Port de ZeOS a arquitectura ARM: Raspberry Pi

Grau en Enginyeria Informàtica

Especialitat
Enginyeria de Computadors

27 de juny, 2014

Autor
Albert Segura Salvador

Director
Juan Jose Costa Prats

Departament d'Arquitectura
de Computadors

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONA**TECH**

Facultat d'Informàtica de Barcelona

Resum

L'arquitectura ARM ha vist un important augment d'adopció i popularitat en els últims anys gràcies als dispositius mòbils i encastats, augment que no s'ha produït en arquitectures més populars com Intel x86.

El document cobreix el desenvolupament del projecte de port del sistema operatiu ZeOS de l'arquitectura Intel x86 a ARM, concretament al dispositiu Raspberry Pi. Es detallen les característiques rellevants de l'arquitectura i els components principals del sistema operatiu.

El resultat final és un sistema operatiu multiprocés, ZeOSpi, funcionant en la Raspberry Pi i fent ús dels perifèrics que ofereix el dispositiu.

Resumen

La arquitectura ARM ha visto un importante aumento de adopción y popularidad en los últimos años gracias a los dispositivos móviles y empotrados, aumento que no se ha producido en arquitecturas más populares como Intel x86.

El documento cubre el desarrollo del proyecto de port del sistema operativo ZeOS de la arquitectura Intel x86 a ARM, concretamente el dispositivo Raspberry Pi. Se detallan las características relevantes de la arquitectura y los componentes principales del sistema operativo.

El resultado final es un sistema operativo multiproceso, ZeOSpi, funcionando en la Raspberry Pi y haciendo uso de los periféricos que ofrece el dispositivo.

Abstract

The ARM architecture has seen a significant increase in adoption and popularity in past years because of mobile and embedded devices, an increase that has not occurred to more popular architectures like Intel x86.

The paper covers the development of the project to port the ZeOS operating system from Intel x86 to ARM architecture, specifically to the Raspberry Pi device. The relevant characteristics of the architecture and main components of the operating system are also detailed.

The end result is a multithreaded operating system, ZeOSpi, running on the Raspberry Pi and using peripherals offered by the device.

Índex

Índex de figures

Índex de taules

1 Introducció

Aquest document és la memòria del treball final de grau **"Port de ZeOS a arquitectura ARM: Raspberry Pi"**.

El document està organitzat en dues parts diferenciades: la primera part es centra en la gestió del projecte i en els aspectes econòmics i de sostenibilitat, i la segona part detalla els aspectes tècnics del projecte i el seu desenvolupament. Finalment, el document també inclou un apartat de revisió del projecte, conclusions i treball futur.

Hi figuren també dos annexos: el primer sobre els fitxers proporcionats junt amb aquest document, i el segon, sobre instruccions assemblador rellevants usades en el projecte.

2 Context

El context i el conjunt de persones a les quals es dirigeix el resultat del projecte es poden classificar en els apartats següents.

2.1 Àmbit acadèmic

L'enfoc principal del projecte és l'àmbit acadèmic i educatiu, en el qual pot ser més rellevant.

El Sistema Operatiu (SO) ZeOS per a Intel x86 és usat en el àmbit educatiu de la FIB, concretament en l'assignatura de SO2 [3] i SOA [4], per a l'aprenentatge i familiarització de conceptes bàsics de sistemes operatius entre els estudiants. En aquestes assignatures, es parteix del boooSO ZeOS sense cap funcionalitat important; a partir d'aquí l'estudiant desenvolupa les diferents parts del SO seguint una sèrie d'entregues en les quals s'han d'implementar funcionalitats com: l'entrada amb interrupcions, crides de sistema, gestió de multiprocés, planificadors, sistema d'Entrada/Sortida (E/S), gestió de sincronització de processos i gestió de memòria dinàmica.

El resultat d'aquest projecte, a més de poder servir pel mateix propòsit pel qual s'utilitza ZeOS de x86 en les assignatures mencionades, podria alhora aprofundir en conceptes de sistemes encastats i dispositius mòbils, sistemes que estan veient un creixement molt important en els últims temps, fet que no succeeix amb els sistemes pc tradicionals, en el qual està basat el ZeOS original.

En aquest mateix àmbit, doncs, el projecte també pot servir com a punt de partida d'altres treballs finals; ampliant-ne funcionalitats, afegint suport a d'altres plataformes, incorporant característiques més avançades de sistemes operatius i implementant característiques típiques de sistemes encastats, per exemple, tenint en compte restriccions de consum, entre d'altres.

2.2 Comunitat entorn la Raspberry Pi

Aquest projecte pot ser interessant per a la comunitat entorn la Raspberry Pi. Atès que, l'objectiu principal de la Raspberry Pi serveix un propòsit educatiu, el projecte té una gran implicació en l'àmbit acadèmic.

Per aquest fet, la comunitat entorn la Raspberry Pi es veuria beneficiada perquè es podrien crear nous projectes guiats per la comunitat, a partir d'aquest projecte, que enriquissin i ampliessin l'abast de la comunitat i el propòsit educatiu de la Raspberry Pi. En resum, es podria estendre el caràcter educatiu del SO de l'àmbit purament acadèmic a un àmbit més ampli i divers.

2.3 Àmbit professional

L'àmbit professional és el més limitat en comparació als anteriors. És així ja que avui existeixen moltes alternatives de sistemes operatius per la Raspberry Pi, la gran majoria sistemes Linux.

Per l'àmbit professional, segurament les principals característiques que requeririen d'un SO per a la Raspberry Pi serien: un suport ampli de dispositius externs, un SO consolidat, i un entorn conegut per a facilitar el desenvolupament d'aplicacions. De sistemes operatius que s'enfoquen en aquesta direcció, o que acompleixen aquests punts n'hi ha en gran quantitat, un d'ells per exemple és Linux, i el seu seguit de distribucions. Com que el projecte no té aquests objectius, seria difícil que una empresa es decantés per aquest SO tota vegada que per a desenvolupar els seus projectes poden trobar al mercat solucions amb més recorregut i amb relació a les quals poden tenir-hi experiència prèvia.

Tot i així, aquest projecte ofereix un producte que pot arribar a ser una solució a considerar per alguna empresa o projecte concret. Com a punts favorables, aquest SO és més senzill i mantenible que altres solucions. Per tant pot arribar a ser més fàcil la familiarització amb l'entorn de desenvolupament i molt probablement és una solució que dóna més llibertat per a personalitzar el producte final que altres sistemes operatius que suporten la Raspberry Pi.

3 Estat de l'art

3.1 Elecció del dispositiu

Com es descriu en l'apartat de la motivació, un dels punts del projecte és aprofundir en el coneixement sobre l'arquitectura ARM. Per poder satisfer aquest requeriment, primer era necessari trobar un dispositiu en el qual es pogués desenvolupar un SO i disposar-ne a un preu raonable.

També era desitjable que el model de Unitat Central de Processament (CPU en anglès) ARM tingués un **sol processador** i no fos massa modern, per facilitar el desenvolupament del SO i tenir un hardware estable i consolidat. El rendiment que se'n pogués obtenir no tenia un pes molt important, car difícilment el SO en farà un ús intensiu. Alhora, també era important que el dispositiu tingués una comunitat prou gran darrere i que hi hagués documentació dels sistemes del dispositiu en grau suficient com per a poder desenvolupar els diferents components del SO, com el d'E/S.

Tot seguit, es mostra un llistat de possibles dispositius pel port de ZeOS a arquitectura ARM. D'entrada es descarten dispositius similars que no siguin arquitectura ARM, com són la **Intel Galileo** o **Arduino Yún**.

Raspberry Pi

Figura 1: Raspberry Pi, Model B

La **Raspberry Pi** en totes les característiques mencionades és una molt bona opció, pels motius següents: el seu preu molt reduït, un CPU amb un sol processador, una comunitat molt important i una documentació suficientment completa per a realitzar el port del ZeOS. Com a principal inconvenient, el sistema de boot és específic del Sistema en Xip (SoC en anglès) i d'ús difícil. Per finalitzar, la Raspberry Pi també té a favor la seva vessant en el món de l'educació i l'aprenentatge, fet que lliga amb el context acadèmic del projecte i agrega valor afegir a aquesta elecció.

Odroid U3

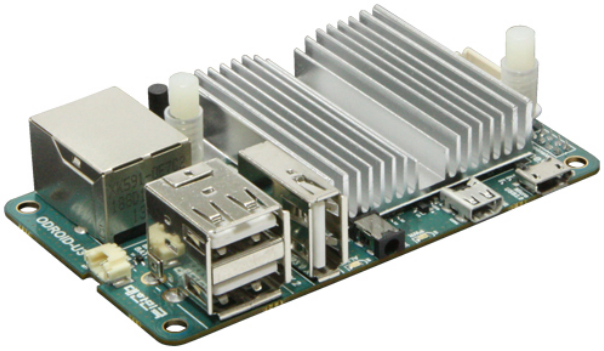

Figura 2: Odroid U3

L'**Odroid U3** [11], fabricat per Hardkernel, és un dispositiu similar a la Raspberry Pi, utilitzat principalment com a plaques de desenvolupament per a Android, i de preu, encara que ajustat, més alt que la Raspberry Pi. Tot i que en conjunt la Odroid U3 és un dispositiu més potent que la Raspberry Pi, el fet d'haver de desenvolupar el SO per a un multiprocessador afegiria una complexitat extra al projecte i al SO, que li restaria viabilitat. Si bé també te una comunitat no massa gran, aquesta està més basada en sistemes Android. A més, Odroid U3 incorpora el sistema U-Boot com a sistema de boot, que és un sistema més universal i fàcil de treballar-hi.

Arduino Due

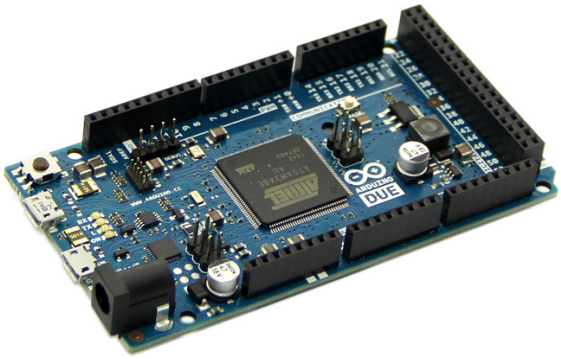

Figura 3: Arduino Due

L'**Arduino Due** [5] és una revisió de la popular placa Arduino Uno que incorpora una CPU ARM d'Atmel, amb una versió de l'arquitectura de la CPU superior a la Raspberry

Pi. Però tot i així, és un processador menys potent i més limitat, especialment pensat per a sistemes encastats, malgrat disposar de components necessaris pel SO com és una Unitat de gestió de memòria (MMU en anglès) i diferenciació entre mode privilegiat i no privilegiat.

Arduino també compta amb una comunitat molt gran, més orientada de cara a l'electrònica i als referits sistemes encastats, i només recentment han proliferat dispositius Arduino que compten amb un SO. Alhora, aquest dispositiu no disposa de sortida de vídeo, que, si bé no s'usarà en aquest projecte, és interessant de cara a treballs futurs. L'Arduino, també incorpora un sistema de bootloader propi substituïble, a diferencia de la Raspberry Pi. Per acabar és més complicat gravar el SO en la memòria del dispositiu.

Aquesta era una opció important a considerar, però la documentació d'aquest dispositiu i la seva comunitat estan més enfocades a sistemes electrònics que no pas al desenvolupament de sistemes operatius. Alhora, els fets d'haver de reescriure el bootloader, de les complicacions amb la memòria i de no disposar de sortida de vídeo fan preferible l'elecció de la Raspberry Pi.

Udoo

Figura 4: Udoo board

La placa **Udoo** [12] és un dispositiu posterior a la Raspberry Pi basat en aquest i en l'Arduino Due, que a més ofereix una CPU, multiprocessador molt més potent i un sistema gràfic també superior al de la Raspberry Pi. Alhora, incorpora l'Arduino Due mantenint tot el entorn de desenvolupament d'Arduino. Té una comunitat relativament petita i incorpora el sistema d'arrencada U-Boot.

Aquest dispositiu a part de comptar amb un multiprocessador, alhora també disposa del processador de l'Arduino Due afegint encara més complexitat al sistema que les anteriors opcions. Si bé és el sistema més potent mostrat aquí, també n'és el més car.

3.2 Raspberry Pi

Naixement de la Raspberry Pi

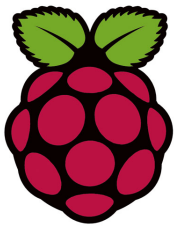

Figura 5: Raspberry Pi logo

El mini-computador Raspberry Pi és un dispositiu creat per la Raspberry Pi Foundation, que neix el 2006, amb l'idea de **promoure l'estudi de conceptes bàsics de programació** i informàtica a les escoles per suplir la falta d'aquests conceptes en els estudiants.

Al començament, els prototipus es basen en microcontroladors de sistemes encastats, però finalment s'opta per processadors dissenyats per a dispositius mòbils degut a la seva popularitat. [7]

Finalment, surt a la llum el 29 de febrer de 2012, any de traspas, té les dimensions d'una targeta de crèdit, i es popularitza ràpidament pel seu baix cost, la versatilitat d'usos i aplicacions, i gràcies a una comunitat activa. El dispositiu es ven en dos models per sota dels 30€, i a finals d'Octubre de 2013 ja s'havien venut més de **2 milions de Raspberry Pi**.

Característiques Hardware

El dispositiu es basa en un SoC Broadcom BCM2835 [2], que és el que incorpora les capacitats gràfiques (VideoCore IV) i també un processador ARM (ARM1176JZF-S, ARMv6) [17]. Disposa d'entre 256 i 512 MB de memòria RAM, sortida de vídeo HDMI, i possibilitat d'interfície Ethernet.

A més a més, el dispositiu proporciona accés a interfícies de més baix nivell com els pins GPIO, interfícies UART i SPI, i CSI per a càmera vídeo, especialment útils per integrar el dispositiu en aplicacions de l'àmbit electrònic, com ara la robòtica.

En la taula següent es llista una descripció més completa del hardware que incorpora cadascun dels models de la Raspberry Pi.

	Model A	Ambdós	Model B
Preu	18€		25€
SoC		Broadcom BCM2835	
CPU		700 MHz ARM1176JZF-S ARMv6	
GPU		Broadcom VideoCore IV @ 250 MHz	
Memòria	256 MB		512 MB
USB 2	1		2
Vídeo		HDMI, RCA, LCD	
Xarxa	-		10/100 Mbits/s
Perifèrics		GPIO, UART, I2C, SPI, I2S	
Emmagatz.		SD/MMC/SDIO	
Cons. energ.	300mA		700mA

Taula 1: Descripció hardware de la Raspberry Pi [8]

3.3 Suport d'arquitectures hardware de ZeOS

El SO ZeOS, es basa i té semblances a un sistema Linux 2.4. ZeOS té característiques de sistemes operatius moderns, com ara gestió de multitasca i execució sense privilegis.

Actualment hi han versions de ZeOS per a Intel x86, usada en les assignatures de SO2 i SOA de la carrera, i per a SISA, desenvolupat per Zeus Gomez al seu projecte final de carrera. [10]

Abans de la realització d'aquests projecte, ZeOS no disposa de cap implementació per a l'arquitectura ARM, ni per a la Raspberry Pi.

3.4 Sistemes operatius

Suport de sistemes operatius per a la Raspberry Pi

El SO principal de la Raspberry Pi és Linux (distribucions com Raspbian o Raspbmc), tot i així, la gran popularitat del dispositiu ha derivat en una gran quantitat de sistemes operatius i distribucions de Linux portades a Raspberry Pi. A continuació es mostra una llista dels sistemes operatius que suporten la Raspberry Pi [21]:

- **AROS:** Es un SO de recerca, basat en AmigaOS.
- **Haiku:** SO basat en BeOS
- **Linux:** Android, Raspbian, Firefox OS, Arch Linux, OpenELEC...
- **Plan9:** SO de recerca.
- **RISC OS:** SO per a arquitectures RISC, principalment ARM.

- **Unix:** FreeBSD i NetBSD.
- **Windows CE:** SO de Windows, per a sistemes encastats.

Sistemes operatius educatius

El sistemes operatius educatius normalment s'originen i són utilitzats en l'àmbit acadèmic, per a l'ensenyament de conceptes de sistemes operatius a estudiants o bé per a la recerca sobre conceptes avançats de sistemes operatius. Per això, difícilment acaben suportant més d'una arquitectura.

Pel que fa a sistemes operatius educatius com ZeOS, hi ha d'altres sistemes com:

- **JOS:** SO educatiu utilitzat al MIT, basat en Unix. Treballa sobre arquitectura Intel x86. Té característiques similars a ZeOS, si bé aquest inclou un sistema de fitxers simple a diferencia del SO ZeOS. [19]
- **OS/161:** SO educatiu utilitzat al Harvard, basat en BSD. Treballa sobre arquitectura MIPS. També incorpora característiques similars al SO ZeOS i alhora implementa una capa VFS per als sistemes de fitxers més complexa que la de JOS. [14]
- **Minix:** SO educatiu, inspiració del sistema Linux, basat en Unix. Originat el 1987, inicialment suportava sistemes IBM PC, i amb l'ultima versió només a Intel. Es un SO complert i inclou programes com X11, vi, gcc...

Pel que fa a sistemes de recerca, podem trobar-nos amb els següents:

- **Plan9:** SO de recerca de Bell Labs. Creat com a reemplaçament del sistema Unix, partint d'aquesta base i redefinint-ne conceptes. Ha aconseguit un cert impacte en sistemes operatius com Linux, on s'han portat dissenys implementats en aquest SO.
- **AROS:** SO de recerca basat en AmigaOS. Creat el 1995 principalment suporta sistemes IBM PC. Sistema complert però amb reduït suport de programes.

4 Abast del projecte i objectius

4.1 Motivació

La motivació d'aquest treball es pot resumir en el següents punts:

- Interès en experimentar amb l'arquitectura ARM
- Interès en el desenvolupament de sistemes operatius
- Interès en sistemes encastats i microcontroladors

Interès en l'arquitectura ARM. Tot i que durant la carrera se'n comenten les característiques principals i les diferències amb altres arquitectures com Intel x86, no es realitzen pràctiques ni treballs que entrin en profunditat en aquesta. Per aquest motiu em va semblar interessant el projecte, que m'ha de permetre adquirir experiència en aquesta plataforma, fet que considero important donat que cada cop més el mercat dels ordinadors sembla disminuir en favor dels dispositius mòbils, com ara tauleta i telèfons intel·ligents, on arquitectures com ARM predominen notòriament.

Interès en el desenvolupament de sistemes operatius. Ja que si bé és cert que en assignatures de la carrera, partint d'un sistema base, es realitza en profunditat, trobo interessant el desenvolupament des del principi, canviant d'arquitectura, per tal d'enfrontar els problemes que pugui plantejar i així tenir una visió més global.

I finalment, interès en sistemes encastats i microcontroladors. És una àrea del meu interès, en la que he desenvolupat més d'un projecte personal i on el mini-computador Raspberry Pi és un exponent important dels últims anys.

