Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

CSEL1 – Construction Systèmes Embarqués sous Linux

Optimisation: Performances

HES-SO//Master TIC/TIN 2020

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Daniel Gachet – HEIA-FR – Télécommunications

Contenu

- Introduction
- Mesure du temps
- Outils
- Compilateur
- Machines virtuelles
- Matériel et µP
- Perf
- Optimisation

MASTER OF SCIENCE IN ENGINEERING
MSE

Réf. support de cours de Wolfram Luithardt et proposition de Geoffrey Papaux

Introduction

MASTER OF SCIENCE IN ENGINEERING

MSE

Quel domaine ?

Temps de démarrage

Energie

Temps d'exécution

Temps de réaction

Variations

MASTER OF SCIENCE IN ENGINEERING
MSE

Introduction – Energie

Cortex-A50 Series: Faster, Better, Lower Power

Relative Comparison

18
16
14
12
10
8
6
4
2
0

Peak Performance

Energy

big.LITTLE Effect

More Performance

Less Energy

ARM big

ARM LITTLE

Cortex-A8 65 nm	Cortex-A8 45 nm	2x Cortex-A9 40 nm	4x Cortex-A9 32 nm	2x Cortex-A15 2x Cortex-A7 28nm	2x Cortex-A57 2x Cortex-A53 20nm

- Continuous improvement on performance and efficiency
- Innovation beyond process technology limitations

MASTER OF SCIENCE IN ENGINEERING

MSE

Introduction – Timing (réaction / variations)

File Control Setup Measure Analyze Utilities Help 4:05 PM

Acquisition is stopped.
25.0 MSa/s

1 On 1.00 V/div 2 On 2.00 V/div 3 On 4 On

H 5.00 µs/div 14.978180 µs T 1.80 V

Markers Scales

	X	Y
A——(2) =	-40 ns	-1.31 V
B---(1) =	25.524 µs	690 mV
Δ =	25.564 µs	2.00 V
1/ΔX =	39.117987 kHz	

MASTER OF SCIENCE IN ENGINEERING
MSE

Introduction – Temps de démarrage

Boot initial

Boot Optimisé

[s] 0 20 40 60 80

- UBoot start
- Kernel copy in ram
- Kernel decompression
- Kernel init
- FS mount
- Userland init

MASTER OF SCIENCE IN ENGINEERING
MSE

Introduction – Profiling

- **L'analyse des performances d'un logiciel est aussi nommée « Profiling »**
- **Lorsque l'on souhaite optimiser une application logicielle, il faut tout d'abord savoir quelle partie optimiser**
- **On cherche à identifier les «hot spots»**
- **On s'intéresse principalement à connaître**
  - Le temps d'exécution
    - *où notre programme passe le plus de temps ?*
  - Le nombre d'exécutions
    - *quelles fonctions sont exécutées le plus souvent ?*
  - L'arbre d'appel des fonctions (call-graph)
    - *quelle fonction appelle quelle fonction ?*

MASTER OF SCIENCE IN ENGINEERING

MSE

Introduction – Granularité

- **Avec quelle granularité souhaite-t-on effectuer la mesure?**
- **Niveaux de mesure possible**
  - Application
  - Méthode/fonction
  - Ligne de code en langage évolué
  - Instruction machine

MASTER OF SCIENCE IN ENGINEERING
MSE

Introduction – Stratégies

- **Différentes stratégies peuvent être mises en place**
  - Instrumentation
  - Statistique

- **Instrumentation**
  - Utilisation des outils du système d'exploitation et du compilateur
  - Mesure manuelle du temps avec la lecture d'horloges haute résolution
  - Mesure manuelle du temps avec un oscilloscope
  - Instrumentation dans une «machine virtuelle»

- **Statistique**
  - Le programme fonctionne sans modification
  - Un outil va l'échantillonné à intervalle régulier
    - *En utilisant les propriétés du microprocesseur*
    - *En échantillonnant sa pile d'appels*

Introduction – Optimisation

- **Une fois les « hot-spots » identifiés, plusieurs pistes peuvent être suivies pour optimiser les performances de l'application**
  - Choix d'algorithmes plus appropriés
  - Emploi plus judicieux des bibliothèques à disposition
  - Meilleure utilisation des options de compilation
  - Adaptation du code aux propriétés du microprocesseur
  - Changement du langage de programmation
  - Changement de la plateforme

Mesure du temps

MASTER OF SCIENCE IN ENGINEERING
MSE

Mesure du temps – Unix times

- time est un outil Linux très simple permettant d'indiquer le temps d'exécution d'une application.
- Exemple

```
# time ./clock 1000000 > /dev/null
real  0m 1.03s  // temps total
user  0m 1.02s  // temps CPU utilisé en mode user
sys   0m 0.00s  // temps CPU utilisé en mode system
```

- Cet outil est basé sur la méthode « times() » fournit par la bibliothèque « times.h ».
- Sa simplicité pâtit d'une résolution souvent insuffisante. Le temps retourné est donné en nombre de cycles d'horloge système, laquelle peut varier fortement selon les performances du µP mis en œuvre.

MASTER OF SCIENCE IN ENGINEERING

MSE

Mesure du temps – Horloges (timers)

- **Linux dispose d'horloges à haute résolution. Celles-ci sont disponibles par la bibliothèque « time.h »**
- **Opérations**
  - Lire l'horloge (syscall: clock_gettime)
  - Lire la résolution (syscall: clock_getres)
- **Le développeur peut modifier l'application pour y ajouter des lectures de ces horloges et ainsi mesurer le temps d'exécution de parties ciblées de code.**
- **Avantages**
  - Simple à mettre en œuvre
  - Grand nombre de points de mesure possible
- **Inconvénients**
  - Il faut connaître les parties du code à mesurer
  - Impacte le comportement de l'application

MASTER OF SCIENCE IN ENGINEERING
MSE

Mesure du temps – Horloges – Exemple

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <time.h>

static void do_something() {}

int main()
{
    struct timespec t1;
    struct timespec t2;

    clock_gettime(CLOCK_MONOTONIC, &t1);
    do_something();
    clock_gettime(CLOCK_MONOTONIC, &t2);

