutando el *kernel*, pero como este

Figura 10. Diagrama de flujo de datos del módulo PWM



Fuente: El Autor

Figura 11. Secuencias de ejecución de las tareas



Fuente: El Autor

deshabilitó el paso de interrupciones (menor prioridad), INT2 se atenderá solo al terminar la ejecución del *kernel*.

Respecto de INT2, que se produce cada vez que es recibido un Byte y asumiendo una velocidad de recepción de 9600 baudios con formato de 11 baudios/byte, se establece que, en el peor evento, se recibiría un byte cada 1.14 mS, por lo cual la posibilidad de que ocurra el último caso es muy baja.

Análisis de tiempos de interrupción

En una impresora 3D por extrusión de termoplástico, las velocidades de trabajo más comúnmente utilizadas son de 50mm/S a 60mm/S con velocidad de traslado del doble (Torres y Torres, 2012). La impresora diseñada utiliza un mecanismo con avance de 0,094248 mm/pulso (precisión mecánica teórica menor a 0.1 mm).

Asumiendo una velocidad máxima de 120 mm/S, se obtiene la frecuencia clk (frecuencia de pulsos al motor):

$$F_{clk} \geq [120mm/S] / [0.094248\ mm/pulso]$$

$$F_{clk} \geq 1274\ pps\ (pulsos\ por\ Segundo) \qquad (5)$$

El período T necesario debe ser, al menos, la mitad del correspondiente a la frecuencia F_{clk} (5), Entonces T debe cumplir:

$$T \leq 1 / [2 * 1274\ pps] \leq 390\ \mu S \qquad (6)$$

Para reducir la *granularidad* al cambiar la velocidad del motor, se debe mantener el período T lo más pequeño posible, pero permitiendo la ejecución de todas las tareas concurrentes y dejando espacio para el desarrollo segmentado de las tareas de primer plano.

CONCLUSIONES

El diseño cuidadoso de las tareas concurrentes administradas por el *kernel*, evitando la presencia de bucles de ejecución no determinística y realizando el control de las salidas hardware según el valor de los almacenes de datos

(variables de memoria), facilita el diseño del *kernel*, al permitir que la ejecución de cada tarea se ejecute en un tiempo muy corto.

La utilización de banderas o semáforos es una manera sencilla de coordinar las comunicaciones entre tareas, pero se debe definir con claridad cuál objeto la activa (*set*), cuál la desactiva (*reset*) y en qué momento, para evitar bloqueos.

RECONOCIMIENTOS

El presente documento es uno de los resultados obtenidos durante el desarrollo del proyecto de investigación aplicada: *Diseño y fabricación de una impresora 3D por extrusión de termoplástico,* el cual fue patrocinado por el Sistema de Investigación, Desarrollo Tecnológico e Innovación del SENA (SENNOVA) y desarrollado en el Centro de Industria y la Construcción de la Regional Tolima en el año 2015.

Quiero expresar mi reconocimiento y agradecimientos a las aprendices Leidy Montero, Tecnóloga en Mantenimiento Electrónico Industrial y Julieth Ríos, Tecnóloga en Análisis y Desarrollo de Sistemas de Información; ellas fueron quienes con su entusiasmo y dedicación hicieron posible este trabajo.

REFERENCIAS

Aldea Rivas, M. (2002). *Planificación de Tareas en Sistemas Operativos de Tiempo Real Estricto para Aplicaciones Empotradas* (Tesis doctoral). Recuperado de https://marte.unican.es/documentation/tesis-mario.pdf.

Alfonsi, A. (2012). Desarrollo e Implementación de un Planificador de Tiempo Real Ejecutivo Cíclico en un Microcontrolador de Gama Media para Algoritmos de Control con Lazos Independientes, 5to Congreso Iberoamericano de Estudiantes de Ingeniería Eléctrica (V Cibelec). Mérida, Venezuela.

Barry, R. (2004). *Multitasking on an AVR*. Recuperado de https://es.scribd.com/ document/49299134/Multitasking-on-an-AVR.

Camargo, D., & Andrade, J. (2013). Análisis de Rendimiento de dos Sistemas Operativos en Tiempo Real implementados en dos Arquitecturas de Hardware para la selección del Sistema Embebido que soportará el Software Command And Data Handling de la Misión Satelital Libertad 2. En: Eleventh Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2013) International Competition of Student Posters and Paper. Cancún, México.

Crespo E., & Alonso, A. (2006). Una panorámica de los Sistemas de Tiempo Real. *Revista Iberoamericana de Automática e Informática Industrial.* 3(2), 7-18. Recuperado de https://polipapers.upv.es/index.php/RIAI/article/view/8121.

Díaz-Ramírez, A., Mejía-Álvarez, P., & Leyva-del-Foyo, L. (2013). Comprehensive comparison of schedulability tests for uniprocessor rate-monotonic scheduling. *Journal of applied research and technology*, 11(3), 408-436. doi: https://doi.org/10.1016/S1665-6423(13)71551-7

Drake, J., González, M., Gutiérrez, J., López, P., Medina, J., & Palencia. J. (2014). *Modeling and Analysis Suite for Real Time Applications.* Recuperado de https://mast.unican.es/mast_description.pdf

Guevara, P., J. Valdez, J. S., & Delgado, G. (2014). Planificadores de tareas en tiempo real concurrentes: Una clasificación basada en funciones y teoría de conjuntos. *Revista Computación y Sistemas*, 18(4), 809–820. doi: 10.13053/CyS-18-4-1543

Laplante, P., & Ovaska, S. (2012). *Real Time Systems Design and Analysis. Tools for the Practitioner. Fourth Edition.* Recuperado de http://dl.finebook.ir/book/95/11575.pdf

Morales, G. (2014). *Sistema Operativo Multitarea de un Microcontrolador Programado en C++ y una interfaz programada en C#.* Recuperado de http://www.academia.edu/8021784/Sistema_Operativo_Multitarea_de_un_Microcontrolador_Programado_en_C_y_una_Interfaz_Programada_en_C_

Pastor, F. (2004-2005). *Apuntes de la Asignatura SITR, Sistemas en Tiempo Real*. Recuperado de http://www.uv.es/gomis/Apuntes_SITR/

Torres, E., León, J., & Torres, E. (2012). *Diseño y Construcción de una impresora 3D aplicando la técnica de Prototipado* rápido modelado por deposición fundida. Ponencia llevada a cabo en el Tercer Congreso Argentino de Ingeniería Mecánica, Buenos Aires, Argentina.

Uzcátegui, E., Dinarle, O., & Delgado, D. (2009).

Metodologías de desarrollo para sistemas de tiempo real. Un estudio comparativo. *Revista Universidad, Ciencia y Tecnología,* 13(50), 59-66. Recuperado de http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S1316-48212009000100008

Yépez, J., Guardia, J., Velasco, M., Ayza, J., Marti, P., & Fuertes, J. (2005). Ciclic: *Herramienta Para Crear Planificadores Cíclicos Factibles*. Recuperado de http://www.aurova.ua.es/previo/ja2005/comu/3191-JornadasAutomatica2005.pdf

Yépez, J. (2007). *Diseño e Implementación de una Herramienta para la Planificación de Ejecutivos Cíclicos*. (Tesis de pregrado). Recuperado de http://upcommons.upc.edu/bitstream/handle/2099.1/4451/Memoria.pdf?sequence=1

Cómo referenciar este artículo

Mora Méndez, M. L. (2017). Diseño de un kernel de tiempo real para el control de una impresora 3D. *Revista Vía Innova*, 4(4), 4-21. doi: https://doi.org/10.23850/2422068X.1177