4.2 Objectius generals

El principal objectiu del projecte és **desenvolupar un sistema operatiu bàsic però funcional per al mini-computador Raspberry Pi**, i en conseqüència, donar el suport necessari per l'arquitectura ARMv6 i el SoC que aquest incorpora. Aquest sistema operatiu es basarà en el codi de ZeOS per l'arquitectura Intel x86, i pretén conservar, en la mesura del possible, les estructures, funcionalitats i l'organització general del sistema ZeOS original.

Com a objectius secundaris, i de caràcter opcional, **el SO ha d'incloure característiques avançades**, com ara ser un sistema multiprocés, incloure mecanismes de sincronització entre processos i incloure gestió de memòria dinàmica.

4.3 Abast del projecte

Per tal d'assolir l'objectiu principal del projecte i aconseguir un SO funcional, l'abast d'aquest és detalla en els punts següents:

- Sistema de Boot, tant del sistema de desenvolupament com del dispositiu final.
- Gestió de modes de funcionament del processador.
- Gestió de la memòria, tant del processador com del SoC.
- Gestió del sistema d'interrupcions, tant del processador com del SoC.
- Gestió de perifèrics i sistemes d'E/S, concretament, el sistema UART del SoC.

Per a la realització dels objectius secundaris, l'abast es composa dels punts següents:

- Gestió de processos i implementació de funcions i estructures pel seu funcionament.
- Inclusió de funcions i estructures per la sincronització entre processos.
- Inclusió de funcions i estructures per la gestió de memòria dinàmica.

4.4 Possibles obstacles

Els possibles obstacles que poden afectar la realització del projecte són:

- Dificultat de la comprovació d'errors de funcionament, especialment en el dispositiu final.
- Documentació poc detallada o incomplerta per part del fabricant del SoC del dispositiu final.
- Dificultat en continuar el desenvolupament si determinades parts del sistema requereixen més temps del establert.

4.5 Desenvolupament

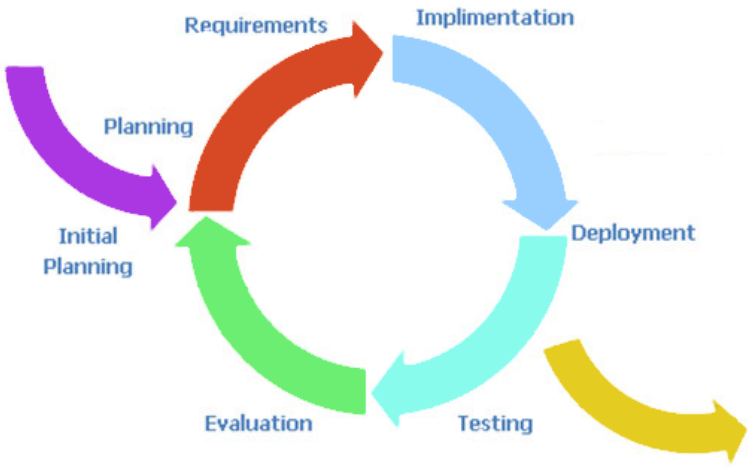

Figura 6: Metodologia de treball

La metodologia de treball a utilitzar serà la de **desenvolupament iteratiu i incremental**. Així doncs, en més d'un component del sistema s'iterarà durant les etapes del projecte per afegir característiques necessàries de cada etapa en que es trobi el projecte. Alhora, el desenvolupament també serà incremental, atès que divisió del projecte en etapes requerirà l'assoliment de determinats nivells de característiques d'un component del sistema.

El desenvolupament s'inicia amb la planificació inicial del projecte, tot seguit s'itera sobre un cicle d'anàlisis, desenvolupament, comprovació d'errors i documentació en cadascuna de les etapes del projecte per desenvolupar de forma incremental el sistema. En acabat, s'entra en la fase final del projecte on s'enllesteix la documentació i s'entrega el projecte.

Les **eines de seguiment del treball** consistiran en el seguiment proper de la planificació, alhora també es tindrà en compte la comunicació directa i freqüent amb el director del treball, via correu electrònic (informant de forma bisetmanal del avenç de la feina) o mitjançant reunions ocasionals per aclarir punts del projecte on puguin sorgir dificultats durant el seu desenvolupament.

El **mètode de validació** consistirà en el control d'errors, durant la implementació de les funcionalitats i especialment al final de cada etapa del projecte, per tal de validar el correcte funcionament de la implementació dissenyada.

5 Planificació del projecte

5.1 Diagrama de Gantt

La planificació d'aquest projecte s'especifica mitjançant un **diagrama de Gantt**. Aquesta planificació es basa en el total d'hores que especifiquen els crèdits ECTS, suposant cada crèdit 30 hores de dedicació. Al treball final de grau es dediquen 15 crèdits, per tant obtenim un total de 450 hores de dedicació.

Les tasques han estat organitzades en dies de 8 hores de dedicació, per tant són aproximadament 56 dies de duració. Tot i així, la planificació s'estén entre 62 i 71 dies (segons si es té en compte, o no, l'etapa d'extensió de caràcter opcional).

En el projecte hi treballarà una persona, i respecte als recursos materials requerirà un ordinador capaç de fer el **cross-compiling** d'ARM i debugging amb la maquina virtual Qemu, i també serà necessari una Raspberry Pi amb una targeta SD pel SO i un adaptador d'interfície sèrie a usb per poder-hi interaccionar des del ordinador.

Finalment el projecte té data d'inici a dia 1 de Febrer de 2014 i finalitza el dia 12 de maig de 2014. A continuació es mostra el diagrama de Gantt del projecte i l'apartat següent es detallen les diferents etapes i tasques.

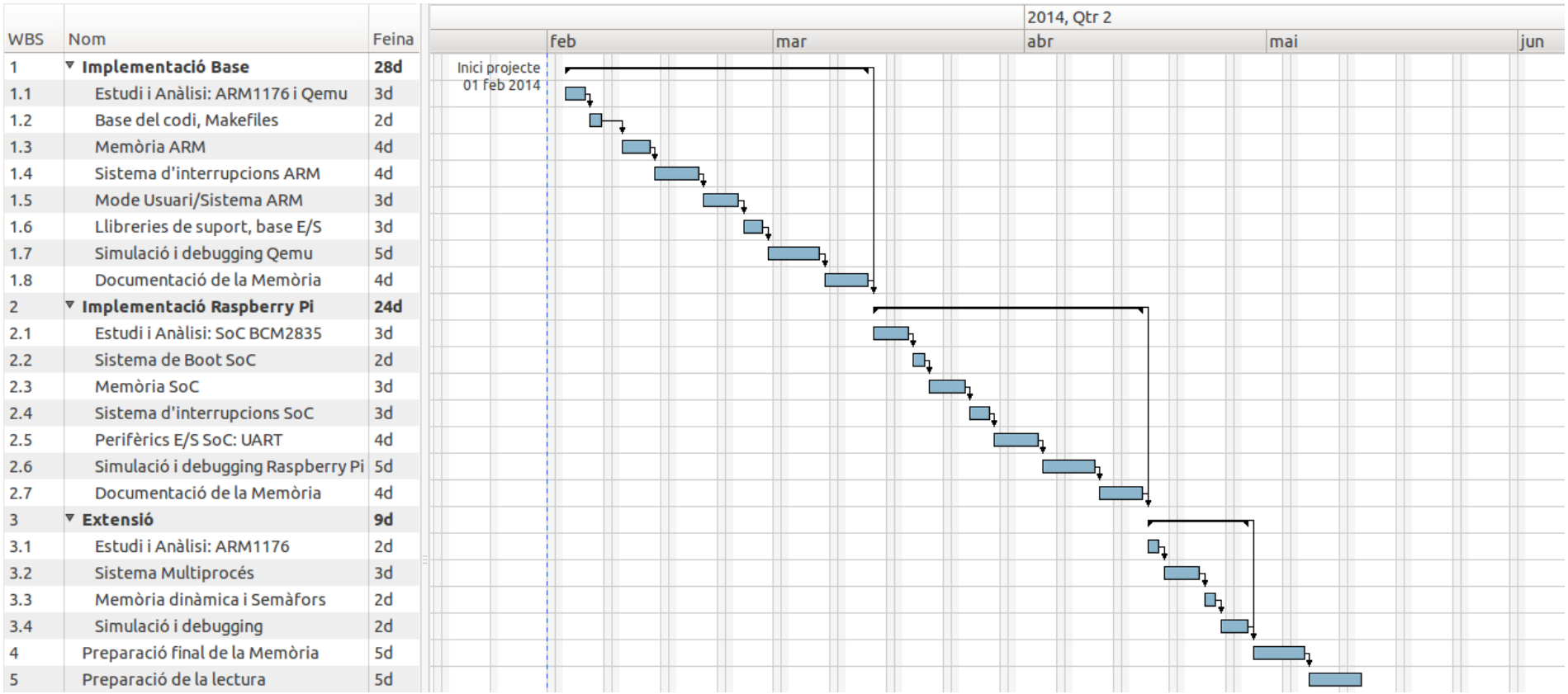

Figura 7: Diagrama Gantt de la planificació inicial

5.2 Definició d'etapes i tasques

El projecte consta de tres etapes principals i una final, que són:

- **Implementació Base**

  Suposa un total de 28 dies de feina. Superada aquesta etapa que engloba el desenvolupament inicial del SO,s'implementen la majoria de les funcionalitats en l'entorn de desenvolupament de la màquina virtual Qemu, dedicant-hi al final una sèrie de dies per al debugging i la documentació per tal portar-ho al dia i no deixar-ho per al final del projecte. Les tasques que agrupa són:

  - **Estudi i Anàlisi ARM1176 i Qemu:** 3 dies.
    Estudi inicial de la documentació d'ARM i l'entorn de desenvolupament.
  - **Base del codi, Makefiles:** 2 dies.
    Neteja inicial del codi i preparació de l'entorn de desenvolupament.
  - **Memòria ARM:** 4 dies.
    Implementació del sistema de memòria del SO.
  - **Sistema d'interrupcions ARM:** 4 dies.
    Implementació del sistema d'interrupcions d'ARM.
  - **Mode Usuari/Sistema ARM:** 3 dies.
    Implementació de la gestió dels entorns d'execució.
  - **Llibreries de suport, base E/S:** 3 dies.
    Finalització del port agregant funcions de suport i la base del sistema d'E/S.
  - **Simulació i debugging Qemu:** 5 dies.
    Testeig del codi desenvolupat fins al moment.
  - **Documentació de la Memòria:** 4 dies.
    Recull de la informació generada durant l'etapa.

- **Implementació Raspberry Pi**

  Suposa un total de 24 dies de feina. Aquesta etapa implica el port final del SO al seu entorn real d'execució. Superada aquesta etapa, el SO s'executa correctament en la Raspberry Pi i implementa la UART com a mètode d'E/S del SO. Com en l'etapa anterior, es dediquen uns dies al debugging i documentació. Les tasques que agrupa són:

  - **Estudi i Anàlisi SoC BCM2835:** 3 dies.
    Estudi de la documentació del SoC.
  - **Sistema de Boot SoC:** 2 dies.
    Gestió del sistema de boot del dispositiu.
  - **Memòria SoC:** 3 dies.
    Implementació del sistema de memòria del SoC.
  - **Sistema d'interrupcions SoC:** 3 dies.
    Implementació del sistema d'interrupcions del SoC.

- **Perifèrics E/S SoC UART:** 4 dies.
  Implementació i integració del mecanisme d'E/S UART.
- **Simulació i debugging Raspberry Pi:** 5 dies.
  Testeig del codi desenvolupat fins al moment.
- **Documentació de la Memòria:** 4 dies.
  Recull de la informació generada durant l'etapa.

- **Extensió**

  Suposa un total de 9 dies de feina. Aquesta etapa opcional agrupa els objectius secundaris d'extensió del SO. Si la planificació es compleix, un cop superada aquesta etapa, el SO incorporà funcionalitats més avançades, com són capacitat de multiprocés i mecanismes de sincronització, i memòria dinàmica. Aquesta etapa inclou uns dies de debugging i cap de documentació que s'inclouen en l'etapa següent. Les tasques que agrupa són:

  - **Estudi i Anàlisi ARM1176:** 2 dies.
    Revisió especifica de la documentació d'ARM.
  - **Sistema Multiprocés:** 3 dies.
    Implementació de la gestió de tasques i suport multiprocés.
  - **Memòria dinàmica i Semàfors:** 2 dies.
    Implementació de sistemes de sincronització i de memòria dinàmica.
  - **Simulació i debugging:** 2 dies.
    Testeig del codi desenvolupat fins al moment.

- **Preparació final**

  Suposa un total de 10 dies de feina. Aquesta etapa agrupa la preparació final de la memòria, 5 dies, i de la lectura, també 5 dies.

5.3 Valoració d'alternatives i pla d'acció

El projecte es pot veure endarrerit i incomplir la planificació inicial degut als possibles obstacles plantejats anteriorment. Si durant la realització del projecte això succeís, es podrien **eliminar tasques de l'etapa d'Extensió** i dedicar el temps que tenien assignat a la tasca que hagi requerit més dedicació. Donat que aquesta etapa es realitza cap al final del projecte i està plantejada com un afegit a les funcionalitats bàsiques que es volen proporcionar pel SO, el fet d'eliminar-ne tasques no té grans implicacions en el desenvolupament del projecte.

Aquest projecte finalitza unes setmanes abans del període de lectures. Així doncs, si el desenvolupament s'allargués en el temps, hi ha un marge per poder-lo finalitzar intentant realitzar totes les tasques especificades. L'acció d'eliminar tasques o endarrerir la finalització del projecte es decidiria segons el seu estat i tenint en compte com s'ha desenvolupat des de l'inici.

6 Pressupost inicial del projecte

Els costos del projecte es desglossen en els apartats següents: recursos humans, recursos materials, recursos de software, despeses indirectes i finalment el total agregat de les despeses.

El cost total del projecte és de **18.752,1 €**, desglossats en diferents apartats a continuació.

6.1 Despesa en recursos humans

En la taula següent es detalla el cost en recursos humans. Els salaris estan extrets de treballs fi de grau d'altres anys. El total d'hores de cada rol es determina segons les categories següents basades en les tasques especificades pel diagrama de Gantt:

- **Documentació i preparació de la lectura:** Suma un total de 18 dies, 144 hores. Cap de projecte.
- **Etapes d'estudi i Anàlisi:** Suma un total de 8 dies, 96 hores. Dissenyador.
- **Etapes de debugging:** Suma un total de 12 dies, 96 hores. Tester.
- **La resta de tasques:** 33 dies, 264 hores, es reparteixen entre programador, dos terceres parts i dissenyador, la part restant.

Rol	**Preu (€/h)**	**Temps (h)**	**Import (€)**
Cap de projecte	40	144	5.760
Dissenyador	20	152	3.040
Programador	20	176	3.520
Tester	12	96	1.152
Total			**13.472**

Taula 2: Despesa inicial en recursos humans. Font: ICSA Grupo 2013 -2014.

6.2 Despesa en recursos materials

Els recursos materials del projecte, per tractar-se del desenvolupament d'un SO, no són molt significatius en comparació amb els humans, si bé per dur a terme el projecte es requereix d'un mini-computador Raspberry Pi i accessoris, a la segona etapa, i un ordinador per desenvolupar el SO.

La part imputable dels costos d'aquest material al projecte es pot calcular amb la fórmula d'amortització següent:

$$cost\_amortitzat = \frac{cost\_total * temps\_projecte}{temps\_amortitzable}$$

Per tant, si el projecte dura 71 dies i assumim un temps amortitzable de 240 dies laborables en un any, el cost imputable al projecte és:

$$cost\_amortitzat = \frac{654,5€ * 71\ dies}{240\ dies} = 193,62\ €$$

En la taula següent es detallen les despeses en recursos materials, junt amb l'amortització.

Material	**Import (€)**
Ordinador portàtil	600
Raspberry Pi B	34
Targeta SD 4GB	10
Caixa Raspberry Pi	6
Adaptador Sèrie-Usb	4,5
Total	**654,5**
Total amortitzat	**193,62**

Taula 3: Despesa inicial en recursos materials.

6.3 Despesa en software

El software utilitzat pel desenvolupament del projecte és software de lliure distribució i gratuït. Per tant no repercuteix sobre les despeses totals del projecte.

6.4 Despeses indirectes

Pel que fa a les despeses indirectes, hi podem comptar les despeses de llum, aigua i lloguer del local on es realitza el projecte. En la taula següent es detallen les despeses indirectes.

Despeses Indirectes	**Factures**	**Cost(€)**	**Import(€)**
Llum	2	164	328
Aigua	2	12	24
Lloguer	4	370	1.480
Total			**1.832**

Taula 4: Despeses inicials indirectes.

6.5 Despeses totals

Finalment, sumem les despeses en recursos humans i materials, i obtenim un total al que després apliquem l'IVA actual, del 21%.

Concepte	**Total (€)**
Recursos humans	13.472
Recursos materials	193,62
Recursos software	0
Despeses indirectes	1.832
Total	**15.497,6**
Total+IVA	**18.752,1**

Taula 5: Despeses inicials totals.

6.6 Viabilitat econòmica

Pel que fa a la viabilitat del projecte, la major part de les despeses provenen dels recursos humans, i per tant, de les hores dedicades a la realització de cadascuna de les etapes i del projecte en general. És per això que s'han de controlar de forma continua les hores invertides en cada etapa, per tal de no sobrepassar el límit, i alhora, adequar les funcionalitats implementades segons a la disponibilitat econòmica del pressupost.

Les despeses de recursos materials, però, són fixes des de l'inici del projecte, i per tant, no s'haurien de veure alterades de cap forma, perquè el pressupost inicial les hauria de cobrir.

7 Sostenibilitat del projecte

En els apartats següents es detallen els aspectes de la sostenibilitat del projecte.

7.1 Impacte social

L'impacte social del projecte, si bé no entra dins dels objectius d'aquest, en el context on s'emmarca fa referència als estudiants i professors, en el fet que podria ajudar a aprendre o ensenyar sobre disseny i conceptes de sistemes operatius i aplicacions per dispositius ARM. Alhora també pot tenir impacte social en desenvolupadors i programadors de dispositius ARM, ja sigui per aprofundir en conceptes o per servir com introducció a l'entorn ARM per facilitar escometre projectes majors.

7.2 Impacte ambiental

Respecte a l'impacte ambiental del projecte, els recursos materials consumits o utilitzats són força reduïts, i no són fruit del producte que crea el projecte, finalitzat el qual, no es genera cap impacte ambiental de cost a considerar.

Finalment, pel que fa a la rendició de comptes, degut al caire del projecte, aquest tampoc té cap efecte, ni positiu ni negatiu, respecte l'empremta ecològica.

8 Entorn de desenvolupament, eines i recursos

Aquesta secció descriu les eines i els diferents recursos utilitzats durant la realització del projecte.

8.1 Hardware

Raspberry Pi

Per a tal de realitzar les proves del SO en el entorn d'execució necessitarem una Raspberry Pi. El model, A o B, és indiferent, el segon té el doble de memòria ram i connexió ethernet, però el SO no en farà ús. En la Figura 8 es poden observar els diversos components del dispositiu.