    // compute elapsed time in nano-seconds
    long long int delta_t = (long long)(t2.tv_sec - t1.tv_sec) * 1000000000
                          + t2.tv_nsec - t1.tv_nsec;
    printf ("elapsed time: %lld [ns]\n", delta_t);

    return 0;
}
```

Mesure du temps – Horloges – Interface

- **Sous Linux la bibliothèque <time.h> propose 3 horloges distinctes**
  - CLOCK_REALTIME
    - Temps actuel synchronisé avec NTP
  - CLOCK_MONOTONIC
    - Temps monotone, temps absolu depuis un point de départ non fixé
  - CLOCK_PROCESS_CPUTIME_ID
    - Temps utilisé par le processus

- **Le temps est donné en nanoseconds à l'aide de la structure suivante**

```
struct timespec {
  time_t tv_sec;          // seconds
  long   tv_nsec;         // nanoseconds
};
```

MASTER OF SCIENCE IN ENGINEERING

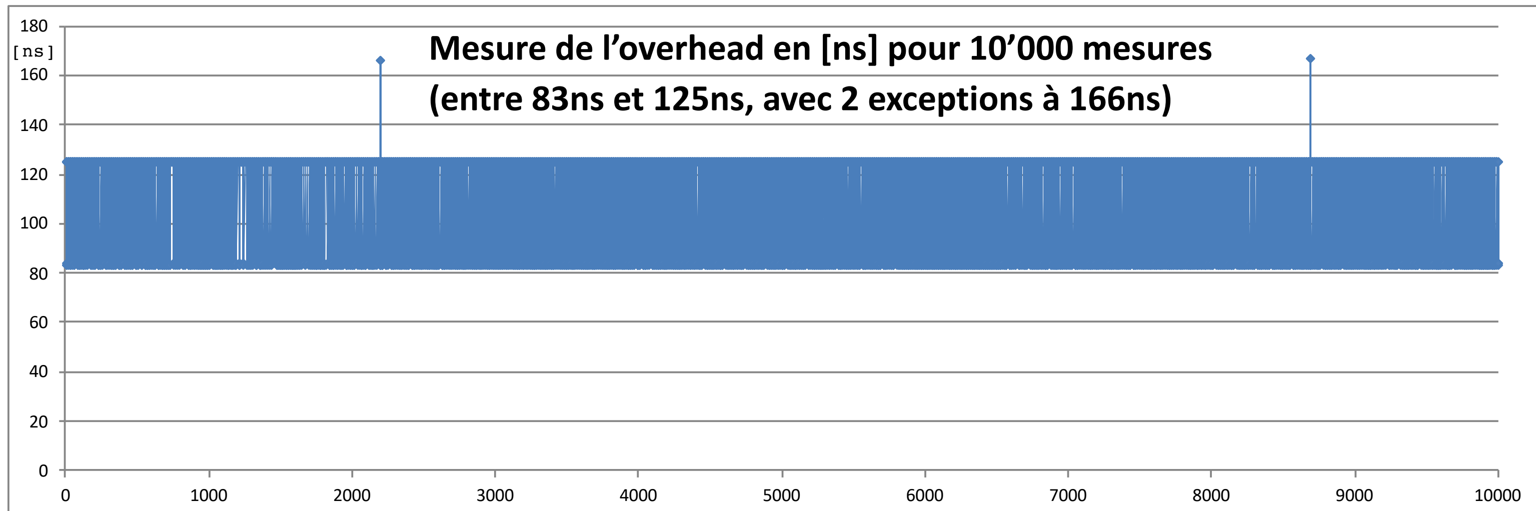
MSE

Mesure du temps – Horloges – Coût de la mesure

- Il est important de noter que la mesure du temps a également un coût (overhead). Pour obtenir ce coût, il suffit de réaliser la mesure suivante

