Construction d'un OS embarqué

MI074

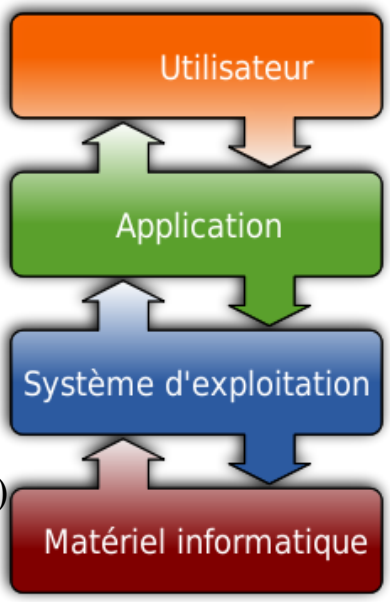
Plan

- objectif du module
- forme des cours et des TME
- difficultés
- plateforme matérielle
- Système construit

Définition d'un OS *(wikipedia)*

http://fr.wikipedia.org/wiki/Système_d'exploitation

Le **système d'exploitation**, ... est l'ensemble de programmes central d'un appareil informatique qui sert d'interface entre le matériel et les logiciels applicatifs.



L'OS visé est destiné à être embarqué

- faible empreinte mémoire
- mono-application multi-tâches
- bibliothèques réduites
- noyau monolithique (tous les services système fonctionnent en mode super-utilisateur)

Objectif du module

- Cours pratique, pas un cours sur les OS
- Partir de rien (ou presque) pour arriver à un embryon d'OS
- Travail en équipe de 4 (2x2) pour équilibrer les forces
  - les personnes d'un groupe doivent se connaitre
  - le niveau de C des groupes doit être homogène
  - un membre par groupe doit avoir suivi le module noyau
- il y a des intersections avec le module Multi
- https://www-soc.lip6.fr/trac/sesi-ose