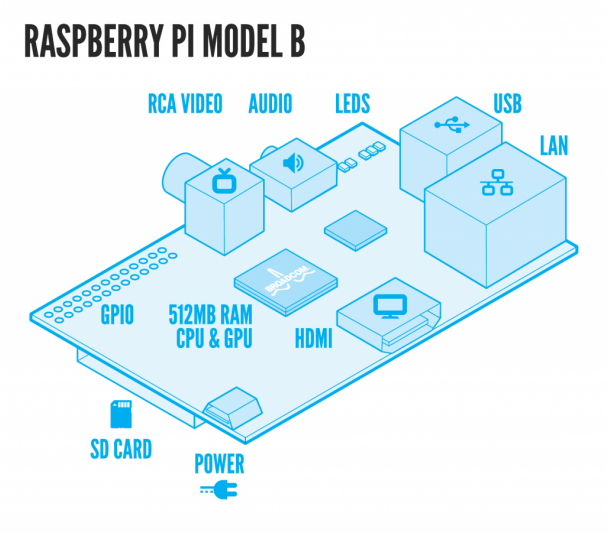

Figura 8: Components de la Raspberry Pi

Adaptador USB-Sèrie

Per a realitzar el sistema d'E/S del SO i poder-lo provar en la Raspberry Pi es necessari un adaptador que ens permeti accedir des d'un ordinador a la interfície UART que es troba situada en els pins GPIO de la Raspberry Pi. Per aquest fi podem utilitzar un dispositiu com el de la Figura 9.

No s'opta per una connexió d'USB a USB perquè per a fer-ne ús **s'hauria d'implementar el codi per donar-ne suport** en el kernel, i degut a la seva complexitat s'opta per usar la interfície serie. Alhora, cal tenir en compte que els ports USB de la Raspberry Pi actuen com a *Host*, igual que els d'un ordinador, fet que complica la seva comunicació.

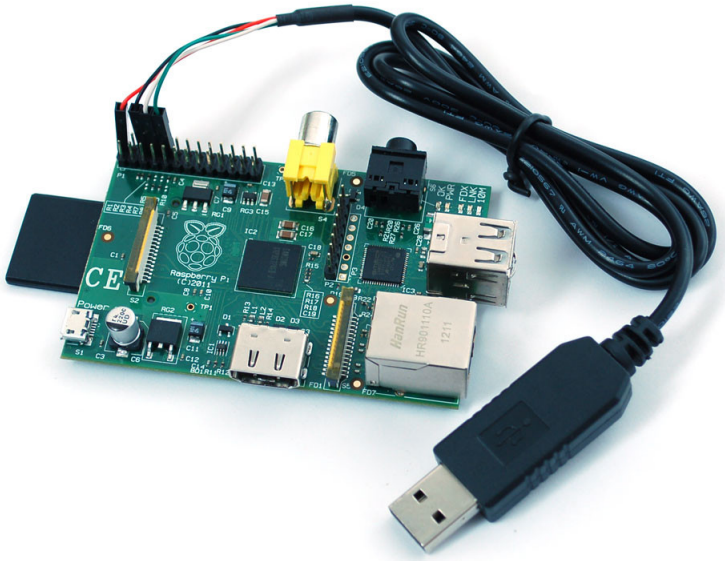

Figura 9: Adaptador USB-Sèrie

8.2 Software

Eines de compilació

Per tal de realitzar la compilació del codi del SO per l'arquitectura ARM i concretament per la Raspberry Pi, s'utilitzen les eines de compilació proveïdes en els repositoris oficials de la Raspberry Pi on es pot trobar el toolchain de GNU.

Linaro [18] és una organització sense ànim de lucre dedicada al desenvolupament d'eines i software open source de Linux per a plataformes ARM, proveint en aquesta ocasió el toolchain de GNU [9]. El fet d'utilitzar aquest toolchain permetrà fer cross-compiling per accelerar el procés de compilació i no haver de realitzar-lo en la Raspberry Pi.

Aquest projecte utilitza les eines següents per a la construcció del executable del SO: el compilador Gcc, muntador Ld, manipulació de fitxers objecte amb Objcopy i com a debugger Gdb.

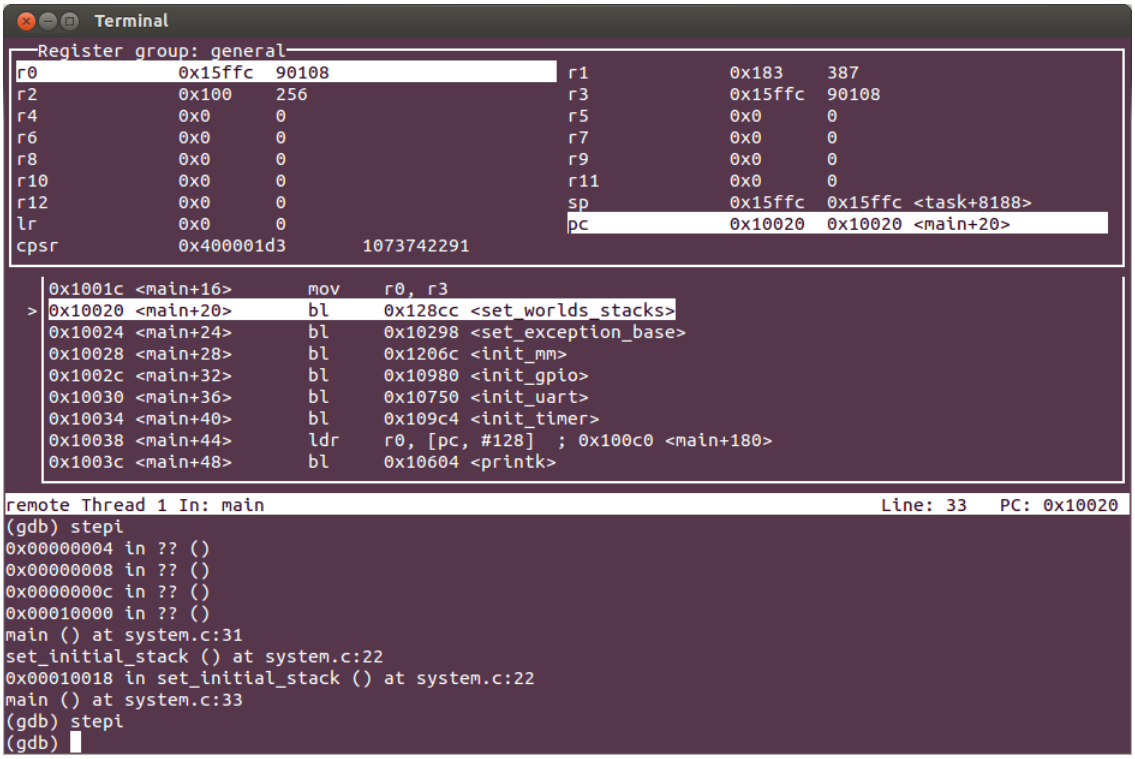

Figura 10: Simulació de ZeOS amb Qemu i el debugger Gdb

Qemu

Figura 11: Logo de Qemu

Qemu és un software open source que serveix d'emulador i màquina virtual, i permet executar sistemes operatius en arquitectures específiques emulant diverses arquitectures de processadors, entre d'elles ARM i en concret l'arquitectura de la CPU ARM1176JZF-S que trobem a la Raspberry Pi.

Aquest emulador, **el podem usar en conjunció amb el programa Gdb** per tenir un control a baix nivell de l'execució del SO: pausar l'execució, avançar instrucció a instrucció, i inspeccionar i modificar els registres de la CPU i la memòria, entre d'altres funcionalitats.

Tot i l'àmplia, varietat de funcionalitats que proporciona Gdb junt amb Qemu, se'n troben a faltar d'altres, com és la visualització de la configuració del sistema de protecció

de memòria, que es troben en programes similars com el software Bochs. Bochs és un software emulador de SO que incorpora un debugger propi, tot i que només emula arquitectura Intel IA-32, essent aquest l'emulador principal usat en les assignatures de la FIB on es treballa amb ZeOS.

Minicom

Minicom és el software de terminal de comandes, que s'utilitzarà per accedir a la interfície sèrie des del ordinador, mostrant aquest programa la informació que rep via sèrie i permetent enviar informació al dispositiu final.

Software de suport

Figura 12: Logo de Git

Com a software de suport en el desenvolupament del projecte s'utilitzarà el sistema de control de versions GIT per tenir un control més acurat del desenvolupament i s'utilitzarà el IDE Eclipse per a la programació del software del SO

9 Arquitectura de la Raspberry Pi

En aquest capítol es descriuen les característiques rellevants de l'arquitectura que inclou la Raspberry Pi, tant el processador en sí com els components del SoC. En aquest capítol es pretén donar uns coneixements bàsics de l'arquitectura, amb l'objectiu d'introduir el lector en el context amb el SO, i alhora ressaltar les característiques importants de l'arquitectura, per a donar un punt de vista prou ampli sobre aquesta.

9.1 Caracterització del processador ARM

El processador amb el que compta la Raspberry Pi és el model ARM1176JZF-S, processador d'ARM que forma part de la família ARM11 i que implementa l'arquitectura ARMv6.

Tota la informació respecte a aquest processador es pot trobar en el document tècnic de referència del model de CPU [17] i el document de referència de l'arquitectura [16] que, tot i ser ARMv6 s'inclou en el document de ARMv7.

9.1.1 Modes i Mons d'execució

El model d'execució d'instruccions es pot resumir en la Figura 13. Com podem veure, existeix la distinció usual entre mode usuari i mode privilegiat, però alhora s'introdueix la variable de seguretat que identifica els dos mons isolats, segur i no segur, i per a comunicar-los trobem el mode Monitor.

Mons d'execució

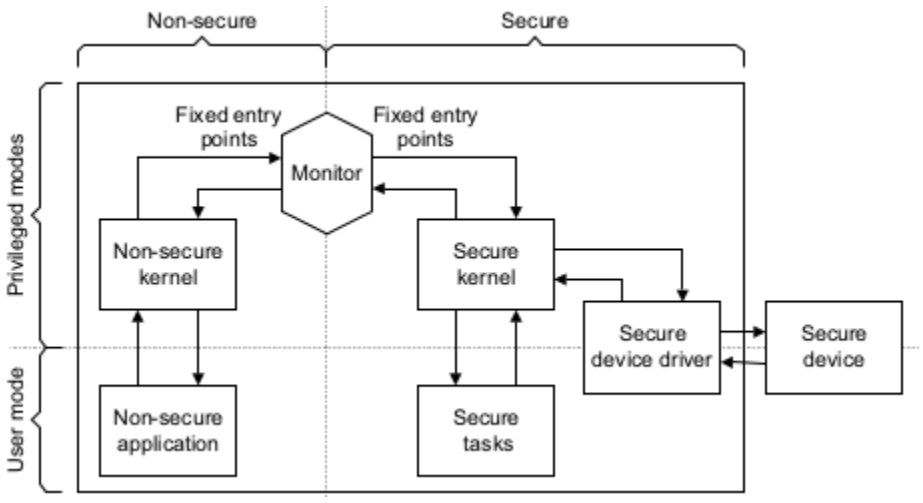

Figura 13: Mons d'execució. ARM Technical Reference Manual [17]

L'idea principal que busca aquest model és mantenir isolat la majoria del codi del kernel que s'executa en món no segur de la part reduïda, i per tant més fàcil de mantenir, del kernel que s'executa en món segur i interacciona amb dispositius segurs. El món segur

també pot tenir interacció directa amb dispositius.

Quan una aplicació d'usuari requereix d'execució en el món segur, realitza una petició al kernel no segur, que en realitza una altra al mode Monitor per a transferir l'execució al món segur. El codi del mode Monitor s'encarrega de fer el canvi de context entre els dos mons i amagar la configuració del coprocessador. La seguretat del sistema depèn de la seguretat del codi que s'executa en mode Monitor i del codi d'arrencada.

Modes d'execució

A més de la separació entre mons, també trobem la distinció entre modes d'execució. El processador disposa de 8 modes d'execució diferents, com podem veure en la Figura 14. La majoria de modes s'executen en mode privilegiat, excepte usuari, i tots existeixen tant en el món segur com no segur, excepte el mode Monitor que sempre requereix d'execució en el món segur.

- **User:** Mode no privilegiat d'execució de codi.
- **FIQ:** Mode dedicat a l'atenció d'interrupcions ràpides.
- **IRQ:** Mode dedicat a l'atenció d'interrupcions generals.
- **Supervisor:** Mode privilegiat del SO, encarregat d'atendre les interrupcions software.
- **Abort:** Mode en el qual entra el processador després d'un Prefetch Abort d'instruccions o codi.
- **Undefined:** Mode en el qual entra el processador en executar una instrucció indefinida.
- **System:** Mode privilegiat del SO.
- **Secure Monitor:** Mode de transició entre el món segur i el no segur.

Modes	Mode type	State of core	
		NS bit = 1	NS bit = 0
User	User	Non-secure	Secure
FIQ	privileged	Non-secure	Secure
IRQ	privileged	Non-secure	Secure
Supervisor	privileged	Non-secure	Secure
Abort	privileged	Non-secure	Secure
Undefined	privileged	Non-secure	Secure
System	privileged	Non-secure	Secure
Secure Monitor	privileged	Secure	Secure

Figura 14: Modes d'execució. ARM Technical Reference Manual [17]

9.1.2 Registres

Registres Generals

El processador compta amb 16 registres com podem observar en la Figura 15, dels quals 13 són de propòsit general, els 3 restants són els següents:

- **Stack Pointer (sp):** Correspon al registre r13, serveix la funció d'apuntador de la pila.
- **Link Register (lr):** Correspon al registre r14, conté l'adreça de retorn des d'on ha estat cridada una funció.
- **Program Counter (pc):** Correspon al registre r15, conté l'adreça de la propera instrucció a executar.

La majoria de registres són compartits entre els diferents modes d'execució; en realitzar un canvi de mode el contingut del registre es manté. Tot i així, existeixen còpies úniques d'alguns registres en cada mode: els registres sp i lr tenen una còpia en cada mode per a facilitar el canvi de context, i de forma excepcional, en mode FIQ els r8-12 també tenen una còpia per agilitzar el tractament ràpid d'interrupcions.

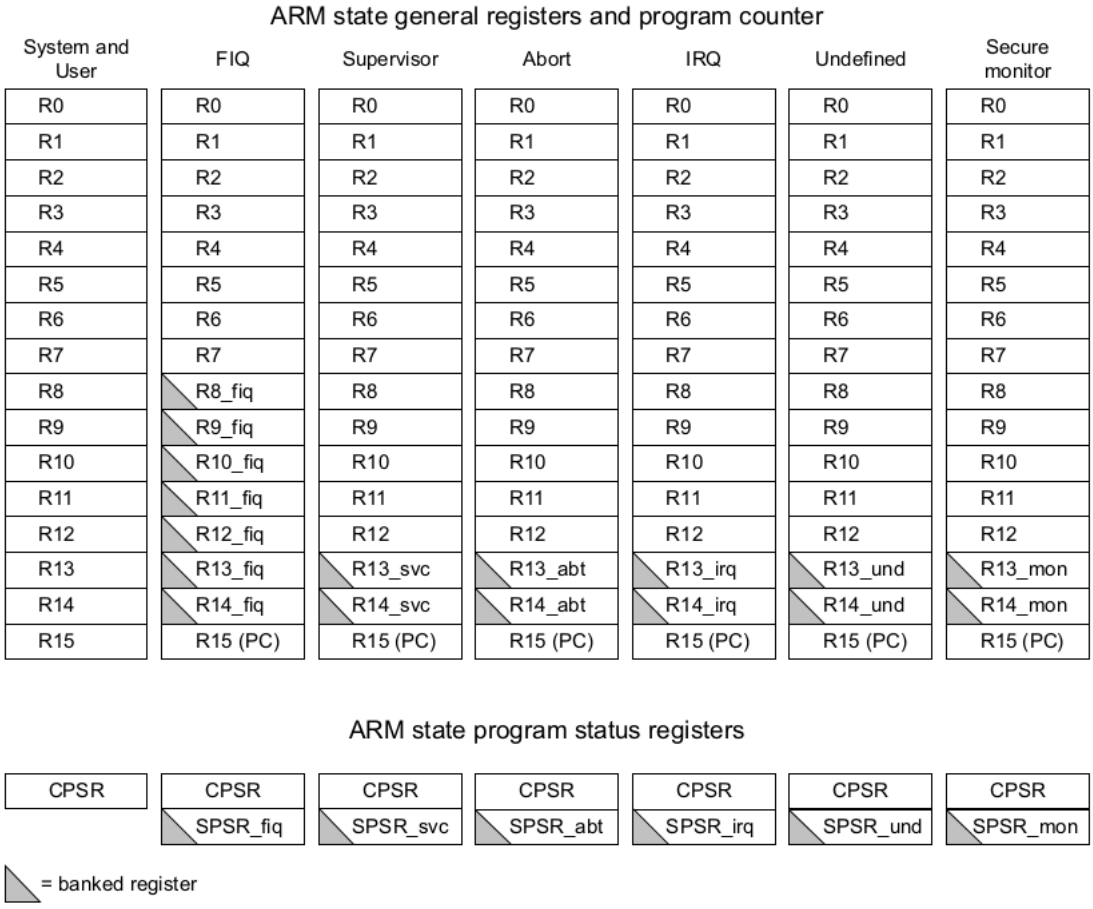

Figura 15: Registres de propòsit general. ARM Technical Reference Manual [17]

Registre CPSR

Finalment, tenim un únic registre anomenat cpsr que indica l'estat del processador. Aquest registre serveix diverses funcions, com podem veure en la Figura 16, entre les més rellevants: indicar l'estat de l'última operació de l'ALU, configurar les interrupcions habilitades, modificar l'endianness del sistema i canviar el mode d'execució del sistema.

Aquest registre és accessible des de tots els modes, i alhora es disposa en cada mode d'un registre que actua com a còpia, que és el spsr. En realitzar un canvi de mode, el cpsr del mode actual es copia al registre spsr del mode destí per agilitzar el canvi de context.

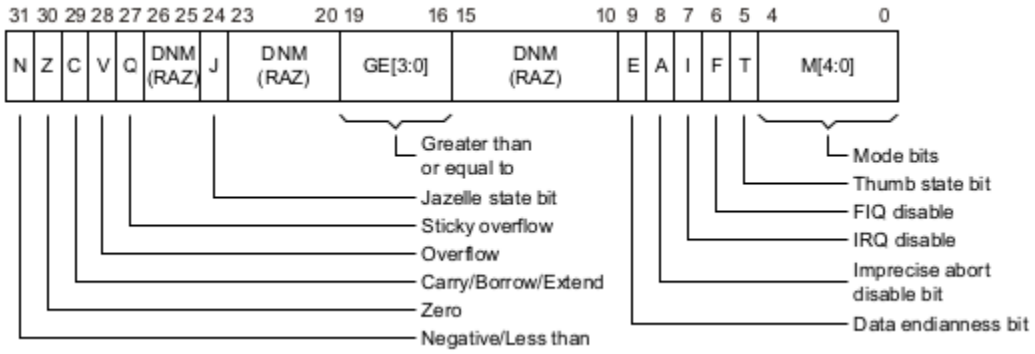

Figura 16: Registre Program Status. ARM Technical Reference Manual [17]

9.1.3 Convenció de crida de rutines

ARM defineix una convenció [15] per a la crida de rutines i l'ús dels registres del sistema per a indicar com han de ser utilitzats per compiladors i programadors.