```
struct timespec t1, t2;
clock_gettime(CLOCK_MONOTONIC, &t1);
clock_gettime(CLOCK_MONOTONIC, &t2);
long long overhead = (long long)(t2.tv_sec - t1.tv_sec) * 1000000000
                     + t2.tv_nsec - t1.tv_nsec;
```

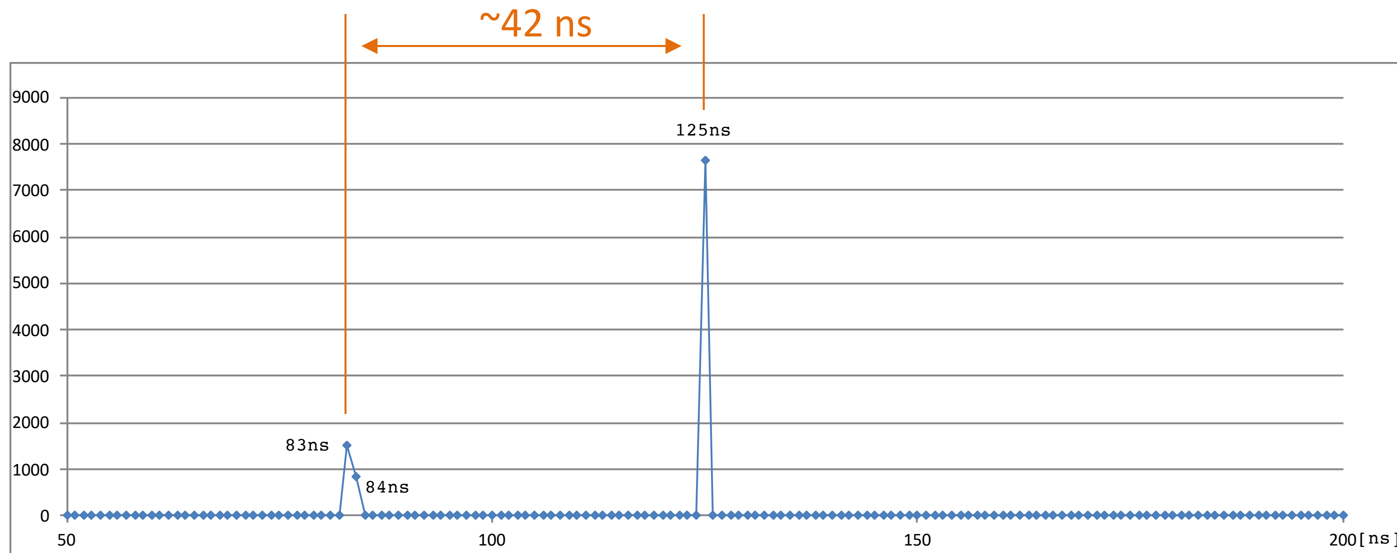
- Si l'on effectue cette mesure plusieurs fois on obtient les points suivants



MASTER OF SCIENCE IN ENGINEERING
MSE

Mesure du temps – Horloges – Coût de la mesure

- Sur la base des mesures précédentes, on peut obtenir la résolution de l'horloge



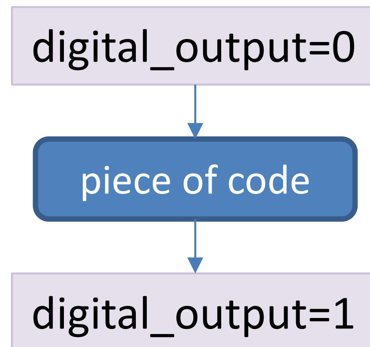
- Dans le cas du NanoPi Neo Plus2 ~ 42 ns (réellement une horloge à 24MHz)

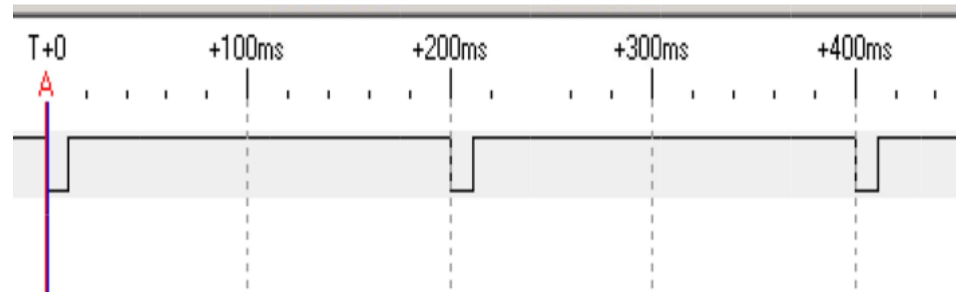
MASTER OF SCIENCE IN ENGINEERING

MSE

Mesure du temps – Oscilloscope

- **L'utilisation de sorties numériques et d'un oscilloscope peut s'avérer très utile si l'on souhaite une mesure très précise du temps**





- **Avantages**
  - Peu d'impact sur le comportement du logiciel
  - Grande précision
  - Mesure de la gigue possible
- **Inconvénients**
  - Mise en œuvre complexe
  - Peu de points de mesure possible

Mesure du temps – Oscilloscope (II)

- **L'utilisation de sorties numériques nécessite un accès aux ports d'entrées/sorties du µP**
- **2 chemins disponibles**
  - Utilisation des GPIO disponibles dans le sysfs (/sys/class/gpio/…)
    - *Simple à mettre en œuvre, mais peut être trop lent (nécessite un accès aux routines du noyau)*
  - Accès direct aux modules GPIO en mappant les contrôleurs en espace utilisateur (méthode mmap)
    - *Peu d'impacte sur le comportement du logiciel (que quelques cycles d'horloge), mais nécessite une très bonne compréhension du hardware*

Outils

Outils – Aperçu

- **Linux propose une large palette d'outils pour la mesure de performances**

Linux Performance Tools

Various, observability: sar /proc dstat dmesg
Various, static: sysctl /sys dmesg lshw journalctl lsmod
Various, tracing: capable

strace perf trace sysdig
opensnoop statsnoop syncsnoop
tcptop tcplife tcpconnect tcpaccept tcpconnlat tcpretrans
mysqld_qslower, ...
gethostlatency
App Config
ugc ucalls
ldd
ltrace
cachestat dcsnoop filetop fileslower mountsnoop

Operating System

Applications
System Libraries
System Call Interface

perf ftrace stap lttng bcc (BPF)
funccount funclatency stackcount kprobe uprobe argdist trace profile

ext4slower ext4dist btrfs* xfs* zfs*
lsof pcstat
df -h
mdadm lvm dmsetup
pidstat -d
multipath
mdflush

Linux Kernel

VFS
File Systems
Volume Manager
Block Device Int.
Sockets
TCP/UDP
IP
Ethernet
Scheduler
Virtual Memory
Clocksource
Device Drivers
Firmware

cpudist execsnoop runlat cpudist offcputime
latencytop
schedtool
mpstat powertop /proc/stat
top htop ps pidstat
vmstat slabtop free /proc/meminfo
memleak oomkill slabratetop
/sys/...
dmesg

biotop biosnoop biolatency bitesize
ss
hardirqs ttysnoop
tcpdump
ip route iptables
netstat iptraf-ng
storcli MegaCli

Hardware

intel_gpu_top intel_gpu_time
intel_gpu_\ frequency
GPU
/proc/cpuinfo cpuid lscpu
turbostat rdmsr
CPU 1
CPU Interconnect
tiptop perf pcm
numactl
Memory Bus
perf tiptop
I/O Bus
DRAM
llcstat profile
numastat
lstopo

Expander Interconnect
I/O Bridge
I/O Controller
iostat iotop blktrace
Disk
Disk
Swap
lsblk lsscsi blockdev smartctl fdisk -l
swapon -s /proc/swaps
lspci lsusb
Interface Transports
Network Controller
Port
Port
nicstat netstat ip
ethtool ifconfig
snmpget lldptool
FAN
sar -m FAN
Power Supply
ipmitool dmidecode

observability tools
static performance tools
perf-tools/bcc tracing tools
these can observe the state of the system at rest, without load
https://github.com/brendangregg/perf-tools https://github.com/iovisor/bcc
style inspired by reddit.com/u/redct
http://www.brendangregg.com/linuxperf.html 2017

MASTER OF SCIENCE IN ENGINEERING
MSE

Outils – strace

- **strace est un outil permettant de surveiller les appels système utilisés par une application sous Linux**

