Custom Arm OS Design

Embedded Systems Design

Author:
Dimitrios Lampros – 03117070
Dimitrios Stamatios Bouras – 03117072
Georgios Pagonis – 03117030

Supervisors :
Manolis Katsaragakis
Dimitrios Soudris

March 2022

National Technical University of Athens

Contents

Concept

Ο στόχος της εργασίας είναι η υλοποίηση ενός kernel για έναν Arm επεξεργαστή. Τα θέματα με τα οποία ασχοληθήκαμε είναι :

- Η διεπαφή ανάμεσα στις περιφεριακές συσκεύες και τον επεξεργαστή.
- Η υλοποίηση exceptions και system calls.
- Η δημιουργία και διαχείρηση διεργασίων.

Contents

Bare Metal

Board

Board

Το board το οποίο επιλέχθηκε είναι το Rasberry Pi 3 A+.

- Chip: Broadcom BCM2837B0
- Processor: Cortex-A53
- Access: Extended 40-pin GPIO header

Το specification του είναι : Raspberry Pi 3 Model A+.

Bare Metal

Architecture

Architecture

Ο arm επεξεργαστής Cortex-A53 χρησιμοποίει αρχιτεκτονική Armv8-A. Το instruction set αυτής είναι:

- 64-bit A64-AArch64.
- 32-bit A32-AArch32.
- 16-bit T32-AArch32(Thumb instruction set).

Η αρχιτεκτονική υποστηρίζει όλα τα Exception levels:

- EL0: Είναι το χαμηλότερο επίπεδο privilege, θεωρείτε το user mode.
- EL1: Kernel mode, όπου τρέχει το λειτουργικό μας.
- EL2: Υποστηρίζει την παραληλοποίηση του επεξεργατή.
- EL3: Παρέχει υποστήριξει για το secure state.

Bare Metal

Bootloader

Bootloader

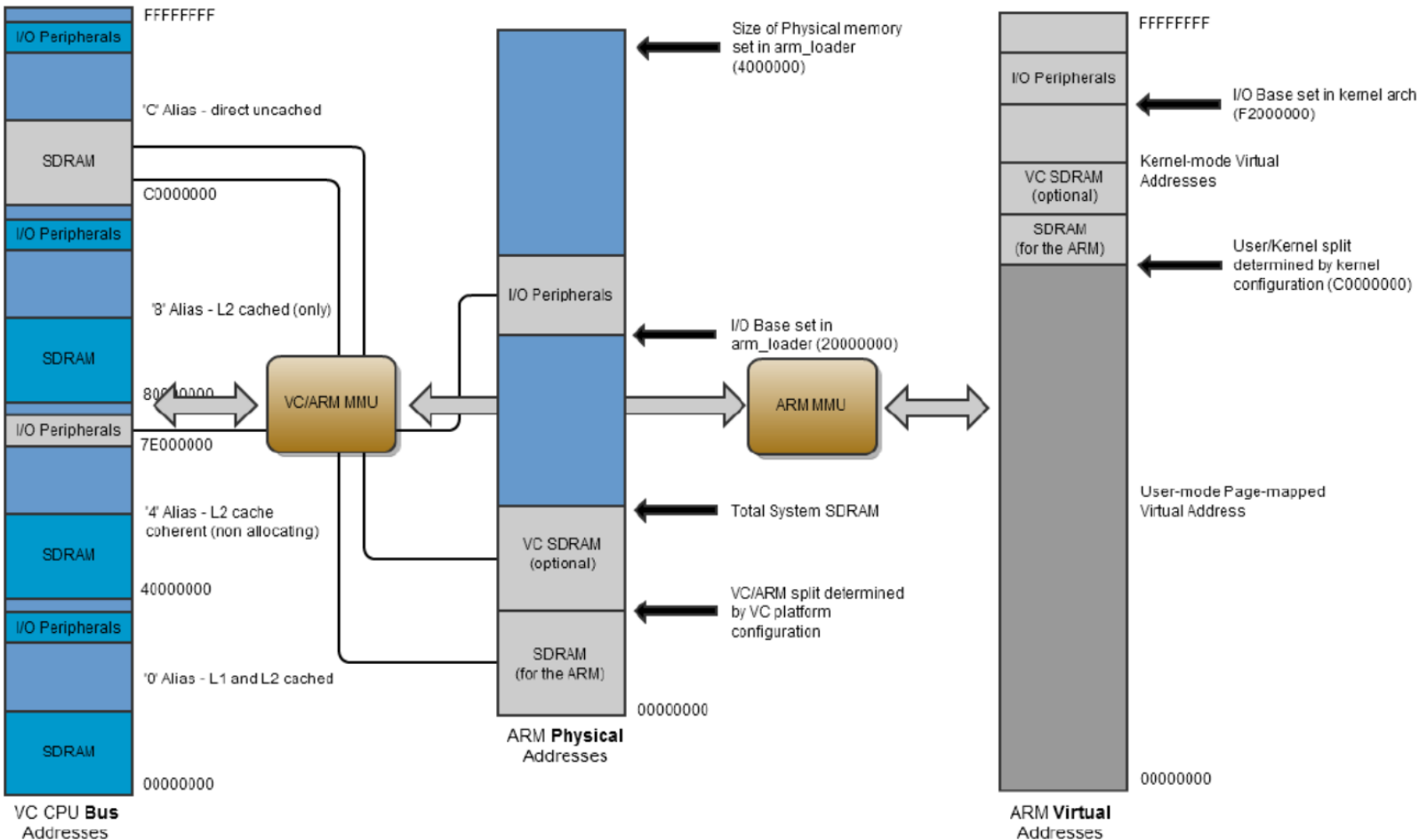
Η διαδικασία για να ξεκινήσουμε τον kernel στο Rasberry Pi :