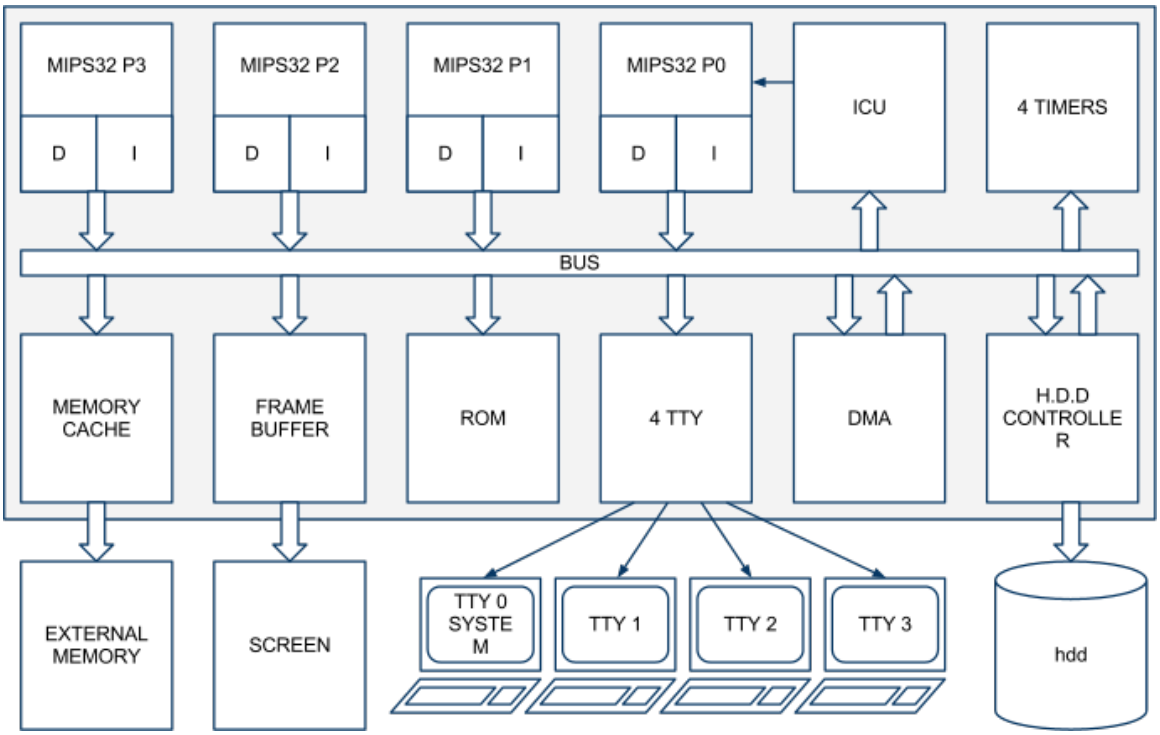
Format du module

- Chaque cours présente l'étape à franchir et les difficultés
- Description de l'API
  - des squelettes de code (voire des bouts de code)
  - la procédure de test
- L'ensemble des TME constitue un espèce de projet
  il faut que tous les groupes avancent au même rythme
- L'évaluation portera en grande partie sur le travail en TME

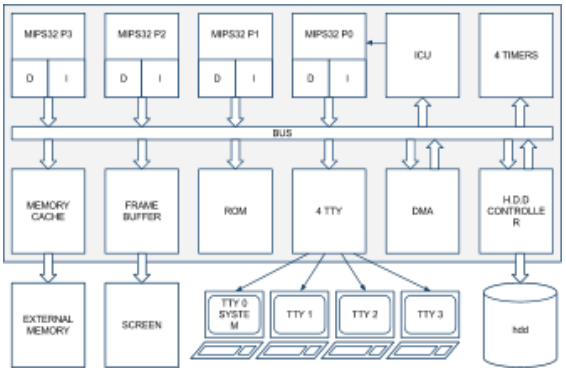
Difficultés

- Le travail en équipe
- Le langage C et les outils de développement
- La gestion du parallélisme de threads
- Le debug du code système sur une plateforme externe
  - Il est imprudent d'écrire du code sans avoir réfléchi
    vraiment car la durée du débug peut exploser
  - moyens: printf, gdb, trace d'exécution des processeurs

La plateforme matérielle



Composants de la plate-forme



- Type TSAR 1 seul cluster
  - 4 x MIPS32 + caches L1 I+D WT (Instruction et Data : 2 x 16ko 4 voies).
- Un inter-connect VCI (Virtual Component Interconnect)
- Une ROM pour le système
- Un cache mémoire (memory cache 256ko 16 voies) WB
- Un contrôleur de disque (block device)
- Un contrôleur TTY 4 terminaux
- Un terminal graphique 512x512 (frame buffer) en couleur :-)
- Un contrôleur de timers programmables 4 timers
  - timer 0 vers l'ICU, timers 1 à 3 vers les processeurs 1 à 3
- Un contrôleur DMA
- Un ICU : concentrateur d'interruptions
  - relié au processeur 0
    - 1 timer, les TTY, le DMA, le BlockDevice

Comportement des caches

- La lecture d'une donnée dans un segment caché est d'abord recherché dans le cache du processeur.
- En cas de hit, le contrôleur du cache fournit la donnée au processeur évitant ainsi de réaliser un accès à la mémoire.
- En cas de miss, le contrôleur du cache réalise la mise à jour d'une ligne de cache avant de fournir la donnée au processeur.
- L'écriture d'une donnée dans un segment caché est toujours envoyée au cache mémoire et le ligne du cache du processeur est mise à jour, si elle y est présente.
- Si une ligne de cache est modifiée dans le cache mémoire et que cette ligne est recopiée dans plusieurs caches de processeur alors ces caches sont mis à jour (ou invalidé).