```
# strace -tt ./clock 1
01:43:36.802013 execve("./clock", ["./clock", "1"], 0xffffc8c547e0 /* 16 vars */) = 0
01:43:36.804217 brk(NULL)               = 0x89f2000
01:43:36.804476 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xffffb05d .
01:43:36.804795 faccessat(AT_FDCWD, "/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directo  .
01:43:36.805084 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or .
01:43:36.805252 openat(AT_FDCWD, "/lib/tls/aarch64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No su .
01:43:36.805444 newfstatat(AT_FDCWD, "/lib/tls/aarch64", 0xffffff027b800, 0) = -1 ENOENT (No such fil ..
01:43:36.805662 openat(AT_FDCWD, "/lib/tls/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file ..
01:43:36.805840 newfstatat(AT_FDCWD, "/lib/tls", 0xfffff027b800, 0) = -1 ENOENT (No such file or    ...
01:43:36.805965 openat(AT_FDCWD, "/lib/aarch64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such ...
01:43:36.806093 newfstatat(AT_FDCWD, "/lib/aarch64", 0xfffff027b800, 0) = -1 ENOENT (No such file   ...
01:43:36.806288 openat(AT_FDCWD, "/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
01:43:36.806450 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\350\365\1\0\0\0\0\0"..., ...
01:43:36.806634 fstat(3, {st_mode=S_IFREG|0755, st_size=1283888, ...}) = 0
01:43:36.806778 mmap(NULL, 1356392, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xffff  ...
01:43:36.806937 mprotect(0xffffb0591000, 61440, PROT_NONE) = 0
......
01:43:36.809857 write(1, "time=0'000'001 ns\n", 18time=0'000'001 ns
) = 18
01:43:36.810015 write(1, "292\n", 4292
)    = 4
01:43:36.810183 exit_group(0)           = ?
01:43:36.810781 +++ exited with 0 +++
#
```

- **Il est important de noter que strace influence énormément le comportement de l'application**

MASTER OF SCIENCE | IN ENGINEERING

MSE

Outils – strace (II)

- **strace propose un grand choix d'options, ici une autre façon de mesurer le temps**

```
# strace -c ./clock 100000
...
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    1.643296          16    100002           write
  0.00    0.000000           0         1           ioctl
  0.00    0.000000           0         1         1 faccessat
  0.00    0.000000           0         5         4 openat
  0.00    0.000000           0         1           close
  0.00    0.000000           0         1           read
  0.00    0.000000           0         3         3 newfstatat
  0.00    0.000000           0         2           fstat
  0.00    0.000000           0         1           sched_setaffinity
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         4           mmap
  0.00    0.000000           0         4           mprotect
------ ----------- ----------- --------- --------- ----------------
100.00    1.643296                100029         8 total
#
```

MASTER OF SCIENCE IN ENGINEERING | MSE

Outils – ltrace

- **ltrace est un outil permettant de surveiller les appels faits par une application à des méthodes de bibliothèques partagées (shared libraries)**

```
# ltrace -tt ./clock 1
00:04:31.538829 __libc_start_main([ "./clock", "1" ] <unfinished ...>
00:04:31.549903 sched_setaffinity(0, 128, 0xffff2218f08, 0xffff2218f08)        = 0
00:04:31.550871 printf("clocks_per_sec=%ld\n", 1000000clocks_per_sec=1000000
)                      = 23
00:04:31.552707 atol(0xffff2219e96, 0, 1, 0)                                     = 1
00:04:31.553544 clock_getres(4, 0xffff2218ef8, 10, 0xffff2219e97)                = 0
00:04:31.554208 printf("time=%ld'%03ld'%03ld ns\n", 0, 0, 1time=0'000'001 ns
)                 = 18
00:04:31.555774 clock_gettime(4, 0xffff2218eb0, 1, 0)                            = 0
00:04:31.556448 clock_gettime(4, 0xffff2218eb0, 0x20f342eff9be1b74, 0x20f3bd10726438f8) = 0
00:04:31.557182 clock_gettime(4, 0xffff2218eb0, 0x20f342eff9be1b74, 0x20f3bd10726438f8) = 0
00:04:31.557924 clock_gettime(4, 0xffff2218ea0, 0x20f342eff9be1b74, 0x20f3bd10726438f8) = 0
00:04:31.558599 printf("%lld\n", 678875678875
)                                       = 7
00:04:31.560006 +++ exited (status 0) +++
#
```

- **Là également, il est important de noter que ltrace influence passablement le comportement de l'application**

MASTER OF SCIENCE IN ENGINEERING
MSE

Outils – ltrace (II)

- ltrace propose aussi un grand choix d’options, ici une autre façon de mesurer le temps

```
# ltrace -c ./clock 1000 > /dev/null
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 57.09    0.910034      910034         1 __libc_start_main
 28.43    0.453122         226      2002 clock_gettime
 14.40    0.229538         229      1002 printf
  0.03    0.000452         452         1 exit_group
  0.02    0.000384         384         1 sched_setaffinity
  0.02    0.000245         245         1 atol
  0.01    0.000238         238         1 clock_getres
------ ----------- ----------- --------- --------------------
100.00    1.594013                  3009 total
#
```

MASTER OF SCIENCE IN ENGINEERING
MSE

Outils – ftrace

- **ftrace est un outil pour l'analyse d'un système mettant en œuvre une instrumentation au niveau noyau Linux**
- **ftrace permet de tracer à la µs les événements suivants**
  - Appels système
  - Fonctions de traitement d'interruptions
  - Fonctions d'ordonnancement
  - Piles de protocoles réseau
- **Une large documentation est disponible dans le noyau**
  https://www.kernel.org/doc/Documentation/trace/ftrace.txt
  https://www.kernel.org/doc/Documentation/trace/events.txt
- **Avant d'utiliser ftrace il faut préalablement configurer le noyau Linux**