- Θα χρειαστούμε ένα crosscompiler για να μπορούμε να κάνουμε complie το αρχείο μας για την αρχιτεκτονική του rasberry. Θα χρησιμοποίησουμε Gcc-Arm-Compiler
- Το harware ψάχνει στο /boot directory της SD κάρτας το αρχείο kernel8.img το οποίο περιέχει τον kernel μας σε binary μορφή.
- Φορτώνει τον πυρήνα στην κατάλληλη διεύθυνση που για αρχιτεκτονικές 64-bit είναι 0x80000.

Contents

Memory Mapped IO

Τα περιφερειακά δεν γράφουν σε κανονικούς registers, αλλά σε θέσεις μνήμης



Contents

Peripherals

Uart 1

Uart 1

Χρειαζόμαστε την διεπαφή με την κονσόλα:

1. Πρέπει να θέσουμε το CLK_Speed σε σταθερή τιμή 250 Mhz για να αποφύγουμε το skewing(στο config file του boot partition).
2. Θέτουμε τα GPIO pins 14,15 στις λειτουργίες TXD1,RXD1 αντίστοιχα.
3. Απενεργοποίηση των pull-up/pull-down αντιστάσεων.
4. Αρχικοποίηση των καταχωρητών.
5. Το braud rate είναι στα 115200 και επιλέγουμε το 8-bit ascii mode(extended).
6. Για να το τεστάρουμε, το πρόγραμμα screen χρησιμοποιήθηκε μαζί με το καλώδιο TTL.

Terminal Command : sudo screen /dev/ttyUSB0 115200

Peripherals

I2C

I2C-Master

Για την σειριακή επικοινωνία το πρωτόκολο είναι το Broadcom Serial Controller (BSC).