Espace mémoire

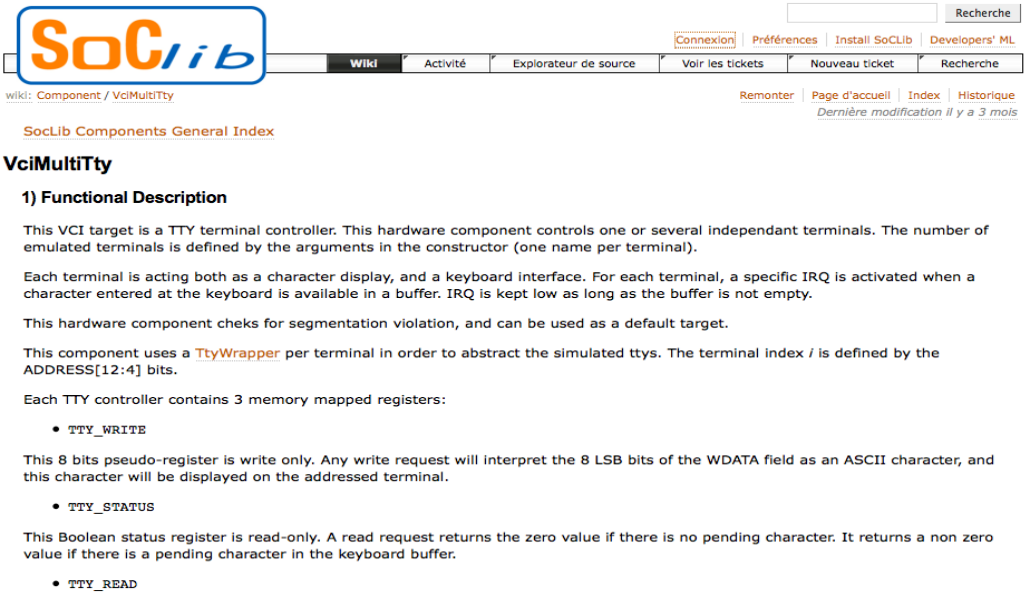
- Espace User
  - text
  - data
- Espace Kernel
  - ktext
  - kdata
  - boot
  - ktext_LMA
  - kdata_LMA
  - devices
    - timer
    - icu
    - dma
    - block device

Périphériques		Segments dans la ROM		Segments dans la RAM	
TIMER_BASE	0xd3200000	KTEXT_LMA_BASE	0xbf800000	RAM_BASE	0x7F400000
TIMER_SIZE	0x00000080	KTEXT_LMA_SIZE	0x00020000	RAM_SIZE	0x01000000
ICU_BASE	0xd2200000	KDATA_LMA_BASE	0xbf820000	KTEXT_BASE	0x80000000
ICU_SIZE	0x00000020	KDATA_LMA_SIZE	0x00020000	KDATA_BASE	0x80020000
DMA_BASE	0xd1200000	UTEXT_LMA_BASE	0xbf840000	KDATA_SIZE	0x003E0000
DMA_SIZE	0x00000014	UTEXT_LMA_SIZE	0x00060000	USR_TEXT_BASE	0x7F400000
TTY_BASE	0xd0200000	UDATA_LMA_BASE	0xbf8A0000	USR_DATA_BASE	0x7F460000
TTY_SIZE	0x00000040	UDATA_LMA_SIZE	0x00020000	USR_DATA_SIZE	0X00B9F000
BD_BASE	0xd5200000	BOOT_BASE	0xbfc00000		
BD_SIZE	0x20	BOOT_SIZE	0x00001000		

Description des périphériques

https://www.soclib.fr/trac/dev/wiki/Component

pour le TTY :
https://www.soclib.fr/trac/dev/wiki/Component/VciMultiTty



SoCLib

Recherche

Connexion | Préférences | Install SoCLib | Developers' ML

Wiki | Activité | Explorateur de source | Voir les tickets | Nouveau ticket | Recherche

wiki: Component / VciMultiTty

Remonter | Page d'accueil | Index | Historique

Dernière modification il y a 3 mois

SocLib Components General Index

VciMultiTty

1) Functional Description

This VCI target is a TTY terminal controller. This hardware component controls one or several independant terminals. The number of emulated terminals is defined by the arguments in the constructor (one name per terminal).

Each terminal is acting both as a character display, and a keyboard interface. For each terminal, a specific IRQ is activated when a character entered at the keyboard is available in a buffer. IRQ is kept low as long as the buffer is not empty.

This hardware component cheks for segmentation violation, and can be used as a default target.