```
$ make linux-menuconfig
   → kernel-hacking
         → Tracers
              → Kernel Function Tracer
              → Trace syscalls
```

MASTER OF SCIENCE IN ENGINEERING
MSE

Outils – ftrace - Utilisation

- **L'utilisation de ftrace passe par les étapes suivantes**
  - Sélectionner un traceur indiquant la fonctionnalité à tracer dans le noyau
  - Définir un filtre pour limiter les traces produites
    (exemple: le nom de la fonction à tracer)
  - Activer la trace en effectuant la commande :

    ```
    # echo 1 > /proc/sys/kernel/ftrace_enabled
    ```
  - Après enregistrement, désactiver la trace afin de limiter la taille des données par :

    ```
    # echo 0 > /proc/sys/kernel/ftrace_enabled
    ```
  - Exploiter les résultats

MASTER OF SCIENCE IN ENGINEERING | MSE

Réf. http://www.linuxembedded.fr/2011/03/introduction-a-ftrace/

Outils – ftrace - Utilisation (II)

- **L'utilisation de ftrace nécessite d'activer la fonctionnalité debugfs**

```
# mount -t debugfs none /sys/kernel/debug
```

- **On dispose alors du répertoire** `/sys/kernel/debug/tracing` **contenant entre-autres le fichier** `trace` **qui contient les résultats de l'instrumentation**

- **Les autres principaux fichiers de ce répertoire sont donnés ci-après :**
  - `available_tracers` contient les différents traceurs disponibles, soit nop, function, function_graph, …
  - `current_tracer` contient le traceur courant, donc une valeur de la liste précédente
  - `tracing_on` permet d'activer/désactiver les traces en écrivant les valeurs 1 ou 0
  - `available_events` contient les évènements traçables comme sched_wakeup. Ces évènements correspondent à des points de trace (ou *Tracepoints*) statiques ajoutés au noyau Linux, voir Documentation/trace/events.txt
  - `set_ftrace_pid` permet de tracer un processus donné par son PID

MASTER OF SCIENCE IN ENGINEERING

MSE

Réf. http://www.linuxembedded.fr/2011/03/introduction-a-ftrace/

Outils – OProfile

- **OProfile est également un outil de profilage de code sous Linux avec une très faible empreinte**
  - Il permet de profiler aussi bien un système complet qu'un sous ensemble, p.ex. les routines d'interruption, les pilotes de périphériques ou un processus
  - Pour effectuer l'instrumentation du système, il utilise les horloges système
  - Il est également capable d'utiliser les compteurs matériels si le processeur en disposant
- **Avec perf, OProfile est probablement un des outils de monitoring les plus utilisés sous Linux**
- **La documentation est disponible sur http://oprofile.sourceforge.net**

MASTER OF SCIENCE IN ENGINEERING
MSE

Compilateur

MASTER OF SCIENCE IN ENGINEERING

MSE

Compilateur – Introduction

- **Les compilateurs disposent d'options pour le profiling**
- **Ces options vont ajouter des instructions dans le fichier exécutable compilé**
- **Lors de l'exécution de l'application spécialement compilée, des données de profiling seront enregistrées dans des fichiers**
- **Des outils spécifiques permettront ensuite de traiter ces données**

- **Avantages**
  - Simple à utiliser (option de compilation)

- **Inconvénients**
  - Application plus lente
  - Comportement de l'application peut se trouver modifié

MASTER OF SCIENCE IN ENGINEERING
MSE

Compilateur – Gcov

- **Gcov est intégré au compilateur GCC de GNU**

- **Il permet de mesurer la couverture de code**
  - Combien de fois chaque ligne de code est exécutée ?
  - Et donc, quelles lignes ne sont jamais exécutées ?

- **Pour l'activer il suffit de compiler l'application avec les options** `-fprofile-arcs` **et** `-ftest-coverage`

- **Lors de l'exécution de l'application, les données seront enregistrées dans des fichiers** `.data`

- **L'analyse des données se fait avec l'application** `gcov` **qui permet d'avoir une vue annotée du code source**

MASTER OF SCIENCE IN ENGINEERING
MSE

Info: ne fonctionne pas sur la cible, doit impérativement être testé sur la machine virtuelle.

Compilateur – Gcov – Exemple

- Les sources se trouvent sur Moodle dans le fichier examples.tar sous gcov
- Pour générer l'application, l'exécuter et évaluer le résultat, utilisez les commandes ci-dessous

```
$ make
$ make gcov-generate
$ make gcov-read
```

```
        -:    1:#include <stdio.h>
        -:    2:
        1:    3:int main (void)
        -:    4:{
        1:    5:    int res = 1;
       17:    6:    for (int i=0; i<16; i++)
        -:    7:    {
       16:    8:        res *= 2;
        -:    9:    }
        -:   10:
        1:   11:    if(res < 10)
        -:   12:    {
    #####:   13:        printf("dead code\n");
        -:   14:    }
        -:   15:
        1:   16:    printf("res=%d\n", res);
        -:   17:
        1:   18:    return 0;
        -:   19:}
```

Compteurs de gcov

Les lignes de code jamais exécutées sont marquées par des #####

MASTER OF SCIENCE IN ENGINEERING

MSE

Compilateur – Gprof

- **Gprof est intégré au compilateur GCC de GNU**

- **Il fournit des informations plus détaillées que `gcov`, mais au niveau des fonctions**
  - Temps d'exécution de chaque fonction
  - Nombre d'appels
  - Peut générer un arbre d'appels (call-graph) des fonctions

- **Pour l'activer, il suffit de compiler et linker l'application avec l'option `–pg`**

MASTER OF SCIENCE IN ENGINEERING

MSE

Info: ne fonctionne pas sur la cible, doit impérativement être testé sur la machine virtuelle.

Compilateur – Gprof– Exemple

- Les sources se trouvent sur Moodle dans le fichier examples.tar sous gprof
- Pour générer l'application, l'exécuter et évaluer le résultat, utilisez les commandes ci-dessous