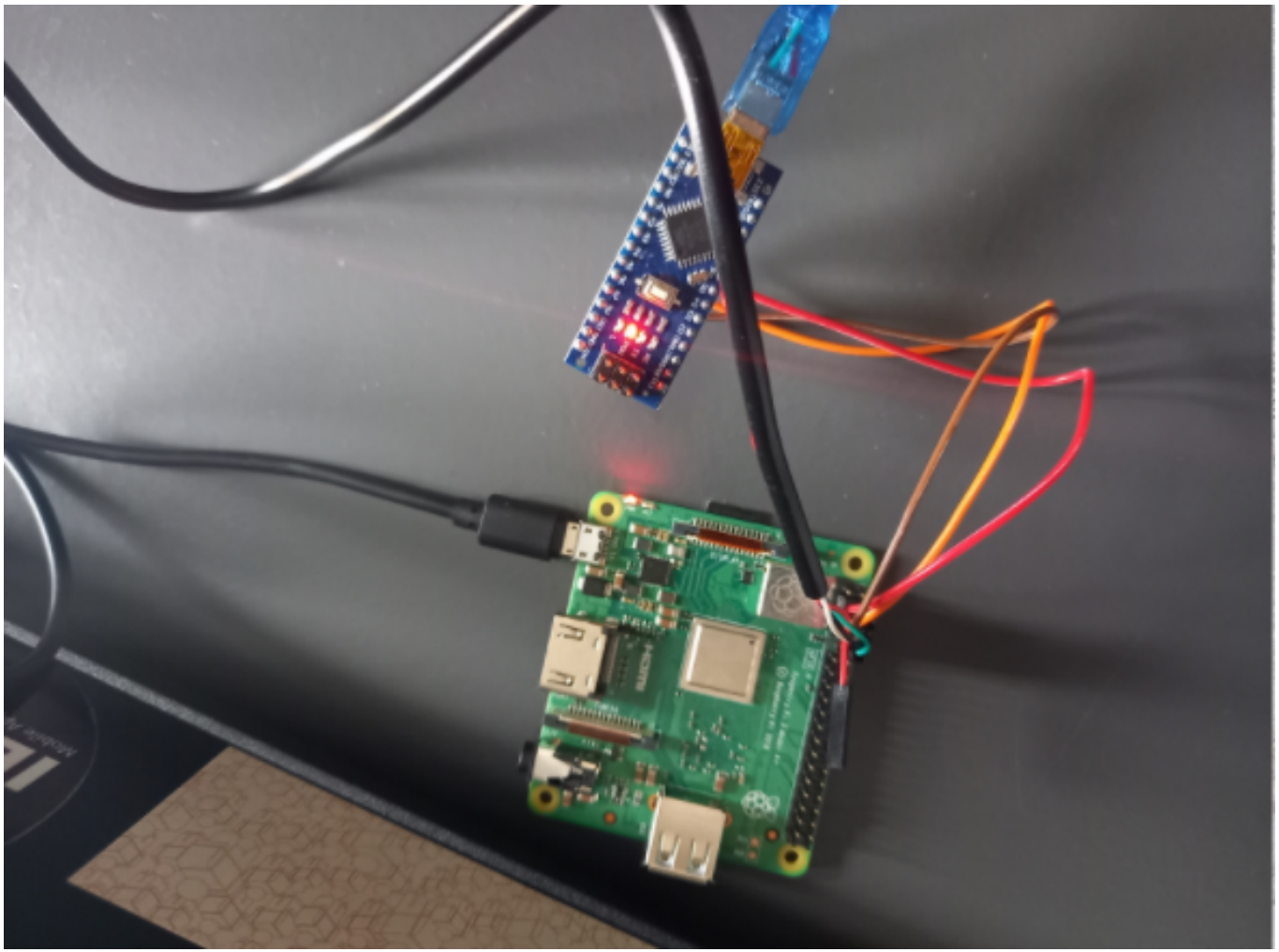
1. Θέτουμε κατάλληλη λειτουργία GPIO.
2. Θέτουμε κατάλληλη ταχύτητα στο I2C διαύλο.**(100khz)**
3. Απενεργοποίηση των pull-up/pull-down αντιστάσεων.
4. Γράφουμε στην διεύθυνση του slave.
5. Γράφουμε το αναμενόμενο αριθμό bytes.
6. Αρχικοποίουμε τους καταχωρήτες control και status.

I2C-Slave

Για τον slave σε αυτή την επικοινωνία επιλέξαμε ένα Arduino Nano.

1. Θέτουμε τα Pins σε pull-up $\rightarrow$ Λογικό 1.
2. Η ταχύτητα ρολογίου είναι πολύ χαμηλότερη από του master. **Αποτελεί πρόβλημα?**
3. Δεν σχετίζονται οι **Serial.begin != Wire.setClock**.
4. Περιμένουμε τον master να ζητήσει και να παραλάβει/στείλει data κάνοντας χρήση interrupts.