This component uses a TtyWrapper per terminal in order to abstract the simulated ttys. The terminal index *i* is defined by the ADDRESS[12:4] bits.

Each TTY controller contains 3 memory mapped registers:

- `TTY_WRITE`

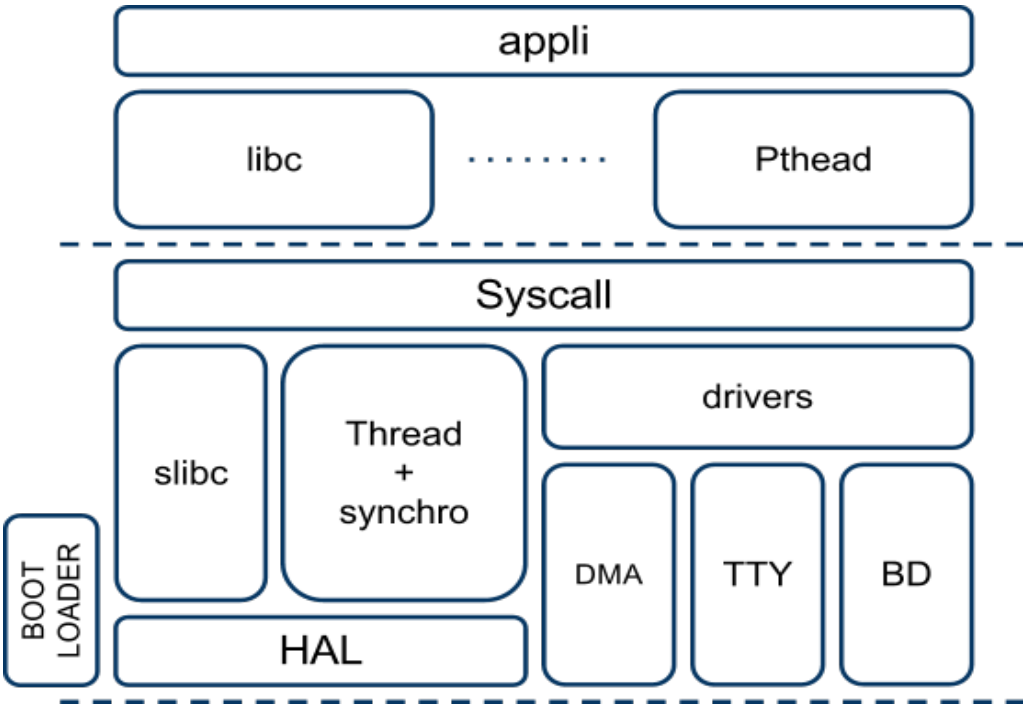
This 8 bits pseudo-register is write only. Any write request will interpret the 8 LSB bits of the WDATA field as an ASCII character, and this character will be displayed on the addressed terminal.

- `TTY_STATUS`

This Boolean status register is read-only. A read request returns the zero value if there is no pending character. It returns a non zero value if there is a pending character in the keyboard buffer.

- `TTY_READ`

Architecture de l'OS final



Les modules

libc	printf, malloc, memcpy, strcmp, ...
pthread	pthread_create, pthread_mutex_init, ...
syscall	th_create, th_mutex, putc, getc, ...
slibc	printf, alloc, putc, getc, list, ...
thread	th_create, th_mutex, ...
drivers	open, read, write, close, ...
DMA, BD, TTY	open, read, write, close, ...
HAL	context_load, context_save, context_restore, ... interrupt_disable, interrupt_restore, ...

pthread_create

```
int                                 // 0 si OK
ptread_create (
   pthread_t  *tid,                 // identifiant
   pthread_attr_t *attr,            // pile, priorité, ordo
   void * (*fonction) (void *arg), // code de la tâche
   void * arg                       // param unique
);
```

Exemple

```
#include <stdio.h>
#include <pthread.h>
pthread_t tid0;
char *str = "bonjour";

void * t0 (void *arg)
{
    while (1) {
        puts( (char *)arg);
    }
}

int main()
{
    if (pthread_create(&tid0, 0, NULL, t0, (void *)str))
        perror("impossible");
   // while (1);
}
```