La convenció defineix l'ús de la pila com a *'Full descending stack'*, és a dir, una pila que augmenta de mida a mesura que decrementa la seva adreça.

Pel que fa als registres, els registres des del **r0-r3** s'utilitzen com a argument d'una funció o resultat; del **r4-r11** s'usen com variables; el **r12** s'usa com a *'Intra-Procedure-call scratch register'*; i els tres restants, **sp**, **lr** i **pc**, tal com hem definit en el capítol de registres.

Quan una rutina realitza una crida a una altra rutina, les dues han de seguir un procediment per evitar perdre informació en ús quan es realitza la crida. ARM segueix el procediment següent.

- La funció que realitza la crida:
  - **Pot salvar els registres r0-r3** en la pila, si hi tenia informació. En aquests registres hi posarà els arguments de la funció a cridar.
  - **Ha de salvar el registre lr** en la pila, perquè la funció a cridar pot haver-ne de cridar d'altres.
  - **Pot salvar el registre r12** en la pila, si l'utilitza i té informació que vol conservar.
  - **No ha de salvar la resta de registres**. La convenció assegura que en retornar conserven el seu valor.
- La funció que és cridada:
  - **Ha de salvar els registres r4-r11 i sp**, en la pila, **si els utilitza**.
  - Si realitza alguna crida **ha de salvar els registres indicats** anteriorment.

En una rutina, el valor de la **sp** és decrementat en iniciar-la i incrementat en sortir-ne, amb instruccions *push* i *pop* respectivament, segons el nombre de registres dels que faci ús la funció. Aquest fet és important perquè **hi ha funcions que necessiten saber el valor de la pila** per una determinada adreça de retorn, sense la modificació dels registres salvats, com és el cas de les funcions *fork, clone i també task_switch*, que veurem més endavant. En ARM, l'única forma per a poder indicar el valor de la pila en aquestes funcions és passant-lo com a paràmetre de la funció; en canvi en Intel, dins d'una funció es pot accedir al registre **ebp** que proporciona aquesta informació.

9.1.4 Coprocessador

La configuració de les funcionalitats del processador no només es realitza amb el registre cpsr, majoritàriament es realitza amb els registres del coprocessador CP15. Entre les funcionalitats que controla trobem:

- Configuració general del sistema
- Configuració de la cache
- Configuració de memòria TCM
- Configuració de la MMU
- Control de DMA
- Monitors de rendiment