I2C-TTL



Contents

Interrupts

Στην οικογένεια των επεξεργαστών υπάρχουν 4 κατηγορίες διακοπών :

- Synchronous Interrupts
  - Διακοπές που προκαλούνται από εντολές, Software Interrupts.
- IRQ
  - Όταν προκληθεί εξωτερική διακοπή, χαμηλής προτεραιότητας.
- FIQ
  - Όταν προκληθεί εξωτερική διακοπή, υψηλης προτεραιότητας.
- System Error
  - Όταν προκληθεί σφάλμα από εξωτερικές διακοπές.

Interrupts

Οι διακοπές που υλοποιήσαμε ανήκουν στην κατηγορία των asychronous normal Interrupts (IRQs) και των sychronous interrupts.
Οι λειτουργίες που θελουμε να εφαρμόσουμε τις διακοπές είναι :

- Timer : IRQ που συμβαίνει σε χρονικα καθορισμένα διαστήματα
- System Call : Synchronous Interrupts όταν εκτελούμε την εντολή svc σε EL0 state

Ο κάθε handler καλείτε ανάλογα με τον είδος της διακοπής, με βάση των registers που ορίζονται στον interrupt controller και εκτελούνται αναλόγως.
Όλες οι διακοπές μπορούν να γίνουν masked και unmasked, για να μην επεμβαίνουν όταν τρέχουμε critical code.

Interrupt Life

Interrupt occurs

↓

Jump to address defined by vector table (what type of interrupt)

↓

Save the Register and Processor State

↓

if IRQ, the handler is called

↓

Find the type of IRQ according to Interrupt Controler Registers

↓

Call the appropriate handler(for timer interrupts the timer handler)

↓

After the interrupt has been handled register and processor state is restored and normal execution continued.

Timer

Ο timer εκτελείται περιοδίκα με ένα συγκεκρίμενο interval. Η αρχιτεκτονική μας παρέχει μέχρι 4 timers, εμείς χρησιμοποίησαμε μόνο το ένα.

Με κάθε χτύπο του ρολογίου το interval μειώνεται.

Όταν φτάσει το 0 τότε ένα IRQ εκτελείται. Ο IRQ halder γνωρίζει ότι είναι timer interrupt , λόγω του IRQ_PENDING_1 register, ο οποίος έχει μια συγκεκριμένη τιμή.

The timer handler is called

↓

It performs a task

↓

Interval is reset to original value

Contents

System Calls

Οι σύχρονες exceptions εκτελούνται με την εντολή svc.

Τα system calls είναι ο τρόπος με τον οποίο οι user processes μπορούν να επικοινωνούν με τον kernel.

Υλοποιήσαμε 7 system call:

- **Write** : takes an argument and prints it to the screen
- **Malloc** : allocate a new memory page for a new user process
- **Clone** : creates a new thread. Takes as argument the location of the stack of the new thread.
- **Exit** : Cleans up after a process has finished . Must be called after the end of all processes.
- **Cat** : prints at the screen pretty pictures of cats
- **Change_prior** : Sets the priority of an process.
- **Get_prior** : Gets the priority of an process.

System Call Life

System call happens

↓

Processor state is stored

↓

From the exception vector talbe the syscall handler is called

↓

Form the syscall table is decides which syscall happened

↓

Switch to EL1(kernel space)

↓

The function os the syscall is executed

↓

Switch back to EL0(User space). The processor state is restored and normal execution continues.

Contents

Task

Για την χρήση του scheduler χρειαζόμαστε διεργασίες.

Ορίζουμε task_struct το οποίο έχει τις απαραίτητες πληροφορίες που χρείαζεται μια διεργασία.

task_struct:

cpu_context	state	counter	priority	preemp_count

Priority:

High	Non-Preemptive
Middle	Preemptive
Low	Preemptive

Non-Preemptive: A process can't be schedule out in the middle of its execution.

Preemptive: A process can be schedule out in the middle of its execution.