```
$ make
$ make gprof-generate
$ make gprof-read
...
```

```c
void func1(void)
{
      for(int i=0; i<0xfffffff; i++); // wait...
}

void func2(void)
{
      for(int i=0;i<0xfffffff;i++); // wait...
      func1();
}

int main(void)
{
      for(int i=0;i<0xfffffff;i++);
      func1();
      func2();
      return 0;
}
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 49.58      0.65     0.65        2   327.21   327.21  func1
 25.93      1.00     0.34                             main
 25.17      1.33     0.33        1   332.24   659.46  func2
```

MASTER OF SCIENCE IN ENGINEERING | MSE

Machines virtuelles

MASTER OF SCIENCE
IN ENGINEERING
MSE

Machines virtuelles – Introduction

- **L'instrumentation d'une application peut s'effectuer à l'aide d'une machine virtuelle**
  - Le fichier exécutable est lancé dans la machine virtuelle
  - La machine virtuelle intercepte le code avant qu'il soit exécuté sur le CPU
  - Elle le modifie à la volée pour y ajouter des instructions d'instrumentation
  - Cela permet de collecter énormément d'informations
    - Allocations mémoires,
    - Call graph,
    - …

- **Avantages**
  - Ne nécessite pas de compilation spéciale
  - Permets de récolter beaucoup d'informations

- **Inconvénients**
  - Impact sur les performances très grand
  - Le comportement de l'application peut se trouver modifié

Machines virtuelles – Valgrind

- **Valgrind est un outil permettant de faire de l'instrumentation de code à la volée**

- **C'est un framework comportant plusieurs outils**
  - memcheck
    - détection d'erreurs de mémoire
      (overflow / underflow, undefined variables, memory leaks, …)
  - callgrind
    - pour faire du profiling et compter le nombre d'exécutions
  - cachegrind
    - analyse de l'utilisation du cache
  - massif
    - analyse de l'utilisation du heap
  - helgrind
    - débogage d'application multi-threadée

MASTER OF SCIENCE IN ENGINEERING | MSE

Matériel et µP

MASTER OF SCIENCE
IN ENGINEERING
MSE

Matériel et µP – Introduction

- **Les processeurs modernes offrent une batterie de compteurs (hardware events counters) permettant de collecter et capturer de toute une série d'informations et d'événements de profiling**
  - Les instructions exécutés
  - Les cache misses (data, instruction)
  - Les branch mispredicted
  - Les cycles CPU gaspillés («stalled»)
  - Beaucoup, beaucoup d'autres…

- **Comme ces informations sont collectées et stockées par le µP, elles offrent une solution avec un très faible impact sur les performances globales des applications**

MASTER OF SCIENCE IN ENGINEERING

MSE

Matériel et µP – Performance Monitor Unit (PMU)

- **«PMU» est le nom donné au composant permettant de gérer les compteurs d'événements d'un processeur**
- **Chaque constructeur de processeurs propose un PMU différent, documenté généralement dans le manuel de l'architecture**
  - Les processeurs Intel [1] intègrent depuis longtemps un PMU
  - Les processeurs ARM [2] intègrent un PMU dans l'architecture ARMv7/ARMv8
- **Le PMU fournit une interface pour activer les différents compteurs d'événements**
  - Des instructions permettent d'activer les compteurs
  - Une fois le compteur activé, celui-ci commence à collecter les événements
  - Il suffit ensuite d'aller lire la valeur du compteur
  - Sur ARM, le PMU s'accède et se configure via le coprocesseur CP15, donc avec les instructions MCR / MRC

[1] Intel Software developer Manual (SDM),
section **PERFORMANCE MONITORING EVENTS**
https://www-ssl.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[2] ARM® Architecture Reference Manual ARMv7-A and ARMv7-R edition,
Chapitre **C12: The Performance Monitors Extension**
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.architecture.reference/index.html

Matériel et µP – Outils

- Il existe différents outils pour collecter des informations de profiling en utilisant les compteurs matériels
- Intel fournit un outil Intel® VTune™ Amplifier (payant)
- ARM n'a pas (encore) d'équivalent
- Le noyau Linux intègre aussi des outils: perf et oprofile
- Ces outils fonctionnent pour les processeurs Intel, AMD et également ARM

Perf

MASTER OF SCIENCE IN ENGINEERING | MSE

Perf – Introduction

- **Depuis la version 2.6.31, perf est intégré au noyau Linux**
- **Il se compose de**
  - Back-end «perf_events» intégré au noyau
  - Une application «perf» s'exécutant en espace utilisateur

- **Le code source de l'application est dans les sources de Linux**
  - linux/tools/perf
  - Documentation sous linux/tools/perf/Documentation/examples.txt

- **Il permet d'enregistrer des événements du PMU, mais également des «tracepoints» du noyau**

MASTER OF SCIENCE IN ENGINEERING

MSE

Perf – Fonctionnalités

- **Perf offre les fonctionnalités suivantes**
  - Génération d'arbres d'appel (call graph)
  - Collecte d'événements à partir des compteurs hardware disponible dans le PMU
  - Collecte de tracepoints du noyau Linux
  - Collecte d'événements en temps réel
  - Annotation de code source / instructions assembleur

- **Une documentation détaillée est disponible sur le wiki officiel (https://perf.wiki.kernel.org)**

MASTER OF SCIENCE IN ENGINEERING
MSE

Perf – Tracepoints du noyau

- **Un tracepoint est un appel de fonction placé à un endroit stratégique dans le code du noyau Linux**

- **Perf est capable d'activer les tracepoints**
  - Si le tracepoint est désactivé, celui-ci ne fera rien (si ce n'est gaspiller quelques cycles pour tester s'il est activé et utiliser quelques bytes dans l'exécutable)
  - S'il est activé, le tracepoint va notifier celui qui l'a configuré

- **Les tracepoints permettent de tracer**
  - Les appels système
  - Les événements TCP/IP
  - Les opérations du file system
  - Les changements de contexte de l'ordonnanceur
  - …

Perf – Application utilisateur

- L'application perf se décompose en plusieurs sous-commandes
- La liste complète peut être obtenue en invoquant perf sans argument

Sous-commande	Description
list	affiche la liste de tous les événements disponibles
stat	exécute une application et affiche des statistiques des compteurs d'événements
top	outil de profiling en temps réel similaire à la commande `top`, affichant les données des compteurs matériels
record	exécute une commande et enregistre des données de compteur dans un fichier de profiling nommé `perf.data`
report	lit un fichier `perf.data` créé avec la commande record et affiche les informations de profiling

Perf – List