El processador disposa de més de 100 registres diferents de configuració del coprocessador i són accedits amb les instruccions MCR i MRC, per a l'escriptura i lectura respectivament. Les dues instruccions compten amb un registre font o de destí i quatre operants, dos adreçadors de registre de 4 bits i dos operands de 3 bits, els quals serveixen per organitzar i agrupar els diferents registres de configuració. A continuació es mostra la construcció de les instruccions:

```
MCR{cond} P15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
MRC{cond} P15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

Entre els registres que configura el coprocessador, uns dels registres destacables són el registre Control *(ó c1, 0, c0, 0)*, i el registre Secure Configuration *(ó c1, 0, c1, 0)*. Aquests registres habiliten o deshabiliten components com la MMU, les caches d'instruccions i dades, i d'altres característiques generals, com el **bit NS** *(Non-Secure)*. Aquest bit especifica el món, segur o no segur, en tots els modes d'execució excepte el mode Secure Monitor, com es destacava en el capítol dels modes d'execució.

9.1.5 Gestió de la memòria

L'adreçament de la memòria per defecte es realitza en mode real o transparent, per tant sense la traducció d'adreces ni suport de cache d'instruccions, ni dades. El processador incorpora una MMU força versàtil i configurable per a realitzar la traducció de les adreces virtuals, les que processa la MMU i utilitzen els processos, a adreces físiques, les que la MMU dona com a traducció de les anteriors.

La traducció de adreces es realitza generalment amb taules de dos nivells (taula de directori i taula de pàgines, o primer i segon nivell).

Seccions i pàgines

Els tipus de divisió de l'espai d'adreces que realitza la MMU, es pot categoritzar en els apartats següents segons la configuració de la MMU:

- **Supersecció:** Blocs de **16MB** de dades, on només s'utilitza el primer nivell de taules de traducció.
- **Secció:** Blocs de **1MB** de dades, com amb les superseccions, on només s'utilitza el primer nivell de la taula de traducció.
- **Pàgines grans:** Pàgines de **64KB**, que utilitzen els dos nivells de taules de traducció.
- **Pàgines petites:** Pàgines de **4KB**, que utilitzen els dos nivells de taules de traducció.

El tipus de traducció d'adreces que es realitza es determina amb el contingut de les taules de traducció. A continuació es detalla com es determina el tipus de traducció, segons el que s'observa gràficament en la Figura 17.

Seccions Si en el contingut del primer nivell observem un *0b10*, la divisió és de seccions. Aleshores si el bit 18 és 0, la traducció es tracta d'una secció, i si és 1, d'una supersecció.

Pàgines En el cas que els bits de menor pes del primer nivell siguin *0b01*, la traducció és de pàgines i cal consultar el segon nivell de taules de traducció. En el segon nivell, si observem el valor *0b01*, es tracta d'una pàgina gran, i si observem *0b1X*, la traducció és de pàgines petites.

Traducció d'adreça virtual a adreça física

L'adreça virtual es divideix en dues o tres parts: Índex de la taula de directori, índex de la taula de pàgines (opcional) i l'offset. Així doncs, per a realitzar la traducció, representada gràficament en la Figura 17, seguirem els passos següents:

1. Primerament, s'indexa sobre la taula de directoris amb el corresponent índex, obtenint del contingut l'indicador de tipus de traducció: Supersecció, secció o pàgines, i la base de l'adreça final o de la taula de pàgines. Si la traducció resulta en una supersecció o una secció, saltem al tercer pas.

2. A continuació, amb la base de la taula de pàgines i l'índex d'aquesta obtenim el contingut de la taula de pàgines que ens permetrà determinar el tipus de pàgina: gran o petita, i la base de l'adreça final.

3. Finalment, agrupant la base de l'adreça final i l'offset, obtenim l'adreça física que resulta de l'adreça virtual proporcionada.

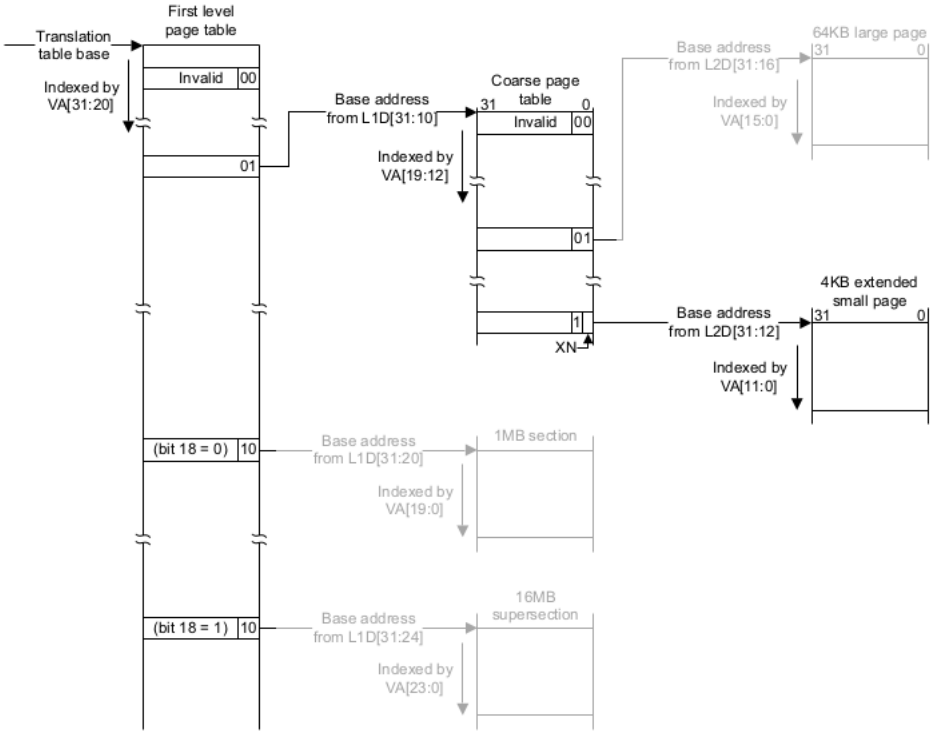

Figura 17: Esquema de traducció de memòria virtual a memòria física. ARM Technical Reference Manual [17]

9.1.6 Gestió de les interrupcions

Les interrupcions o excepcions **trenquen el flux d'execució del codi** d'un SO per a atendre la petició que ha generat una determinada font interrupció, que pot ser tant el propi codi del processador com un dispositiu extern. Les interrupcions, les podem classificar en tres grups diferents: Excepcions, Interrupcions hardware i Interrupcions software, detallats a continuació.

Excepcions Les excepcions són interrupcions síncrones i generades per la CPU després de l'execució d'una instrucció que ha provocat un error. El processador canvia el mode d'execució al requerit per atendre el problema segons la font d'aquest: si l'error és provocat per l'execució d'una instrucció no definida, aquest canviarà a mode Undefined, i si és produït per un error en l'obtenció d'una dada, al mode Abort.

Interrupcions Hardware Les interrupcions hardware són asíncrones i generades per dispositius hardware. Com en les excepcions, el processador canvia el mode d'execució per atendre la interrupció, i en aquest cas els modes són IRQ i FIQ. L'única distinció entre elles és la priorització de les FIQ.

Interrupcions Software Les interrupcions software, que són síncrones i produïdes per instruccions assemblador especifiques, com les instruccions *SVC* (supervisor call), i *SMC* (secure monitor call), serveixen per a implementar mecanismes d'entrada al sistema, que canvien el mode d'execució a supervisor o Secure Monitor com són les syscalls d'un sistema.

Sistema d'atenció de les interrupcions

Les interrupcions tenen un mateix sistema per a la seva atenció, que és el vector d'excepcions, el qual pot estar situat a diferents direccions del sistema: *0x0*, *0xFFFF0000* o una adreça concreta. En la direcció escollida se situa la taula d'excepcions que conté una instrucció de salt a la funció de d'atenció de la interrupció. Mentre en l'arquitectura Intel, en canvi, aquesta taula indica les direccions on estan les funcions d'atenció. En l'arquitectura ARM, la taula té aquest aspecte:

Excepció/Interrupció	**Offset**
Reset	0x00
Instrucció Indefinida	0x04
Interrupció software	0x08
Prefetch Abort	0x0c
Data Abort	0x10
Reservada	0x14
Petició d'Interrupció	0x18
Petició d'Interrupció ràpida	0x1c

Taula 6: Vector d'excepció

Punts d'entrada compartits

Les interrupcions software generades per les instruccions *SVC* i *SMC* tenen el **mateix punt d'entrada** i és el handler l'encarregat de gestionar quina interrupció servir. El mateix problema es dona en les interrupcions hardware, que també tenen un mateix punt d'entrada, si no tenim en compte les interrupcions FIQ, i aquest és compartit per tots els

diferents dispositius dels quals disposi el processador. Aquest fet pot ralentitzar la gestió de les interrupcions si s'ha de fer una comprovació per conèixer qui ha generat una petició. Per aquest motiu el processador incorpora un controlador **VIC** (Vectored Interrupt Controller), amb l'objectiu d'aplicar el mateix mecanisme del vector d'excepcions però per a les interrupcions hardware. Com veurem més endavant, però, aquest *controller* no està implementat per al dispositiu Raspberry Pi.

9.2 Caracterització del System-on-Chip Broadcom BCM2835

El System-on-Chip de Broadcom BCM2835 és el cor de la Raspberry Pi. Aquest element es compon principalment del processador ARM, introduït en el capítol anterior, i la GPU VideoCore IV Multimedia, responsable de les capacitats gràfiques del dispositiu, com ara la reproducció de vídeos en alta definició.

A més a més, un SoC no només es compon de un processador, sinó que normalment inclou altres components, com la memòria, perifèrics o timers, essent aquest també el cas del SoC de la Raspberry Pi.

A continuació s'introdueixen els aspectes més rellevants d'aquest SoC.

9.2.1 Sistema de boot

El **sistema de boot** (o sistema d'arrencada) del SoC, i per extensió de la Raspberry Pi, es força **singular i s'allunya de sistemes més comuns**, com els que ens podríem trobar en un ordinador, amb un Master Boot Record (MBR), o d'altres força comuns per a dispositius mòbils, com és l'U-Boot.

L'aspecte més característic i sorprenent és el fet que la **GPU és l'element principal d'aquest sistema** i la CPU juga un paper secundari. El procediment d'arrencada segueix els passos següents:

1. Primer de tot, la CPU es manté apagada i un core RISC, que forma part de la GPU, inicia l'execució del bootloader emmagatzemat a la memòria ROM del SoC.

2. El bootloader llegeix el contingut de la primera partició de la targeta SD, en **format fat32**, i carrega el segon bootloader del fitxer *bootcode.bin*.

3. Aquest segon bootloader s'encarrega de llegir el firmware de la GPU del fitxer *start.elf*, que carrega a continuació. S'utilitza també el fitxer *fixup.dat* per a configurar la divisió de la SDRAM entre la GPU i la CPU.

4. Un cop arribats en aquest punt, la GPU ja està preparada per a iniciar l'execució del SO del processador, però primer llegeix els fitxers següents:

    - *config.txt:* En aquest fitxer s'especifiquen diverses configuracions de tot el SoC, com són les característiques de la sortida HDMI, la freqüència del processador, la divisió de la memòria entre la GPU i la CPU, la configuració de perifèrics com la UART, i l'adreça on haurà de carregar la imatge del SO.
    - *cmdline.txt:* Aquest fitxer indica paràmetres que es proveiran al SO un cop comenci l'execució.

5. Finalment, la GPU carrega el fitxer *kernel.img*, que conté el contingut del SO, a l'adreça indicada en el fitxer *config.txt* i allibera el senyal de reset a la CPU, iniciant la seva execució.

9.2.2 Bus de memòria

La distribució i l'accés a la memòria també és particular, com podem veure en la Figura 18, que és la que ve a continuació:

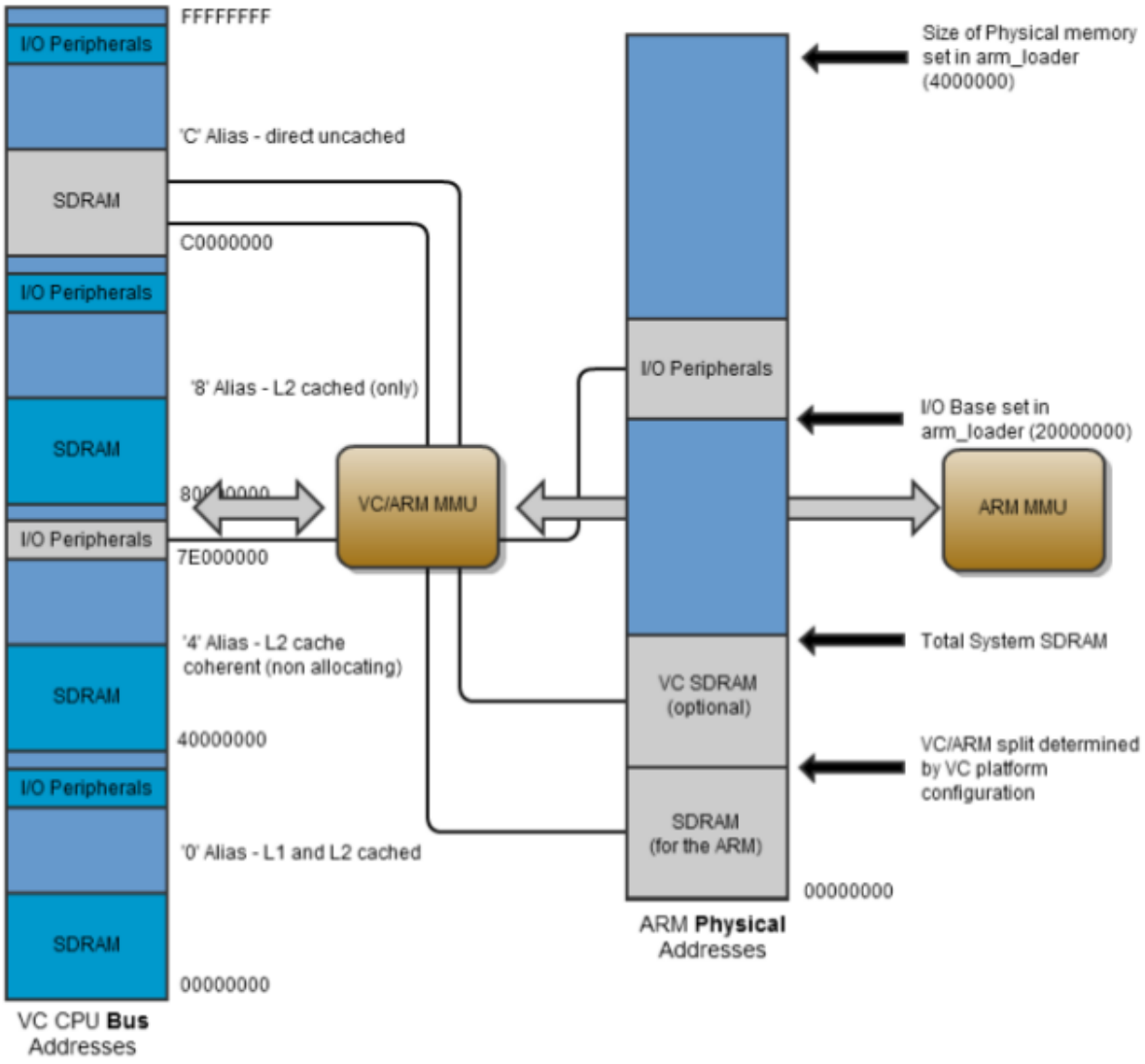

Figura 18: Esquema de la distribució de la memòria en el SoC

A la esquerra, es pot veure representat el bus de la GPU, on hi podem trobar la SDRAM, i la configuració dels diversos perifèrics del SoC. Aquest bus, però, **no és accessible directament per la CPU ARM**, sinó que ho és mitjançant una MMU. Aquesta MMU, sense accés directe a la seva configuració, realitza la traducció de les adreces físiques del dispositiu ARM, representades al centre de la imatge 18, a les del bus de la GPU.

Com podem veure en la imatge 18, aquesta primera MMU relaciona una zona de la SDRAM, amb grandària especificada en els fitxers d'arrencada, i una zona de perifèrics

(0x73000000 en la GPU a 0x20000000 en la CPU) de l'espai d'adreces de la GPU a l'espai d'adreces de la CPU. Per tant, la CPU no té accés directe a la resta de recursos del SoC.

Finalment, a la dreta de la imatge 18 hi ha representada la MMU del processador, que és l'encarregada de fer la traducció entre l'espai d'adreces físic del processador i el virtual, que veurem més endavant.

9.2.3 Sistema d'interrupcions

El processador ARM en aquest SoC té dues fonts d'interrupció: les provinents dels perifèrics de la GPU, i les provinents dels perifèrics del propi ARM. La gestió de les interrupcions a nivell del SoC de la Raspberry Pi tampoc es gestionen amb un controlador VIC, com havíem vist amb el processador ARM. Per tant, s'utilitza un sistema més rudimentari per filtrar les interrupcions.

Registres	**Offset**
IRQ basic pending	0x200
IRQ pending 1	0x204
IRQ pending 2	0x208
FIQ control	0x20c
Enable IRQ 1	0x210
Enable IRQ 2	0x214
Enable Basic IRQs	0x218
Disable IRQ 1	0x21c
Disable IRQ 2	0x220
Disable Basic IRQs	0x224

Taula 7: Registres de control de les interrupcions del SoC

La gestió de les interrupcions es fa mitjançant uns registres, mostrats en la Taula 7, localitzats en l'adreça 0x2000B000 en la CPU ARM. Aquests registres permeten habilitar i deshabilitar fonts d'interrupcions, i també consultar si hi ha una interrupció pendent de ser tractada.

Per agilitzar la velocitat de trobar quina interrupció ha generat la petició, el registre *IRQ basic pending* agrupa els dos registres següents d'IRQ pendents en dos bits, junt amb les fonts d'interrupció de perifèrics de la CPU mostrats a la Taula 8.

Les interrupcions FIQ tenen un tractament diferenciat; s'utilitza el registre *FIQ control* per tal d'especificar quina font d'interrupció serà conduïda cap a la interrupció FIQ. Així, en la rutina d'atenció de la interrupció FIQ, no cal realitzar cap control extra per filtrar les interrupcions.

GPU	Índex	CPU	Índex
Aux int	28	ARM timer	0
I2c spi	43	ARM Mailbox	1
Pwa0	45	ARM Doorbell 0	2
Pwa1	45	ARM Doorbell 1	3
Smi	48	GPU Halted 0	4
Gpio int 0	49	GPU Halted 1	5
Gpio int 1	50	Illegal Access 1	6
Gpio int 2	51	Illegal Access 0	7
Gpio int 3	52		
I2c int	53		
Spi int	54		
Pcm int	55		
Uart int	57		

Taula 8: Índex de les fonts d'interrupció dels perifèrics de la GPU i CPU

En aquesta taula es mostren les fonts d'interrupció que s'indiquen en el datasheet de Broadcom [2]. Les interrupcions de la GPU es codifiquen en els respectius registres 1 i 2 de la Taula 7, tot i que la majoria no estan documentades. Les interrupcions de la CPU es codifiquen en els registres basic.

Per últim, la neteja del senyal que activa la interrupció s'ha de realitzar a nivell de perifèric. Per tant, escriure el bit del registre d'interrupcions pendents no neteja el senyal.

9.2.4 Gestió dels dispositius perifèrics

El SoC de la Raspberry Pi incorpora un nombre important de perifèrics per ampliar les seves capacitats, entre els quals podem trobar:

- Comptadors (o Timers)
- GPIO
- USB
- 2 UARTs
- 3 SPI
- I2C
- PWM

A continuació, es destaquen i se'n fa un resum de les característiques dels perifèrics que s'utilitzen en el SO. Per a més informació sobre els esmentats i la resta dels perifèrics es pot consultar el document de referència **BCM2835 ARM Peripherals [2]**, que alhora és el principal document sobre aquest SoC.

Perifèric GPIO

Els pins del perifèric GPIO (General Propouse Input/Output, en anglès) són pins configurables que podem trobar en molts dispositius encastats, com és el cas de la Raspberry Pi, en un **total de 54**. La majoria d'aquests pins són accessibles directament per a connectar-hi el dispositiu a fer servir. A continuació, en la Figura 19 es poden observar els connectors directament accessibles en la Raspberry Pi.

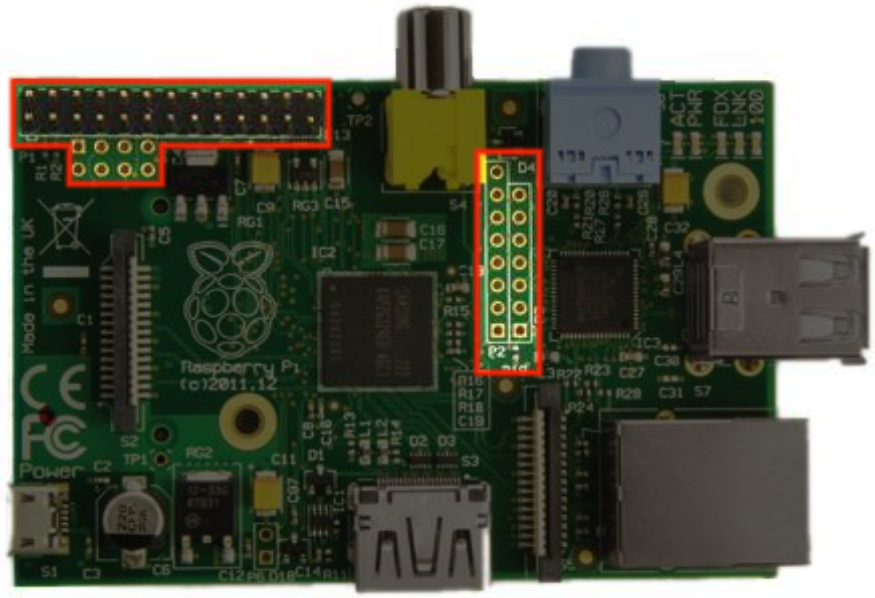

Figura 19: Disposició dels connectors GPIO en la Raspberry Pi

En la Raspberry Pi, els GPIO poden ser configurats com **entrada o sortida, i també, per realitzar fins a sis funcions diferents**, les quals permeten la utilització d'un mateix pin per diferents perifèrics del SoC. Alhora, el perifèric GPIO també permet utilitzar els pins com a font d'interrupció, i fins hi tot permet configurar el nivell en el que es genera la interrupció (Falling/Rising) i el valor.

Pel que fa a les funcions, un exemple és el dels GPIO 14 i 15 que realitzen la funció de transmissió i recepció de la UART 0 i de la UART 1, segons si estan configurats amb la funció 0 o 5, respectivament. La llista de pins i funcionalitats és extensa, de manera que no s'inclou en aquest document, tot i que es pot trobar en la documentació del SoC [2].

Perifèric UART

El perifèric UART habilita un mecanisme de comunicació senzilla i fàcil d'implementar, entre la Raspberry Pi i el dispositiu al qual es connecti.

La Raspberry Pi disposa de dues interfícies UART amb implementacions diferents, la UART 0 es basa la implementació 16500 i la UART 1 en la implementació 16C650. Tot i això, les característiques són similars, i la major diferència és que el segon model incorpora buffers més grans.

En el document tècnic que proporciona Broadcom [2] manca informació necessària per al funcionament de les UARTs, i és per això que s'ha de recórrer a documents d'altres fabricants [13]. De la que existeix més informació és de la UART 0, suficient per a configurar el baudrate de transmissió amb la fórmula següent:

$$baudrate = \frac{system\_clock\_freq}{(registre\_baudrate + 1) * 8}$$

El rellotge del **sistema funciona a 250MHz**. Per tant, si es volgués configurar el baudrate de transmissió a 115200, s'hauria d'indicar el registre del baudrate amb el valor 270.

Perifèric Timer

La Raspberry Pi inclou també perifèrics comptadors (o Timers) que permeten al dispositiu portar un control del temps d'execució. El SoC disposa de dos timers, un System Timer i un ARM Timer.

El System Timer és un comptador de 64 bits amb 4 registres comparadors que permeten generar una interrupció, en cas que el valor d'un d'aquest es correspongui amb el del comptador. Tot i que no s'indica en el document, aquest comptador és usat pel software de la GPU; així doncs, si s'utilitza, afecta el funcionament de la Raspberry Pi.

Per últim, el ARM Timer és un comptador basat en un SP804 modificat. Disposa de un comptador de 32 bits que es decrementa i genera una interrupció en arribar a 0, i un pre-scaler i un pre-divider per ajustar el nombre d'interrupcions generades. Aquest comptador, a diferencia de l'anterior, sí que és utilitzable des d'ARM. La fórmula del càlcul del pre-scaler és la següent:

$$freq\_dec = \frac{system\_clock\_freq}{prescale + 1}$$

Com en la UART, el rellotge del sistema funciona a 250MHz. Si volem que es generi una interrupció cada segon, hauríem d'usar un pre-scaler de 250.000.000 i un valor al comptador d'un per generar el decrement. Aquest valor de pre-scaler, però, no es pot especificar amb els bits dels que disposa el comptador, i per aconseguir aquesta freqüència d'interrupció s'ha de trobar una combinació adequada dels valors de comptador i de pre-scaler.

10 Implementació de ZeOSpi

Un cop vistes les característiques de la plataforma on es desenvolupa el SO, en aquest capítol veurem els detalls de la implementació d'aquest. Per tant, l'objectiu és revisar la implementació, per a donar tota la informació sobre les característiques rellevants de la mateixa, i diferenciar canvis respecte la implementació hardware del SoC o respecte del funcionament en una arquitectura Intel.

En aquest capítol s'anomena **ZeOSpi** al port de ZeOS per a la Raspberry Pi realitzat en aquest projecte.

10.1 Inicialització del sistema operatiu

La inicialització del SO cobreix des de l'arrencada de la Raspberry Pi fins que l'execució del codi arriba a l'espai d'usuari. Així doncs, l'objectiu d'aquesta inicialització és **configurar el hardware necessari**, com ara la MMU i les interrupcions, i inicialitzar les estructures de dades per a la gestió de les tasques del SO.

Construcció de la imatge

El fitxer (o imatge) del SO es construeix ajuntant, gràcies al **executable build**, els dos fitxers objecte de SO, *system* i *user*, que són el resultat de compilar el codi dels fitxers de cadascun d'aquests espais.

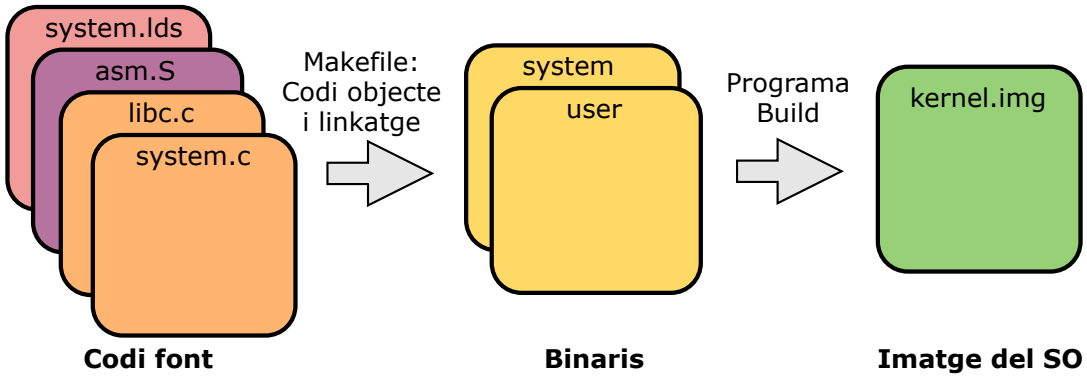

Figura 20: Procés de creació de l'imatge del SO

L'executable build deixa espai per a **3 enters al inici de la imatge** per a indicar informació sobre el propi fitxer, informació que el SO llegirà un cop iniciï l'execució. La informació indicada és una **instrucció de salt**, un **enter amb la mida** de les dades de **system** i un altre amb la mida de les dades de **user**. S'introdueix la instrucció de salt perquè es la primera instrucció que s'executarà, i de no ser per aquesta, el processador interpretaria les dades de les mides com a instruccions en assemblador. En l'imatge 21 es poden apreciar els diferents components de la imatge del SO.

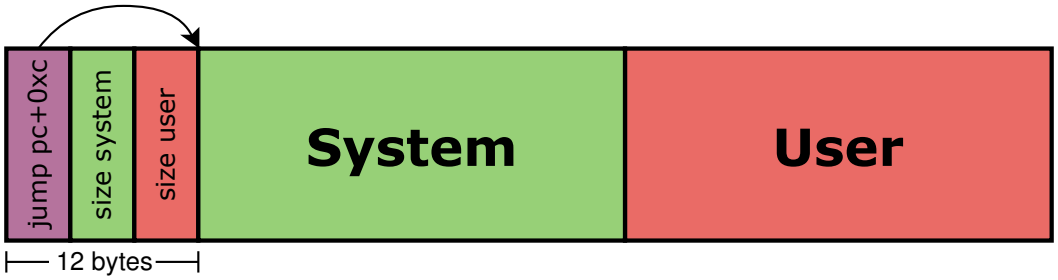

Figura 21: Contingut del fitxer kernel.img

Procés d'arrencada

Un cop obtenim la imatge del SO i volem executar-lo, l'hem de proporcionar a la màquina virtual, per tant, emulant l'execució, o bé a la Raspberry Pi, fent una execució directa. Aquests són els dos mètodes, o les dues plataformes, que s'han tingut en consideració en aquest projecte:

- **Màquina virtual Qemu:** Per a arrencar el SO, s'ha d'executar el programa Qemu, que emula màquines ARM: **qemu-system-arm**. A aquest programa se li ha d'indicar quina CPU es vol emular amb l'opció *-cpu arm1176*, i el lloc on trobarà la imatge del SO amb l'opció *-kernel kernel.img*.

  Normalment, a Qemu també li podem especificar quina plataforma ha d'emular, no només la CPU; emulant la màquina, disposem dels perifèrics d'aquesta i simulem la implementació del SoC, cosa que no es dóna si només especifiquem la CPU a emular. Malauradament, **la versió actual no permet emular la Raspberry Pi, només la CPU**. Per tant, Qemu **no permetrà emular cap dels perifèrics de la Raspberry Pi**, i per a fer les proves relacionades amb aquests components s'haurà de recórrer a l'execució directa.

- **Raspberry Pi:** Per a arrencar el SO en la Raspberry Pi, se segueix el procediment amb els requeriments de fitxers, descrit en el capítol anterior. Haurem de proveir la imatge amb el nom de *kernel.img* i indicar en el fitxer config.txt, la paraula clau *kernel_address* amb valor *0x10000*. Aquesta serà l'adreça on carregarà la imatge i s'iniciarà l'execució.

Inicialització de hardware i estructures

Un cop hem iniciat o bé l'emulació o bé l'execució, el SO entra en la fase d'inicialització. Aquesta fase es pot organitzar en els conjunts d'activitats següents:

1. Inicialització dels registres de la CPU: En iniciar, s'han de modificar l'adreça de la pila i la de retorn en el registre sp i lr, respectivament, en tots els mons d'execució.

2. Inicialització de les excepcions i interrupcions, sense habilitar les últimes.

3. Inicialització de les estructures de la memòria i la MMU.

4. Inicialització dels perifèrics GPIO, UART i Timer.
5. Inicialització d'estructures per l'execució multiprocés, la gestió de l'E/S i sincronització per semàfors.
6. Inicialització del planificador multiprocés i les tasques bàsiques; idle i task1.
7. Inicialització de l'espai d'usuari.
8. Habilitació de les interrupcions i canvi d'execució a mode i espai d'usuari.

Un cop arribat al últim punt, **l'execució es transfereix a l'usuari** i el SO està completament inicialitzat. Només s'accedirà a aquest per mitjà de les interrupcions o crides a sistema que realitzi el codi d'usuari. En els capítols a continuació s'entra en detall en cadascun dels sistemes llistats anteriorment.

10.2 Gestió de l'espai d'adreces en ZeOSpi

En aquest capítol es detalla el ús del sistema de protecció de memòria i la organització de l'espai d'adreces de ZeOSpi.

10.2.1 Sistema de protecció de memòria

Estructures

En ZeOSpi, la MMU està configurada per a utilitzar el **format de pàgines ARMv6 amb pàgines de 4KB de dades** cadascuna. Això implica que una adreça virtual es divideixi de la forma següent:

Figura 22: Divisió d'una adreça virtual

La primera part de l'adreça indica de quina entrada de la primera taula se'n utilitza el contingut com a base per a la segona. En aquesta segona taula s'utilitza el segon index. Finalment el contingut d'aquesta i el offset ens dóna l'adreça física final.

Cada procés independent té **una taula de primer nivell, i dues de segon nivell**. Per a utilitzar tot l'espai d'adreces, el procés hauria de tenir una taula de segon nivell per a cada entrada de la primera taula. En aquesta implementació, però, **només s'utilitzen les dues primeres entrades de la taula de primer nivell**; la primera per les taules de kernel i la segona per l'espai d'usuari. En la versió per a Intel només s'utilitzava una entrada de directori, car l'adreçament de la primera taula només comptava amb 10 bits en lloc de 12 bits, fent que l'espai d'usuari entrés en el primer directori.

Les estructures de la MMU tenen les dimensions següents:

- **Primer nivell:** fl_ptable[NR_TASKS][4096]
- **Segon nivell:** sl_ptable[NR_TASKS][2][256]

També implementa una taula de segon nivell i una pàgina, marcades per a direccionar-hi les entrades de la taula de primer i segon nivell no utilitzades. El motiu és poder tenir un millor control dels accessos a memòria no controlats, i poder-ho observar amb més facilitat, evitant que configuracions errònies modifiquin pàgines sense el permís necessari.

Finalment, s'implementa un ByteMap de 1024 entrades per a marcar les **pàgines físiques** utilitzades.

Inicialització

El document tècnic de referència del processador ARM [2] detalla, per a la inicialització de la MMU, els passos següents:

1. Programar els registres del coprocessador CP15.
2. Programar les taules de primer i segon, i estructures.
3. Deshabilitar i invalidar la cache d'instruccions.
4. Habilitar la MMU marcant el bit 0 del Registre de Control de CP15 *(c1, 0 , c0, 0)*.
5. Habilitar i invalidar la cache d'instruccions.

Entre els registres de CP15 a modificar, ens podem trobar amb el registre TTB0, on es configura quina **taula de primer nivell s'usa**, o TTBC on s'habilita la TLB. La majoria dels registres, però, funcionen correctament amb el valor per defecte.

El contingut de les taules de primer nivell configura el món d'execució, segur o no segur i l'adreça base de la taula de pàgines, entre altres menys rellevants.

El contingut de les taules de segon nivell configura força característiques relatives a les pàgines: permisos d'execució del contingut de la pàgina, d'entrada a la cache, de **lectura i escriptura**, i l'adreça de la pàgina.

10.2.2 Distribució de l'espai d'adreces

En l'esquema 23 es mostra la distribució del espai d'adreces de ZeOSpi, on es poden apreciar dos grans blocs.

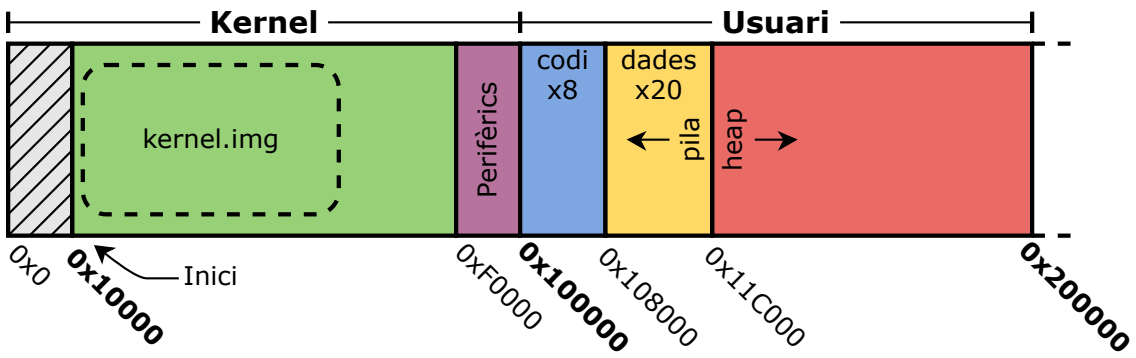

Figura 23: Espai d'adreces de ZeOSpi

Kernel

Aquesta zona comprèn des de la direcció 0x0 fins a la 0x100000, per tant, tot l'espai d'adreces que cobreix la **primera entrada d'un directori**. Només és accessible per als modes privilegiats.

- **Kernel start**: És la direcció on comença l'execució del SO, i per tant l'adreça base on s'ha carregat la imatge del SO *kernel.img*.
- **Perifèrics**: Aquesta zona, d'un total de 16 pàgines de l'espai d'adreces de Kernel, està reservada per a ser associada amb les adreces físiques del bus de memòria que corresponguin als diferents **perifèrics del SoC** que s'utilitzen. El motiu és poder accedir a la configuració dels seus registres.

Usuari

Aquesta zona comprèn des de la direcció 0x100000 fins a la 0x200000, per tant, tot l'espai d'adreces que cobreix la **segona entrada d'un directori**. És accessible per tots els modes. Per al mode no privilegiat, cada zona té diferents permisos de lectura i escriptura. Els modes privilegiats no tenen limitacions de permisos.

- **Codi**: Un total de 8 pàgines, contenent el codi d'usuari. Només és executable, no modificable.
- **Dades i stack**: Un total de 20 pàgines, contenent dades temporals i la pila del procés, que s'expandeix de forma inversa. No executable, però accessible i modificable.
- **Heap**: El heap és la zona utilitzada per a implementar el sistema de memòria dinàmica del SO. La zona de heap s'expandeix de forma incremental. Té els mateixos permisos que la zona Data.

Com es pot apreciar, aquests dos blocs no cobreixen tot l'espai d'adreces. La resta apunta a l'espai buit reservat, que s'ha comentat anteriorment.

10.3 Gestió de les interrupcions

En ZeOSpi les excepcions tenen la taula *exception_vector_table* com a punt d'entrada al kernel. En produir-se una excepció, l'execució saltarà a una de les vuit primeres posicions del vector i continuarà l'execució fent un salt a la rutina d'atenció de la interrupció indicada en aquesta taula.