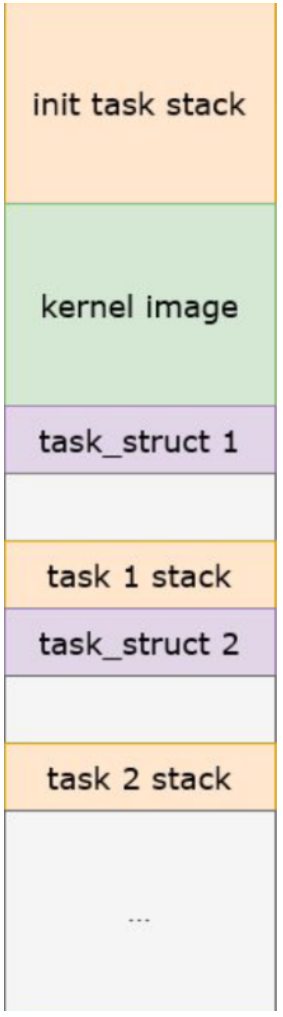
State:

Running	Zombie

Task Memory Allocation

- Δημιουργούμε συναρτήσεις **get_free_page()**, **free_page(p)**, που αποδίδουν στην διεργασία ένα memory page μεγέθους 4KB.
- Ορίζουμαι την συνάρτηση **copy_process()**, που λαμβάνει σαν όρισμα μια διεργασία και τα ορίσματα αυτής.

```
int task(array,priority){
sys_change_priority(priority);
count = 0;
while(count<5){
    for (i=0;i<len(array);i++){
        sys_call_write(sys_call_get_priority());
        sys_call_write(array[i]);
        delay(_);
        }
    count++;
    }
sys_exit_process();
}
```



Scheduler

Κάθε φορά που έχουμε ένα timer interrupt ή κάποιο process ολοκληρώθηκε (**sys_exit_process()**), καλούμε τον scheduler.

Η πολιτική του scheduler είναι:

- Προτίμαμε τις higher priority task για να τις κάνουμε schedule. (High-Middle-Low)
- Εάν current task εκτελεί τον scheduler ή το current task έχει High priority και δεν έχει ολοκληρωθεί, συνεχίζουμε με το ίδιο task (**High priority** → **Non-Preeptive**).
- Εάν έχουμε να επιλέξουμε 2 ή περισσότερα tasks με το ίδιο priority (**Middle-Low**), τα εναλλάσουμε κυκλικά.
- Για να αποφύγουμε το **Starvation** εάν ένα task δεν έχει γίνει schedule για ένα μεγάλο διάστημα τότε αλλάζει το priority του στην επόμενη βαθμίδα(Low → Middle, Middle → High).

Contents

Console

Υλοποίησαμε μία βασική console για την διεπαφή του χρήστη με τον λειτουργίκο.

Οι βασικές εντολές είναι :

- **help** : Εμφανίζει τις υπάρχουσες εντολές.
- **schedule** : Εκτελεί ένα demo για να παρατηρήσουμε όλες τις υπάρχουσες λείτουργίες του scheduler.
- **i2c** : Εκτελεί μια επικοινωνία i2c με ένα Arduino nano.
- **cat_1** : Εκτελεί το syscall cat(1).
- **cat_2** : Εκτελεί το syscall cat(2).

Contents

Functionality demo

Έχουμε προσθέσει ένα demo για να μπορέσουμε να παρατηρήσουμε την καθολική λειτουργία του scheduler αλλά και των system calls και του i2c.

Οι διεργασίες με την σείρα που εμφανίζονται για εκτέλεση:

- task(”12345”,L)
- task(”zqrty”,H)
- task(”abcde”,M)
- task(”rtyui”,M)

Functionality demo

Θα παρατηρήσουμε και στο demo ότι:

- Αρχίζει η εκτέλεση του Low task
- Έρχεται το High priority task,δρομολογείται από τον scheduler και παρά τα timer interrupts, δεν γίνεται schedule out .
- Στην συνέχεια, αρχίζουν να εκτελούνται διαδοχικά και επαναλαμβανόμενα τα 2 Middle Priority task.
- Λογώ του μεγάλου χρόνου που δεν έχει γίνει reschedule το Low priority task, αλλάζει το priority του σε Middle , και ακολουθεί την λογική των Middle priority task.

Links

Github repository

Google Drive