```
$ perf list
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  cache-references                                   [Hardware event]
  cache-misses                                       [Hardware event]
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]
  ref-cycles                                         [Hardware event]

  cpu-clock                                          [Software event]
  task-clock                                         [Software event]
  ...
```

Perf – Stat

```
$ perf stat ./sudoku-resolver
 Performance counter stats for './sudoku-resolver':

      17200.968507 task-clock (msec)         #    0.999 CPUs utilized
             2,019 context-switches          #    0.117 K/sec
                21 cpu-migrations            #    0.001 K/sec
               150 page-faults               #    0.009 K/sec
    66,111,521,094 cycles                    #    3.843 GHz
    13,661,886,259 stalled-cycles-frontend   #   20.66% frontend cycles idle
   <not supported> stalled-cycles-backend
   166,918,528,666 instructions              #    2.52  insns per cycle
                                             #    0.08  stalled cycles per insn
    12,592,498,548 branches                  #  732.081 M/sec
       368,812,217 branch-misses             #    2.93% of all branches

      17.222153960 seconds time elapsed
```

MASTER OF SCIENCE IN ENGINEERING

MSE

Perf – Record et Report

```
$ perf record ./sudoku-resolver
[ perf record: Woken up 11 times to write data ]
[ perf record: Captured and wrote 2.669 MB perf.data (~116626 samples) ]
```

```
$ perf report
Samples: 69K of event 'cycles', Event count (approx.): 66377473404
 84.68%  sudoku-resolver  sudoku-resolver    [.] isAvailable
 15.02%  sudoku-resolver  sudoku-resolver    [.] fillSudoku
  0.08%  sudoku-resolver  [i915]             [k] gen6_read32
  0.05%  sudoku-resolver  [kernel.kallsyms]  [k] _raw_spin_lock_irqsave
```

Perf – Top

```
$ perf top -e cpu-clock
PerfTop: 1579 irqs/sec  kernel:99.9% exact: 0.0% [4000Hz cpu-clock], (all, 4 CPUs)
-------------------------------------------------------------------------------------

    99.73%  [kernel]          [k] arch_cpu_idle
     0.05%  [kernel]          [k] _raw_spin_unlock_irqrestore
     0.04%  [kernel]          [k] _raw_spin_unlock_irq
     0.03%  [kernel]          [k] tick_nohz_idle_exit
     0.01%  [kernel]          [k] __softirqentry_text_start
     0.01%  [kernel]          [k] copy_page
     0.01%  [kernel]          [k] tcp_write_timer_handler
     0.01%  [kernel]          [k] rcu_idle_exit
     0.01%  [kernel]          [k] __local_bh_enable_ip
     0.01%  [kernel]          [k] preempt_count_sub
     0.00%  [vdso]            [.] __kernel_clock_gettime
     0.00%  [kernel]          [k] put_ctx
     0.00%  perf              [.] 0x000000000008d2ac
     0.00%  [kernel]          [k] __schedule
     0.00%  [kernel]          [k] timespec64_add_safe
     0.00%  [kernel]          [k] el0_svc_common
     0.00%  libc-2.28.so      [.] __vsnprintf_chk
     0.00%  [kernel]          [k] unwind_frame
```

MASTER OF SCIENCE IN ENGINEERING
MSE

Perf – Résolution des symboles

- **Perf est capable de donner les informations de performances de manière symbolique, pour ceci il procède de la façon suivante**
  - Les compteurs hardware donnent l'adresse de l'instruction exécutée
  - Perf associe ces adresses aux lignes de code correspondantes
  - Les symboles de debug contenus dans l'exécutable sont utilisés
  - Il faut donc que notre application soit compilée avec les symboles de debug (option **-g** du compilateur)
  - Perf peut également résoudre les symboles au niveau du noyau

Perf – Note sur les compteurs

- **Les compteurs matériels affichés par perf list sont des compteurs génériques**
- **Cependant chaque processeur dispose d'une liste de compteurs propre à son architecture**
- **Perf effectue un mapping entre les compteurs d'une architecture donnée vers ses compteurs génériques**
- **Par exemple, le compteur cycles est mappé**
  - Processeur Intel:
    - UNHALTED_CORE_CYCLES (voir arch/x86/kernel/cpu/perf_event.c)
  - Processeur AMD:
    - CPU_CLK_UNHALTED
  - Processeur ARM:
    - voir arch/arm/kernel/perf_event_v7.c

Perf – Note sur la précision des événements

- **Perf est basé sur un échantillonnage par événement «event-based sampling»**
  - Cela signifie que seulement une partie des événements (et donc des instructions) sont enregistrés (dès que le compteur génère un overflow)
  - On peut configurer l'échantillonnage de deux manières, en spécifiant:
    - le nombre d'occurrences d'événements (période) ➔ option -c
    - le nombre de échantillons / secondes (fréquence) ➔ option -F
  - A cause de l'échantillonnage, on peut avoir des cas spéciaux ou notre application se trouve synchronisée avec la collecte d'événements de perf
    - On aura ainsi des résultats faussés, ce sera toujours la même instruction qui sera enregistrée au moment de l'overflow du compteur.

- **De plus le nombre de compteurs est limité par l'architecture**
  - Si on configure plus d'événements (option –e) que de compteurs, perf va s'occuper de faire du multiplexage entre les différents événements.

- **Plus d'informations sur le wiki de perf:**
  https://perf.wiki.kernel.org/index.php/Tutorial#Sampling_with_perf_record

Optimisation

MASTER OF SCIENCE IN ENGINEERING

MSE

Optimisation – Algorithmes et bibliothèques

- **Le choix des bons algorithmes et des bonnes bibliothèques mis en œuvre par une application a un impact majeur sur le comportement et les performances de cette dernière.**
  - Il est impératif que la complexité et évolutivité (scalability) des algorithmes soient en adéquation avec les contraintes imposées au système,
  - If faut contrôler la notation Big-O. L'emploi d'un container de type vecteur est tout à fait approprié pour des accès avec indices, mais peut s'avérer catastrophique si l'on doit faire des recherches aléatoires. Dans ce cas, un container de type set ou map serait probablement plus approprié.

- **Lors d'utilisation de boucles, il est également important de bien choisir le type de boucle ainsi que l'ordre d'évaluation des conditions, p.ex.**