```
1  exception_vector_table:
2    ldr   pc, [pc, #0x18] ;@ reset_handler
3    ldr   pc, [pc, #0x18] ;@ undefined_instruction_handler
4    ldr   pc, [pc, #0x18] ;@ software_interrupt_handler
5    ldr   pc, [pc, #0x18] ;@ prefetch_abort_handler
6    ldr   pc, [pc, #0x18] ;@ data_abort_handler
7    ldr   pc, [pc, #0x18] ;@ not_implemented
8    ldr   pc, [pc, #0x18] ;@ interrupt_request_handler
9    ldr   pc, [pc, #0x18] ;@ fast_interrupt_request_handler
10   .long reset_handler
11   .long undefined_instruction_handler
12   .long software_interrupt_handler
13   .long prefetch_abort_handler
14   .long data_abort_handler
15   .long not_implemented
16   .long interrupt_request_handler
17   .long fast_interrupt_request_handler
```

Aquesta taula es **similar a d'altres sistemes com Intel**. Cal notar que en el cas de ARM, el contingut de la posició és una instrucció, i no l'adreça a carregar al registre pc, com en el cas de Intel.

En general, les rutines d'atenció a les excepcions de ZeOSpi tenen com a propòsit **tres punts**: emmagatzemar el context d'execució en la pila, cridar a la funció que tracti l'excepció i, al retornar, restaurar el context que s'havia guardat i tornar al punt on s'havia produït l'excepció (incloent-hi el canvi de mode d'execució). El codi a continuació n'és un exemple:

```
1  undefined_instruction_handler:
2    stmfd sp!, {r0-r12,lr}
3    bl    undefined_instruction_routine
4    ldmfd sp!, {r0-r12,pc}^
```

Moltes excepcions també **requereixen modificar el punt de tornada a on s'havia produït l'excepció**. Per exemple, una interrupció IRQ ha de tornar a executar la instrucció que ha interromput, i per tant, al guardar l'adreça de retorn, aquesta s'ha de decrementar. En la taula següent s'inclouen els canvis necessaris per a cada excepció:

Registres	Contingut lr	Retorn
FIQ	pc+4	lr-4
IRQ	pc+4	lr-4
Supervisor	pc+4	lr
PAbort	pc+4	lr-4
DAbort	pc+8	lr-8
Undefined	pc+4	lr
Secure Monitor	pc+4	lr

Taula 9: Adreces de retorn de les excepcions

Finalment, en ZeOSpi, les **interrupcions hardware i software tenen un tractament específic**: han de guardar també el context (sp i lr) del mode usuari. Això es degut a que són punts d'entrada al kernel, en els quals es pot produir o un canvi de la tasca en execució, o la creació d'una nova. Per tant, s'ha de guardar aquesta informació en la pila del procés durant la funció d'atenció a l'excepció. En el codi següent es pot observar com es realitza:

```
1  cps   #0x1F ;@ system
2  mov   r4, sp
3  mov   r5, lr
4  cps   #0x13 ;@ supervisor
5  bic   r6, sp, #0xFF0
6  bic   r6, r6, #0xF
7  add   r6, r6, #0x1C
8  stmda r6, {r4,r5}
```

Al principi, es fa ús de la instrucció *cps* per a accedir als registres sp i lr d'usuari, canviant al mode sistema i retornant d'aquest, i a continuació, s'emmagatzemen en la pila del procés.

10.3.1 Interrupcions hardware

En ZeOSpi, només s'utilitza la interrupció hardware IRQ. En produir-se només dues interrupcions, no es justifica utilitzar també les FIQ. Tot i que les rutines tenen un mode propi, en ZeOSpi la funció de **la interrupció IRQ realitzarà un canvi a mode supervisor**. El motiu és la simplificació del codi, ja que aleshores, en el codi de kernel pots assumir que estàs en un únic mode i no en dos diferents.

La funció d'atenció de la interrupció IRQ és l'única que utilitza les instruccions assemblador *srs* i *rfe*. Són usades per a **guardar el spsr i restaurar-lo des del mode supervisor**. Mode en el que s'executarà la funció a realitzar per la interrupció. També deshabilita les interrupcions en mode supervisor, que només es donen si s'interromp l'execució de la tasca idle.

Finalment, com es detallava anteriorment, **no tenim un controlador VIC** per a poder

entrar directament en la rutina del perifèric que ha generat la interrupció, havent-se, doncs, de realitzar comprovacions amb els registres *Pending* de la taula 7 i prioritzar les interrupcions més importants.

En ZeOSpi, quan es produeix una interrupció del Timer s'actualitza el rellotge i la informació del planificador de tasques (incloent un canvi de context, si així ho decideix). Quan es produeix una interrupció de la UART, concretament de recepció, s'insereix en el buffer de la UART, sobre el qual s'entrarà en detall més endavant.

10.3.2 Interrupcions software

La rutina d'atenció de les instruccions software té com a particularitat el fet que **no guarda tot el context de registres**. El motiu és que al ser una syscall ha de rebre els paràmetres proporcionats des del mode usuari i, per tant, no guarda els registres de r0 a r3.

Des del mode usuari s'utilitzen aquests registres per a proporcionar els paràmetres a la funció de la syscall del kernel i s'escriu en el registre r7 l'identificador de la syscall a executar, seguint el model de Linux en ARM [6].

Un cop en el kernel s'utilitza aquest identificador sobre la taula *sys_ call_ table* i l'execució salta a l'adreça continguda en aquesta posició. En la figura 24 es mostren gràficament els passos que segueix l'execució d'una syscall en ZeOSpi des de l'espai d'usuari fins al kernel.

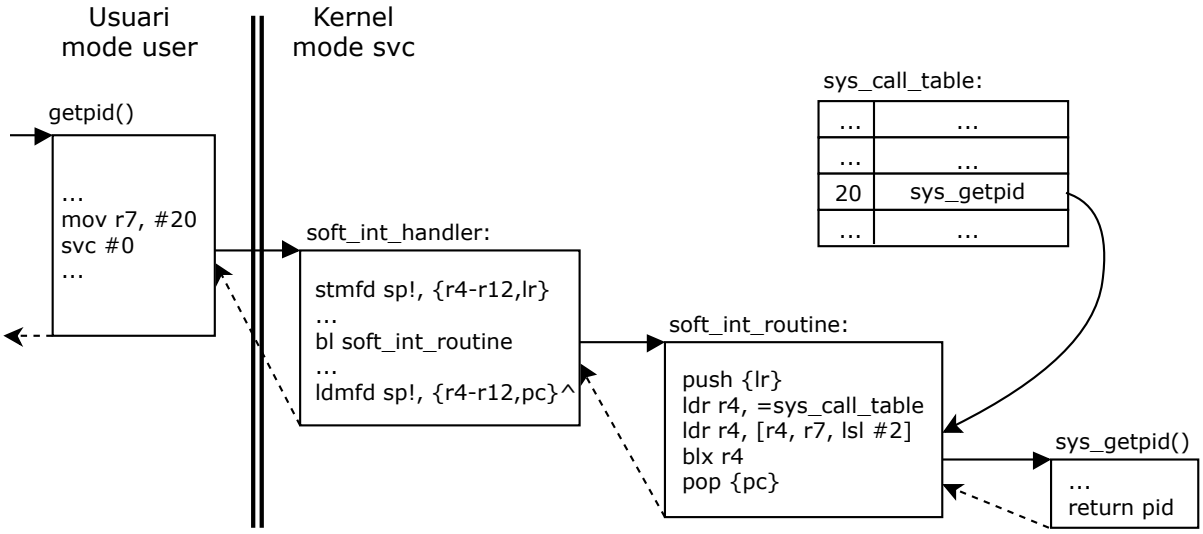

Figura 24: Procediment d'execució d'una syscall

10.3.3 Habilitació de les interrupcions

Per tal d'habilitar les interrupcions dels perifèrics del SoC cal poder accedir a la configuració de les del SoC. L'accés és el mateix que el que es requereix per a un perifèric: s'ha

de configurar la MMU per a que realitzi la traducció d'una adreça virtual a una de física del SoC.

Un cop es té accés, cal habilitar la interrupció de cadascun dels perifèrics a usar. Això es fa modificant els registres *Enable* de la taula 7.

- Per habilitar l'**UART**, hem de modificar el bit 29 del registre *Enable IRQ 1*
- Per habilitar el **Timer ARM**, hem de modificar el bit 1 del registre *Enable Basic IRQs*.

Finalment, només falta habilitar les IRQ (i/o les FIQ) per a que generin interrupcions al processador ARM. En ZeOSpi, les **interrupcions estan deshabilitades a nivell de kernel i habilitades en mode usuari**. Només hi ha una excepció: quan ens trobem executant la tasca idle necessitarem que el planificador del sistema canviï el procés en execució.

Com hem vist anteriorment, les interrupcions s'habiliten en el registre **CPSR**, implicant que per a poder habilitar les interrupcions cal un canvi de mode. Aquest canvi, però, **s'ha de realitzar preservant privilegis**. Per poder modificar el registre primer s'ha de canviar a mode sistema, ja que comparteix el registre CPSR. Aleshores, ja es possible habilitar les interrupcions modificant el bit corresponent. En ZeOSpi, s'habiliten en el pas de kernel a usuari, per aprofitar el canvi de mode.

10.4 Gestió dels perifèrics

En aquest capítol es revisarà la configuració dels perifèrics per al seu funcionament. Com hem vist anteriorment, en ZeOSpi els perifèrics tenen un espai reservat en la memòria del kernel per a poder accedir als registres de configuració d'aquests. Si generen interrupcions cal consultar els registres de gestió de les interrupcions del SoC, tot i que els perifèrics també inclouen registres propis.

Perifèric GPIO

El perifèric de GPIO és molt versàtil i pot servir per a implementar interfícies i comunicarnos amb altres dispositius electrònics, en ZeOSpi, però, només s'aprofiten les seves capacitats per donar accés a la UART i també per a **controlar un led** indicador que inclou la Raspberry Pi.

Per a realitzar aquest propòsit s'ha de configurar la funció a executar pels pins, configuració que es pot indicar en els registres *GPFSEL0-5*, especificant, cada tres bits, quina de les vuit funcions es vol que serveixi el pin en qüestió.

La UART usada en ZeOSpi fa servir els pins 14 i 15, i per a la configuració dels quals per part d'aquest dispositiu cal seleccionar la funció 5. En canvi, el led utilitza el pin 16, i l'hem de configurar amb la funció de sortida, podent després canviar el valor d'aquest pin amb els registres *GPSET0* i *GPCLR0*.

Perifèric UART Auxiliar

La UART en ZeOSpi està configurada per funcionar com a **8N1**, que és el mateix que 8 bits sense paritat i 1 bit de stop, a **115200 baudrate**. L'enviament de dades és síncron, no s'utilitzant-se interrupcions, i la recepció, en canvi, es asíncrona, usant-se la interrupció de recepció.

En aquesta implementació també s'utilitzen els buffers hardware FIFO de 8 caràcters tant d'enviament com de recepció, de la mateixa manera que a nivell de SO també s'utilitzen buffers d'enviament, com veurem més endavant.

Perifèric Timer

El Timer d'ARM està configurat a ZeOSpi amb un comptador de 32 bits, i per a generar una **interrupció de rellotge cada 1 ms**.

Per a generar la interrupció cada 1 ms s'utilitza el pre-scaler i el comptador, seguint la fórmula que hem vist anteriorment. El pre-scaler es configura amb valor 250, generant un decrement del comptador cada 1 μs, i el comptador es configura amb valor 1000. Quan el comptador arriba a 0, ha passat 1 ms i **el comptador es recarrega automàticament** amb el valor inicial.

Aquest perifèric ens serveix per a implementar un rellotge i donar una variable que el planificador de tasques utilitza per al canvi de processos. Cal tenir en compte, però, que en el kernel les interrupcions estan desactivades, excepte en l'execució de la tasca idle, i per tant el rellotge no s'incrementa en aquest cas.

10.5 Sistema d'Entrada/Sortida

El sistema d'E/S del ZeOSpi es basa en el perifèric de la UART per a realitzar les dues funcions centrals que són: read i write, (o lectura i escriptura).

Lectura

La **lectura de dades és síncrona**, i per tant, bloquejant del propi procés. Quan un procés vol fer una lectura, aquest intenta llegir informació del buffer de recepció; si té suficient informació, finalitza l'execució, i, en canvi, si requereix més informació de la que conté, el **procés es bloqueja en una cua** per a processos bloquejats. Aquest procediment es pot veure representat gràficament en la **primera fase** de la Figura 25.

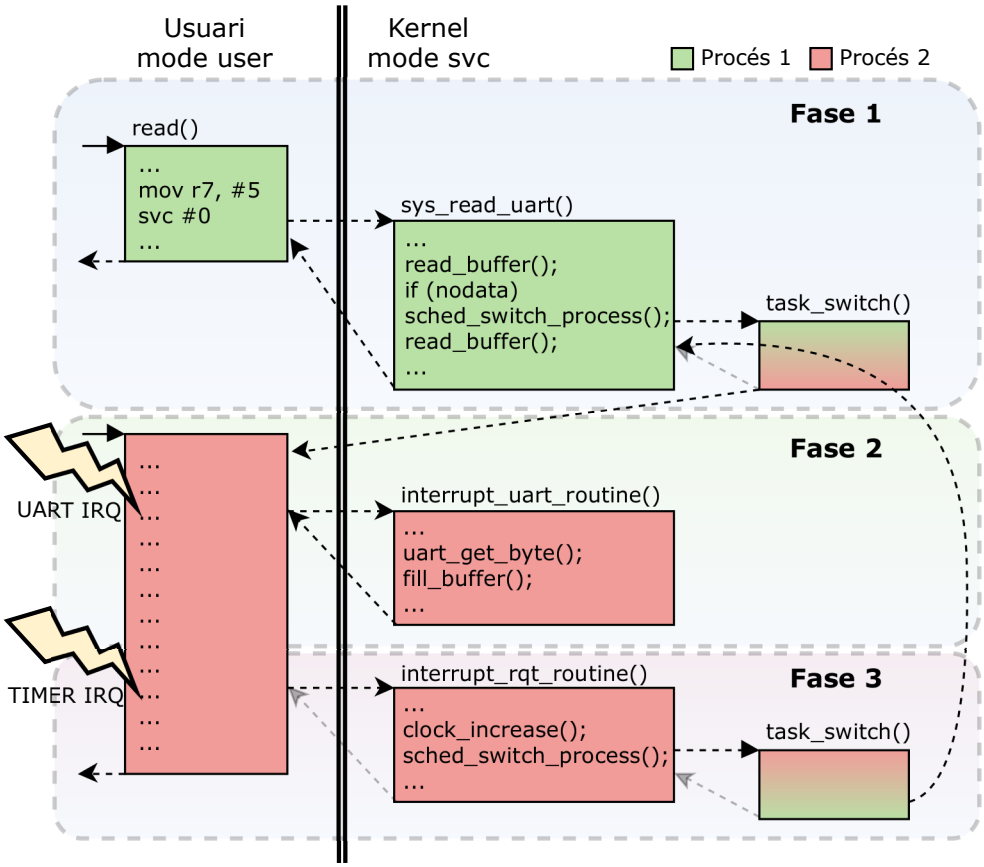

Figura 25: Esquema del procés de lectura per UART (Simplificació)

El procés de **recepció de dades**, per tant, **es asíncron** per processar-se a través d'una interrupció, emmagatzemant-se les dades en un buffer circular, com podem observar en la **segona fase** de la Figura 25. El procés que efectuava la lectura es desbloquejarà si el planificador del sistema determina que s'ha de realitzar un canvi de procés i si el SO ha rebut noves dades, com podem observar en la **tercera fase** de la Figura 25. Aquest procés es pot repetir, si la primera fase no té dades suficient per a finalitzar la lectura i aleshores es torna a bloquejar a l'espera de més dades.

Aquest sistema permet alliberar la CPU per a que executi altres tasques durant la lectura. Del contrari si la lectura no allibera la CPU, el bloqueig potencial pot ser força significatiu: si la lectura espera la **interacció d'una persona**, el temps entre la lectura de cada caràcter és molt important.

Escriptura

El **procés d'escriptura és síncron**, i per tant, bloquejant per al procés i, en aquest cas, també per a tot el SO. Ho és per al sistema, perquè l'escriptura no passa per un buffer com la lectura, i per tant en fer una escriptura, el procés interactua directament amb el perifèric de la UART per l'enviament de cada dada.

En aquest cas no hi ha interacció amb cap interrupció. El fet de no efectuar un enviament asíncron utilitzant un buffer implica un temps de bloqueig. Aquest **temps de bloqueig**, però, és d'esperar que sigui **molt menor** al que es podria donar en una lectura síncrona sense interrupcions. És així perquè l'enviament té com a destí un dispositiu que és el que s'encarregarà del seu tractament i no una persona.

En la figura 26 es pot apreciar com funciona l'enviament de dades: el bloqueig es produeix en l'espera de la funció *uart_send_byte()*. Aquesta funció es bloqueja esperant que el dispositiu estigui preparat per a l'enviament de dades.

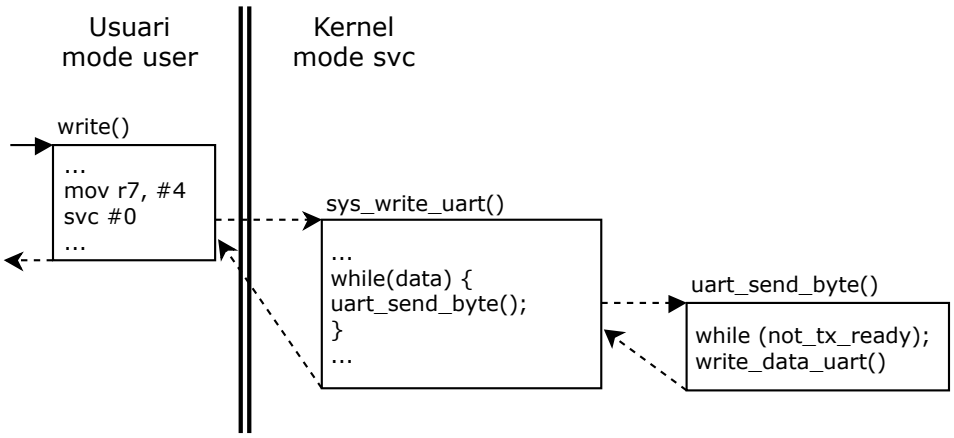

Figura 26: Esquema del procés d'escriptura per UART (Simplificació)

Finalment, cal recordar que el propi perifèric de la UART inclou uns buffers FIFO de 8 caràcters de profunditat, no massa gran. El fet que ja disposi de buffers no invalida utilitzar-los a nivell de SO, perquè tenint major profunditat permeten llegir dades causant un menor impacte en el temps de bloqueig d'un procés, i alhora són una generalització per a dispositius sense buffers propis.

10.6 Sistema multiprocés

El ZeOSpi en la Raspberry Pi permet l'execució de diverses tasques de forma concurrent, que no paral·lelament, perquè la CPU és uniprocessador. **ZeOSpi és un sistema preemptiu**, car el kernel decideix quan treu o posa un procés en execució. Amb un sistema multiprocés el SO s'apropa molt més a un SO modern, i permet un millor aprofitament dels recursos del dispositiu, com per exemple la UART.

Per a donar suport a aquesta característica el sistema ha de ser capaç de poder crear tasques, eliminar-ne, tenir-ne una representació de les dades d'aquestes i disposar d'un mecanisme per a decidir i gestionar el intercanvi de tasques en execució.

10.6.1 Estructures

El suport de ZeOSpi a l'execució multiprocés requereix de les estructures següents:

Estructura task_union

L'estructura task_union, normalment anomenada PCB (Process Control Block, en anglès), representa i emmagatzema la informació d'un procés. En ZeOSpi només hi ha declarades **10 instàncies**, per tant com a molt poden haver-hi 10 processos executant-se concurrentment. Aquesta estructura es compon d'informació sobre el procés i la pila de kernel d'aquest. En la figura a continuació es pot apreciar amb més detall:

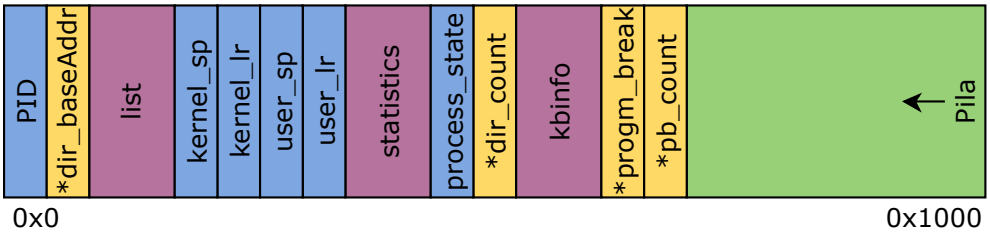

Figura 27: Task_union d'un procés

Els components d'un *task_union* són:

- **PID**: És un enter que identifica el procés.
- **dir_page_baseAddr**: Punter dirigit a la base de la taula de directoris (o taula de primer nivell) d'aquest procés.
- **list**: Estructura que permet inserir el procés en una cua, que veurem més endavant.
- **kernel_sp**: Variable que guarda l'adreça de la pila en el kernel quan es treu la tasca d'execució per a poder restaurar-la.
- **kernel_lr**: Variable que guarda l'adreça de retorn en el kernel quan es treu la tasca d'execució per a poder restaurar-la.
- **user_sp**: Variable que guarda l'adreça de la pila d'usuari per poder restaurar correctament la tasca si es produeix un canvi de tasca en execució.
- **user_lr**: Variable que guarda l'adreça de retorn d'usuari per poder restaurar correctament la tasca si es produeix un canvi de tasca en execució.
- **statistics**: Estructura que conté informació estadística relativa al procés, com ho són el temps d'execució, el nombre de canvis de context i la informació respecte el planificador.
- **process_state**: Indicador de l'estat del procés: running, ready, blocked o zombie.
- **dir_count**: Punter a l'enter que indica el nombre de processos que referencien la mateixa taula de directoris que aquest procés.

- **kbinfo**: Informació respecte l'E/S del procés: punter a un buffer, i caràcters per llegir i llegits.
- **program_break**: Punter indicador de l'espai de memòria dinàmica.
- **pb_count**: Punter al enter que indica el nombre de processos que referencien la mateixa zona de memòria dinàmica.
- **Pila**: Pila de sistema del procés.

Cues de planificació

Les cues de processos tenen com a objectiu la **gestió dels diferents estats de les tasques**. Per exemple, si tenim més de dues tasques, volem saber quines són les que no s'estan executant per a poder transferir l'execució quan ho creguem convenient. Aquesta és la feina que s'encarregarà de realitzar el planificador del sistema, del qual en parlarem més endavant.

Les diferents cues que s'utilitzen a ZeOSpi són les següents:

- **freequeue**: Cua que agrupa les task_union lliures per a ser utilitzades.
- **readyqueue**: Cua que agrupa els processos que esperen entrar en execució.
- **keyboardqueue**: Cua de processos bloquejats esperant dades de l'E/S.

Tasques inicials

Inicialment, el SO transfereix l'execució a l'espai d'usuari. En un sistema uniprocés no s'ha de realitzar cap gestió extra, en canvi, en un sistema multiprocés es transfereix l'execució de la **primera tasca, task1,** que s'ha d'inicialitzar corresponentment en la inicialització del sistema.

Quan el sistema es queda sense tasques per a posar en execució, per exemple quan aquestes esperen dades de l'E/S, el sistema executa la **tasca idle** pensada per quan el SO no té cap tasca a executar. Aquesta tasca, de kernel, **activa les interrupcions** per a poder ser despertada i deixar que el planificador actuï si s'ha de posar un procés en execució.

Ampliació de les estructures de la MMU

Per a la gestió de processos en un sistema multiprocés, també s'han de proporcionar suficients estructures per a representar l'espai de dades de cadascun dels processos, això és, les taules de primer i segon nivell. També s'ha de tenir en compte que els processos tenen un espai d'adreces diferent i en un canvi d'execució de processos s'ha de reconfigurar la base de la MMU i netejar la TLB. Com el nombre de tasques del sistema està fixat a 10, es declaren inicialment.

Compartició de l'espai d'adreces

Els processos de ZeOSpi poden ser de dos tipus: processos independents, o processos amb recursos compartits (també anomenats threads). La principal distinció és que, mentre els independents no comparteixen l'espai de dades i heap d'usuari, els threads sí que ho fan. En la imatge 28 es pot veure una representació gràfica:

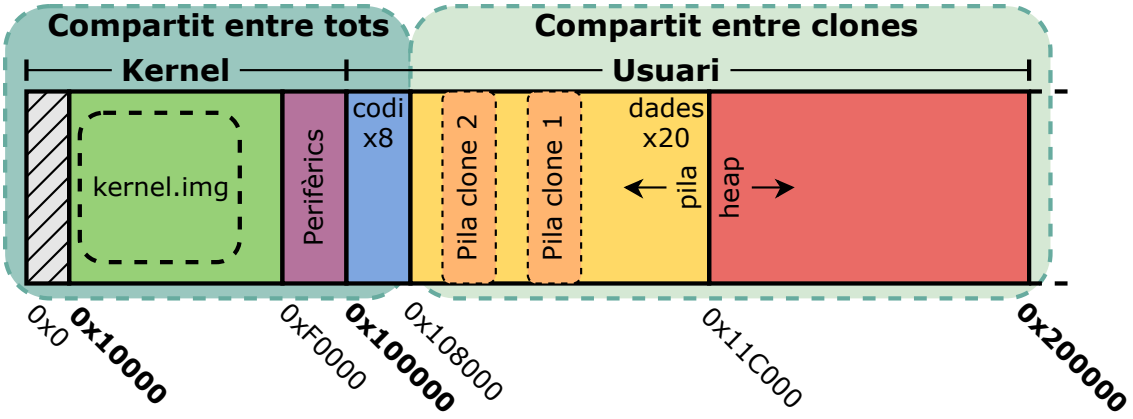

Figura 28: Compartició de l'espai d'adreces entre processos

Tots els processos comparteixen les mateixes pàgines de kernel i de codi d'usuari. L'espai de dades i heap també és privat per a la majoria de processos, excepte a aquells que s'han creat per a compartir recursos, mitjançant la funció clone que veurem a continuació. Aquests últims processos requereixen, com tots, una pila. Aquesta, però, no pot estar a la mateixa posició que altres processos amb els que comparteixin l'espai d'adreces, perquè interferirien l'execució. Per aquest motiu s'ha de deixar un espai específic per a aquests processos, com es pot veure en la Figura 28.

10.6.2 Creació i destrucció de processos

A continuació es descriu el funcionament de les crides de sistema; fork, clone i exit, utilitzades per a la creació i destrucció de processos.

Syscall Fork

El Fork és la **funció bàsica per a crear un nou procés**. El procés pare fa una crida a la syscall fork i, en retornar de l'execució rep, el pid del fill i continua l'execució on l'havia deixat. Com a conseqüència de la crida de la funció fork, es crea un nou procés, el fill. Aquest té un nou PCB, noves taules de primer i segon nivell, i les dades del procés són una còpia de les dades.

Els dos nous **processos són independents i no comparteixen cap espai d'adreces propi**. Aquest nou procés entrarà en execució quan el planificador ho decideixi i iniciarà l'execució just després d'on el seu pare havia realitzat la crida al fork. Aquest fork retornarà zero, per indicar que és el fill.

Els passos que segueix el fork per a la creació del nou procés són els següents:

- Obtenció d'una *task_union* nova.
- Obtenció de *frames* lliures per a l'espai de dades d'usuari.
- Còpia de la pila i obtenció d'un directori nou.
- Assignació dels mateixos *frames* de codi en el procés nou (espai de codi compartit).
- Assignació dels *frames* nous a l'espai de dades.
- Còpia de les dades en el nou espai de dades.
- Còpia de la memòria dinàmica.
- Assignació del context de retorn de kernel i usuari:
  - **Kernel**: El registre sp apunta a la posició abans d'entrar en la funció *sys_fork* relativa al nou *task_union* i lr a la funció *ret_from_fork*.
  - **Usuari**: Els registres sp i lr mantenen el mateix valor que el procés pare.
- Inicialització dels valors de estadístiques i obtenció del nou PID.
- Introducció del nou procés en la cua *readyqueue* del planificador.
- Finalització l'execució.

L'assignació de l'adreça de retorn a *ret_from_fork* es produeix per a poder controlar el valor de retorn del fill de la funció del fork, perquè volem que retorni un zero.

Syscall Clone

Clone és la segona syscall utilitzada per a crear processos, de la que disposa ZeOSpi. En aquest cas, però, crea un thread, perquè **el procés que crea comparteix espai de memòria amb el procés original**. Concretament, com veiem en la Figura 28, comparteixen el mateix espai d'adreces i la mateixa zona de dades i memòria dinàmica, o heap.

Els passos que segueix el clone per a la creació del nou procés són els següents:

- Obtenció d'una *task_union* nova.
- Còpia de la pila.
- Actualització del nombre de referències al mateix directori i program_break.
- Assignació del context de retorn de kernel i usuari:
  - **Kernel**: El registre sp apunta a la posició abans d'entrar en la funció *sys_fork*, relativa al nou *task_union* i lr a la funció *ret_from_fork*.

  - **Usuari**: El registre sp apunta a la pila d'usuari proporcionada. El valor de lr és indiferent, perquè és el valor que prendrà lr en mode usuari, i a on saltarà l'execució del nou procés.
  - **Pila de kernel**: Modificació del contingut per a canviar l'adreça de retorn a l'usuari per tal que executi la funció proporcionada.

- Inicialització dels valors de estadístiques i obtenció del nou PID.
- Introducció del nou procés en la cua *readyqueue* del planificador.
- Finalització de l'execució.

Com es pot observar, el procediment de la funció clone és una versió simplificada del fork. És així perquè s'elimina la gestió i copia de les dades d'un espai d'usuaris a l'altre, car les dades són compartides.

Finalment, com en el fork, la funció clone surt del kernel a través de la funció *ret_from_fork*, en aquest cas, però, és per conveniència.

Syscall Exit

Els recursos del sistema són limitats, i el nombre de tasques que es poden crear també, un total de 10. Per això, la syscall exit **s'encarrega de la destrucció de tasques** i l'alliberació dels seus recursos per a poder ser reutilitzats, com ara les estructures task_union, i les estructures i pàgines físiques de memòria, evitant fer-ho si són recursos compartits. També allibera els recursos de la memòria dinàmica i els semàfors, que veurem més endavant.

10.6.3 Gestió de l'execució dels processos

A continuació es descriu el funcionament de sistemes com: el sistema de canvi de procés en execució (task switch), i el planificador de tasques en execució, utilitzats per a la gestió de l'execució dels processos.

Task switch

Un cop el SO té la capacitat de crear tasques, per a fer-ne ús ha de ser capaç de canviar la tasca en execució; d'això és del que s'encarrega la funció de task_switch. Aquesta funció **ha de realitzar l'intercanvi de la tasca en execució** per una qualsevol, preservant el context d'execució de la primera, i recuperant el de la segona.

El context d'execució d'un procés està definit pels registres, incloent-hi sobretot sp i lr, i la memòria. Per tant, task_switch ha de ser capaç de salvar els del procés actual i modificar-los pels del nou.

Procés de task_switch per a l'intercanvi de dos processos:

- Guardar registres generals (r0-r12) a la pila del procés actual (procés a ser canviat).
- Canviar l'espai de memòria. No té efecte sobre les accions que es faran a continuació perquè bàsicament es modifica l'espai d'usuari.
- Guardar registres sp i lr de kernel a la pila del procés actual. La còpia dels d'usuari es realitzen en entrar al mode kernel.
- Canviar els registres sp i lr pels del nou procés.
- Accedir temporalment a mode system i modificar els registres sp i lr d'usuari pels nous.
- Retornar l'execució saltant al contingut de lr.

Planificador

Un cop el sistema pot crear tasques i canviar la que està en execució, s'ha de decidir una **política per a l'intercanvi de tasques**, feina de la qual s'encarregarà el planificador (o scheduler, en anglès). El que es busca amb un planificador és repartir l'ús d'un recurs a partir d'unes polítiques.

En **ZeOSpi, s'usa la política de Round Robin**, que té com objectiu el *"fair scheduling"* o l'ús equitatiu dels recursos, l'ús de la CPU, i indirectament l'E/S.

El funcionament de Round Robin ve marcat per l'ús de **quàntums**, una unitat de temps. Quan una tasca inicia l'execució, se li assigna el *quàntum*. A mesura que passa el temps el *quàntum* es decrementa fins esgotar-se, moment en el qual el planificador intercanvia el procés en execució per un altre de la cua, que de nou torna a tenir un *quàntum*.

En ZeOSpi, el planificador posa en execució el primer procés de la cua *readyqueue*, a no ser que existeixin processos que puguin llegir informació del dispositiu d'E/S, i que estiguin en la cua de processos bloquejats; la *keyboardqueue*. De ser així, el planificador prioritza l'execució d'aquests processos.

El tractament de la cua de processos bloquejats és especial, perquè es prioritza que el primer procés d'aquesta realitzi tota la lectura sobre el dispositiu, moment en el qual podrà començar a llegir el segon procés de la cua. Es podria descriure la planificació dels processos bloquejats com a FIFO (First in first out, en anglès). Aquest fet ve motivat perquè no es vol repartir una entrada de dades de forma intercalada entre dos o més processos.

Finalment, la implementació del planificador permet ser substituïda fàcilment per una altra política, i també **proporciona informació estadística dels processos**, com ara el nombre de canvis de context o el temps d'execució.

10.7 Sistema de sincronització

Per a poder tenir control sobre els processos des de l'espai d'usuari, es poden fer servir sistemes de sincronització de processos. Aquests sistemes ens permeten limitar l'accés a un recurs o una regió critica, ordenar l'execució de diferents processos, i, en un sistema multiprocés, evitar condicions de carrera.

El sistema de sincronització de ZeOSpi és el de semàfors implementats amb quatre syscalls per a utilitzar-los:

- **Sem_init**: Inicia un semàfor amb un valor determinat
- **Sem_wait**: Demana accés al semàfor.
- **Sem_signal**: Informa al semàfor que ja no necessita accés.
- **Sem_destroy**: Elimina semàfor, i allibera processos bloquejats.

En iniciar un semàfor, aquest té un valor associat. Quan un procés demana accés al semàfor, aquest decrementa el valor del semàfor, i si el valor es zero bloqueja el procés en una cua pròpia d'aquest. En fer el *signal*, incrementa el valor o allibera un procés de la cua si el valor es zero. Finalment, en destruir-lo, s'alliberen tots els processos bloquejats.

Un exemple pràctic és un semàfor amb valor zero, que s'usi entre dos o més processos. Aquest semàfor, que implementa una **zona d'exclusió mútua**, permet bloquejar l'accés a una zona de codi o una variable. També podríem pensar en **limitar el nombre de processos que accedeixen al dispositiu** UART d'E/S usant un semàfor amb valor superior a zero, com ara ú per a que només dos processos poguessin llegir alhora.

10.8 Sistema de memòria dinàmica

La memòria dinàmica ens permet aconseguir nou espai de memòria en temps d'execució, i alhora, també és l'únic sistema de comunicació entre processos que implementa ZeOSpi.

La memòria dinàmica està implementada en la zona de heap d'un procés, el qual compta amb un punter, program_break, que indica fins a on s'estén la memòria reservada. La memòria reservada per un procés va des de l'inici del heap en la posició 0x11C000 fins a la indicada pel program_break. Per tant, la reserva de memòria és lineal en el sentit que no és possible alliberar espai entre el inici i el final.

Des de l'espai d'usuari es pot utilitzar la syscall sbrk per a augmentar o disminuir aquesta zona reservada tot indicant-ho en el valor que se li proporciona. Aquesta syscall retorna l'últim valor del program_break i reserva o allibera l'espai requerit.

11 Revisió del projecte

El desenvolupament del projecte ha tingut desviacions respecte a la planificació que es plantejava l'inici, així com també n'han tingut els pressupostos inicials. En aquest apartat es fa una revisió de la planificació i els pressupostos inicials.

11.1 Desviacions en la planificació

La principal desviació del projecte és deguda al fet que la **planificació inicial no incloïa el curs de GEP** realitzat de 17 de Febrer fins al 20 de Març, que implicava realitzar una serie d'entregues de documents i diverses presentacions. Durant aquest període es va continuar treballant en el projecte, però la dedicació no va poder ser complerta, com s'havia planificat inicialment.

Una segona desviació és deguda a la **complicació de la tasca de Memòria** de l'Etapa d'Implementació base i d'Implementació en la Raspberry Pi, que va suposar una dedicació superior a l'esperada, amb problemes que requerien repassar la documentació del processador detalladament.

Finalment, la planificació inicial es basava en **8 hores de dedicació per dia**, la qual ha estat alterada per l'assistència, durant la realització d'aquest projecte, a dues assignatures i també la participació en esdeveniments com són diversos seminaris i el Mobile World Congres, entre d'altres. Per aquest motiu, es considera que **6,5 hores de dedicació per dia** és una aproximació més fidel.

Alternatives de la planificació

Entre les opcions detallades en el capítol de la Valoració d'alternatives, es va optar per la segona, que consistia en realitzar totes les etapes, durant el marge de temps que es preveia. Inicialment es va valorar no incloure la implementació dels semàfors i la memòria dinàmica, però finalment es van poder mantenir dins del projecte.

11.2 Nova planificació

En la figura 29 s'inclou el diagrama de Gantt de la planificació final on es poden observar les modificacions degudes a les desviacions. Durant la realització del curs GEP, la dedicació es va repartir en parts iguals al curs i al projecte en sí.

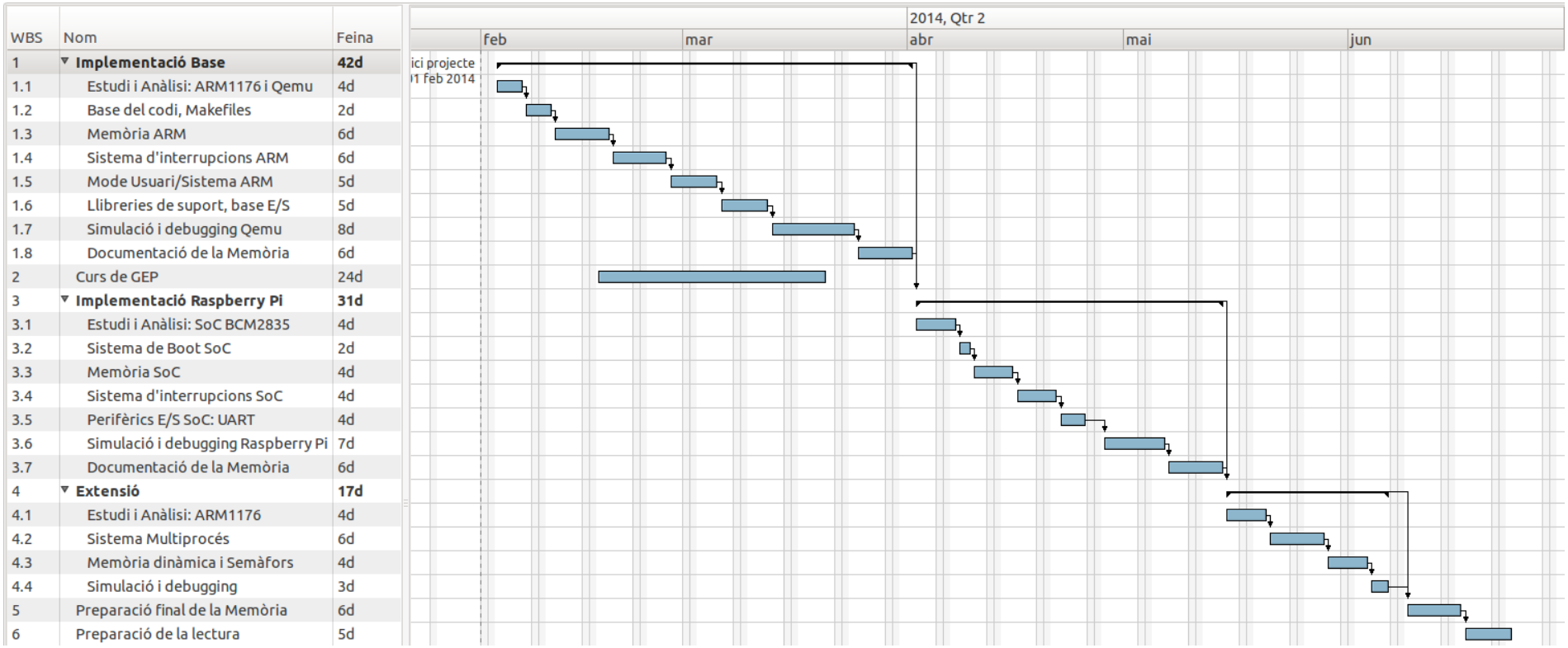

Figura 29: Diagrama Gantt de la planificació final

11.3 Pressupost final

El projecte va començar el dia 1 de Febrer de 2014 i finalitza l'ultima setmana de juny, on es prepara la lectura. La dedicació ha estat de 6,5 hores diàries durant el total de 101 dies de realització del projecte.

Com en l'apartat dels pressupostos inicials, els costos del projecte es desglossen en els apartats següents: recursos humans, recursos materials, recursos de software, despeses indirectes i finalment el total agregat de les despeses.

El cost total final del projecte és de **20.823,6 €**.

Despesa en recursos humans

La despesa en recursos humans es veu directament afectada pels canvis en la planificació. El total d'hores de cada rol es determina segons les categories següents, basades en les tasques especificades en el diagrama de la planificació:

- **Documentació i preparació de la lectura:** Suma un total de 23 dies, 149,5 hores. Cap de projecte.
- **Etapes d'estudi i Anàlisi:** Suma un total de 12 dies, 78 hores. Dissenyador.
- **Etapes de debugging:** Suma un total de 18 dies, 117 hores. Tester.
- **La resta de tasques:** 48 dies, 312 hores, es reparteixen entre programador, dos terceres parts i dissenyador, la part restant.

Rol	**Preu (€/h)**	**Temps (h)**	**Import (€)**
Cap de projecte	40	149,5	5.980
Dissenyador	20	182	3.640
Programador	20	208	4.160
Tester	12	117	1.404
Total			**15.184**

Taula 10: Despesa en recursos humans. Font: ICSA Grupo 2013 -2014.

Despesa en recursos materials

Els recursos materials del projecte, no s'han vist alterats, i per tant es mantenen en un **total de 193,62 €**.

Despesa en software

El software utilitzat pel desenvolupament del projecte és software de lliure distribució i gratuït. Per tant no repercuteix sobre les despeses totals del projecte.

Despeses indirectes

Pel que fa a les despeses indirectes, tampoc ha hagut cap alteració, mantenint-se a un **total de 1.832€**.

Despeses totals

Finalment, sumem les despeses en recursos humans i materials, i obtenim un total al que després apliquem l'IVA actual, del 21%.

Concepte	**Total (€)**
Recursos humans	15.184
Recursos materials	193,62
Recursos software	0
Despeses indirectes	1.832
Total	**17.209,6**
Total+IVA	**20.823,6**

Taula 11: Despeses totals.

12 Conclusions

En aquest projecte s'ha realitzat el port del SO ZeOS de l'arquitectura Intel x86 a ARM, específicament pel dispositiu Raspberry Pi.

El port és perfectament funcional, i dóna suport per als perifèrics UART i GPIO del dispositiu per a poder interaccionar amb la Raspberry Pi. És un SO multiprocés que compta amb suport per a memòria virtual, sistemes de sincronització de processos i memòria dinàmica.

Aquest projecte ha estat el meu primer contacte a baix nivell amb l'arquitectura ARM, i ha estat molt interessant veure les implicacions i canvis que comportava l'arquitectura ARM respecte a Intel en el software.

13 Treball futur

Tot i que el SO desenvolupat incorpora moltes característiques, tanmateix existeixen moltes àrees en les que es pot ampliar, com són:

Aplicacions d'espai d'usuari Aplicacions generals que donin informació sobre els processos, com ara *top*, o un shell per l'execució de comandes o utilitats. També podria ser molt interessant ampliar la implementació de la libc.

Característiques avançades del Hardware Característiques que facin ús tant del SoC com del processador ARM per agilitzar processos del kernel o en aspectes com un millor consum energètic.

Característiques d'altres SO Com ara polítiques de planificació diferents, sistemes d'intercanvi d'informació entre processos o una gestió de dispositius i fitxers en ús per un procés.

Implementació dels perifèrics El SO es podria beneficiar molt, de perifèrics com la sortida HDMI, USB o Ethernet, dispositius que podrien donar molts més usos a la Raspberry Pi amb ZeOSpi.

Redefinició de l'espai d'adreces Amb el propòsit d'usar tot l'espai disponible, ja que actualment segueix el mateix model que el SO original per conveniència, però això té limitacions en quan a l'espai utilitzable de memòria pels processos.

Sistema de fitxers Pot ser molt interessant dotar el SO de suport per a sistemes de fitxers, donat que la Raspberry Pi incorpora una targeta SD.

Bibliografia

[1] Daniel P. Bovet. *Understanding the Linux Kernel*. O'Reilly Media, 2008.

[2] Broadcom. *BCM2835 ARM Peripherals*. Broadcom, 2012.

[3] FIB Departament d'Arquitectura de Computadors. Pàgina web so2, sistemes operatius 2, 2014. http://docencia.ac.upc.edu/FIB/grau/SO2/, [Accedit Març 2014].

[4] FIB Departament d'Arquitectura de Computadors. Sistemes operatius avançats, 2014. http://docencia.ac.upc.edu/FIB/grau/SOA/, [Accedit Març 2014].

[5] Arduino Due. Arduino, overview, 2014. http://arduino.cc/en/Main/ArduinoBoardDue/, [Accedit Abril 2014].

[6] Free Electrons. Linux cross reference, 2014. http://lxr.free-electrons.com/, [Accedit Abril 2014].

[7] Raspberry Pi Foundation. Raspberry pi, about us, 2014. http://www.raspberrypi.org/about, [Accedit Març 2014].

[8] Raspberry Pi Foundation. Raspberry pi, faq, 2014. http://www.raspberrypi.org/faqs, [Accedit Març 2014].

[9] Inc Free Software Foundation. Gcc, the gnu compiler collection, 2014. https://gcc.gnu.org, [Accedit Març 2014].

[10] Z. Navarro J J Gómez and J García. Disseny i implementació en fpga d'un computador sisa i del sistema operatiu zeos per a aquesta arquitectura. Master's thesis, Universitat Politècnica de Catalunya, 2004-2009.

[11] Hardkernel. Odroid-u3, technical detail, 2014. http://hardkernel.com/main/products/prdt_info.php?g_code=G138745696275, [Accedit Abril 2014].

[12] SECO USA Inc. Udoo, features, 2014. http://www.udoo.org/features/, [Accedit Abril 2014].

[13] Texas Instruments Incorporated. *PC16550D Universal Asynchronous Receiver/-Transmitter with FIFOs*. Texas Instruments Incorporated, 1995.

[14] David A. Holland Ada T. Lim and Margo I. Seltzer. *A New Instructional Operating System*. Harvard University, 2002.

[15] ARM Limited. *The ARM-THUMB Procedure Call Standard*. ARM Limited, 2000.

[16] ARM Limited. *ARM Architecture Reference Manual*. ARM Limited, 2004-2012.

[17] ARM Limited. *ARM1176JZF-S™ Technical Reference Manual*. ARM Limited, 2008.

[18] Linaro. Linaro: open source software for arm socs, 2014. http://www.linaro.org/, [Accedit Març 2014].

[19] Parallel and MIT distributed operating system group, CSAIL. Jos, 2014. http://pdos.csail.mit.edu/6.828/2012/, [Accedit Abril 2014].

[20] Andrew N. Sloss Dominic Symes and Chris Wright. *ARM System Developer's Guide*. MK, 2004.

[21] Wikipedia The free encyclopedia. List of operating systems raspberry pi, 2014. http://en.wikipedia.org/wiki/Raspberry_Pi#List_of_operating_systems, [Accedit Abril 2014].

Glossari

ARM Arquitectura, processador o dispositiu desenvolupat per l'empresa ARM, principalment utilitzada en dispositius mòbils. 9, 10, 12, 13, 15, 18, 20, 21, 25

CPU Sistema hardware encarregat d'executar les instruccions en que es codifica un programa software. 9–11

cross-compiling És refereix a compilar un codi per a una arquitectura diferent del dispositiu que s'està utilitzant per fer-ho.. 18, 27

datasheet Document tècnic de referencia. Habitualment sobre un dispositiu. 43

E/S Un sistema d'Entrada/Sortida és el sistema que s'encarrega de la comunicació entre el processador i els perifèrics associats. 7, 9, 16, 20, 21

FIFO First in, first out, en anglès. Concepte d'estructures de dades que descriu el funcionament d'una cua.. 56, 58, 64

Linux Kernel del sistema operatiu GNU/Linux creat per Linux Torvalds el 1991. 8, 13, 14

MMU Sistema hardware encarregat de gestionar i protegir l'accés a la memòria d'un computador. 11

Raspberry Pi El mini-computador Raspberry Pi és un dispositiu creat amb l'idea de promoure l'educació de conceptes bàsics de programació. 7–13, 15, 18, 20, 22

scheduler Planificador. Gestiona i decideix l'execució d'una tasca en un instant determinat. 64

sistema de boot És el sistema d'arrencada d'un ordinador o dispositiu que s'encarrega d'iniciar l'execució del sistema operatiu. 9, 10, 20

SO Sistema software encarregat de gestionar l'accés als recursos d'un computador amb l'objectiu de protegir els recursos respecte l'execució de programes. 7–11, 13–15, 18, 20–22

SoC Circuit integrat que agrupa tots els components d'un computador o d'un sistema electrònic en un sol xip. 9, 12, 15, 16, 20, 21

syscall Crida de sistema d'un sistema operatiu. 54, 61–63, 65

toolchain Conjunt de programes o eines usades per a la creació d'un executable. 27

UART Sistema universal serie asíncron de recepció i enviament de missatges. 12, 13, 16, 20, 21

x86 Arquitectura desenvolupada per Intel, principalment utilitzada en ordinadors de propòsit general. 7, 14, 15

A Fitxers del projecte

Junt amb aquest document s'inclouen els fitxers codi font de ZeOSpi, un makefile, l'imatge del SO i un script per a gravar l'imatge del SO en la tarjeta SD. També s'inclouen en la carpeta boot_folder els fitxers, preparats per a ZeOSpi, que ha de contenir la tarjeta SD.

Per a compilar el codi font és necessari obtenir les eines de compilació proveïdes en els repositoris oficials de la Raspberry Pi, on es pot trobar el toolchain de GNU. El projecte ha estat realitzat amb les següents versions dels programes:

- **gcc**: gcc versión 4.8.3 20140106 (prerelease) (crosstool-NG linaro-1.13.1-4.8-2014.01 - Linaro GCC 2013.11)
- **ld**: GNU ld (crosstool-NG linaro-1.13.1-4.8-2014.01 - Linaro GCC 2013.11) 2.24.0.20131220
- **as**: GNU ensamblador versión 2.24.0 (arm-linux-gnueabihf) utilizando BFD versión (crosstool-NG linaro-1.13.1-4.8-2014.01 - Linaro GCC 2013.11) 2.24.0.20131220
- **Objcopy**: GNU objcopy (crosstool-NG linaro-1.13.1-4.8-2014.01 - Linaro GCC 2013.11) 2.24.0.20131220
- **gdb**: GNU gdb (crosstool-NG linaro-1.13.1-4.8-2014.01 - Linaro GCC 2013.11) 7.6.1-2013.10
- **Qemu**: QEMU emulator version 1.5.0 (Debian 1.5.0+dfsg-3ubuntu5.4), Copyright (c) 2003-2008 Fabrice Bellard

La imatge de ZeOSpi proveïda està programada per a reenviar tot el que rep pel dispositiu UART a través d'aquest mateix.

B Instruccions assemblador ARM rellevants

En aquest annex es descriu el funcionament de instruccions assemblador relevants utilitzades en el projecte. Es pot trobar més informació en el document de referencia de ARM.

MRS i MSR Instruccions per accedir al registre CPSR o SPSR i llegir-ne el resultat o escriure'l, respectivament. La sintaxi d'ús es la següent:

```
MRS{cond}  <Rd>,CPSR|SPSR
MSR{cond}  CPSR|SPSR,<Rm>
```

MRC i MCR Instruccions per accedir als coprocessadors i llegir-ne el resultat o escriure'l, respectivament. La sintaxi d'ús es la següent:

```
MRC{cond}  P15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
MCR{cond}  P15,<Opcode_1>,<Rd>,<CRn>,<CRm>,<Opcode_2>
```

Els valors CRn i CRm indiquen el coprocessador, de c0 a c15. I els opcode de 0 a 7.

CPS És una instrucció que modifica el registre CPSR canviant només el mode, i els flags d'abort i habilitació d'interrupcions IRQ i FIQ. Permet modificar el CPSR més rapidament que usant les instruccions MRS i MSR. La sintaxi d'ús es la següent:

```
CPSeffect  iflags{, #mode}
CPS        #mode
```

El *effect* pot ser IE o ID segons es vol habilitar o deshabilitar les *iflags* proporcionades, que poden ser a, i o f. El *mode* indica amb els últims cinc bits del mode al que es vulgi canviar.

STM i LDM Instruccions usades per a guardar i carregar, respectivament, de registres a memòria i viceversa. La sintaxi d'ús es la següent:

```
STM{addr_mode}{cond} Rn{!}, reglist{^}
LDM{addr_mode}{cond} Rn{!}, reglist{^}
```

On *addr_mode* es refereix a l'accés a la pila i pot ser IA, IB, DA, DB o FD. El registre *Rn* és usat per l'accés a la pila, i la *reglist* és la llista de registres que es copien o es modifiquen. El simbol *!* indica a l'instrucció que l'adreça final es vol tornar a escriure al registre. El simbol ^, usat en la LDM, i només si *reglist* conté el pc, indica que s'ha de copiar SPSR al CPSR, realitzant un canvi de mode per a retornar d'excepcions.

SRS i RFE SRS guarda l'estat de retorn, registre lr i spsr, en la pila del mode indicat. RFE retorna d'una excepció usant els registres guardats en la pila per l'instrucció SRS. La sintaxi d'ús es la següent:

```
SRS{addr_mode}{cond} sp{!}, #modenum
RFE{addr_mode}{cond} Rn{!}
```

On *addr_mode* es refereix a l'accés a la pila i pot ser IA, IB, DA, DB o FD (full descending). El simbol *!* indica a l'instrucció que l'adreça final es vol tornar a escriure al registre. Finalment, el *mode* indica amb els últims cinc bits del mode del qual es vol utilitzar el registre sp.