```
for (int i=0; i<=expr(); i++) do_something();
```

  Cette implémentation peut s'avérer très gourmande suivant la complexité de « expr() » qui sera évalué à chaque itération. Voici une meilleure implémentation

```
for (int i=expr(); i>=0; --i) do_something();
```

- **Il en va de même avec les services de l'OS, des bibliothèques standard et des appels système.**

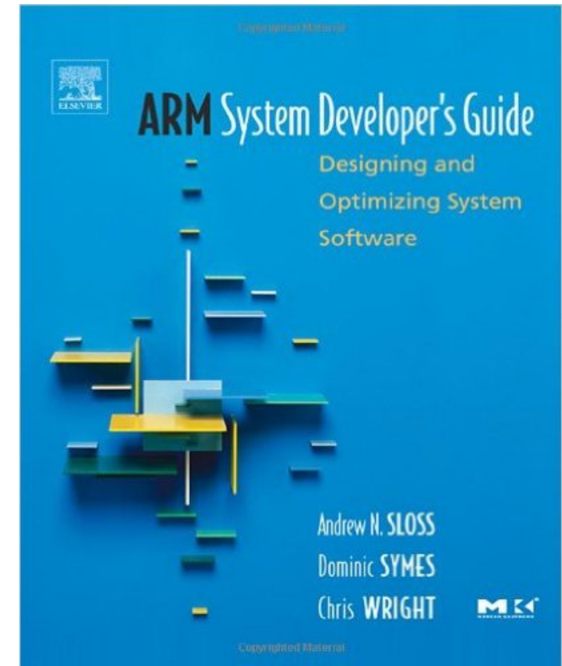
MASTER OF SCIENCE IN ENGINEERING
MSE

Optimisation – Options du compilateur

- Les compilateurs modernes offrent un grand d'options d'optimisation avec une granularité très fine.
- Ces options sont généralement regroupées en 3 ou 4 catégories, p. ex. avec le compilateur de GNU de -O0 à -O3, -O0 ne propose aucune optimisation, tandis que -O3 offre le niveau le plus élevé.
- Les options offrant une très forte optimisation sont optimales pour les performances, mais rendent le débogage des applications souvent très difficile.
- L'option -O2 est très couramment utilisée.
- GCC propose aussi une option -Og, laquelle devrait déjà offrir un bon degré d'optimisation, mais ne pas trop péjorer le débogage.

Optimisation – µP

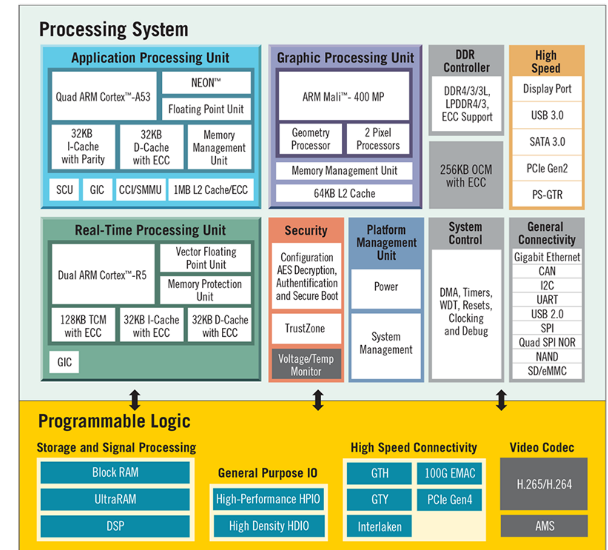
- **Les compilateurs génèrent normalement un code excessivement performant pour les diverses architectures de processeurs. Cependant dans certaines parties critiques du système il peut s'avérer très important de bien connaître les caractéristiques du µP et les conventions d'appel de fonction.**
- **P. ex. avec les processeurs ARM le code généré pour des variables de 32 bits est plus performant que celui 16 ou 8 bits.**
- **Les accès à la mémoire principale et à la mémoire cache ainsi que l'utilisation du pipeline vont également influencer les performances du système.**



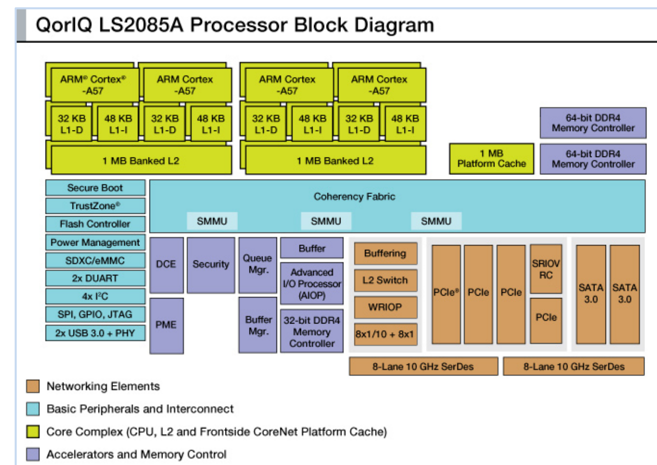
MASTER OF SCIENCE IN ENGINEERING
MSE

Optimisation – µP vs SoC

- Il existe actuellement une très grande variété de microprocesseurs
- Il est courant de trouver des machines multi-cœurs dans des systèmes embarqués modernes et exigeants en performances
- Ces nouvelles machines disposent aussi régulièrement de coprocesseurs et/ou contrôleurs spécialisés pour décharger la CPU, p.ex. GPU, NEON, crypto, …



Ref. http://linuxgizmos.com/16nm-zynq-soc-mixes-cortex-a53-fpga-cortex-r5/



http://www.freescale.com/products/arm-processors/qoriq-arm-processors/qoriq-ls2085a-and-ls2045a-multicore-communications-processors:LS2085A

MASTER OF SCIENCE IN ENGINEERING
MSE