LE PARALLELISME

Xavier NICOLAY 2016

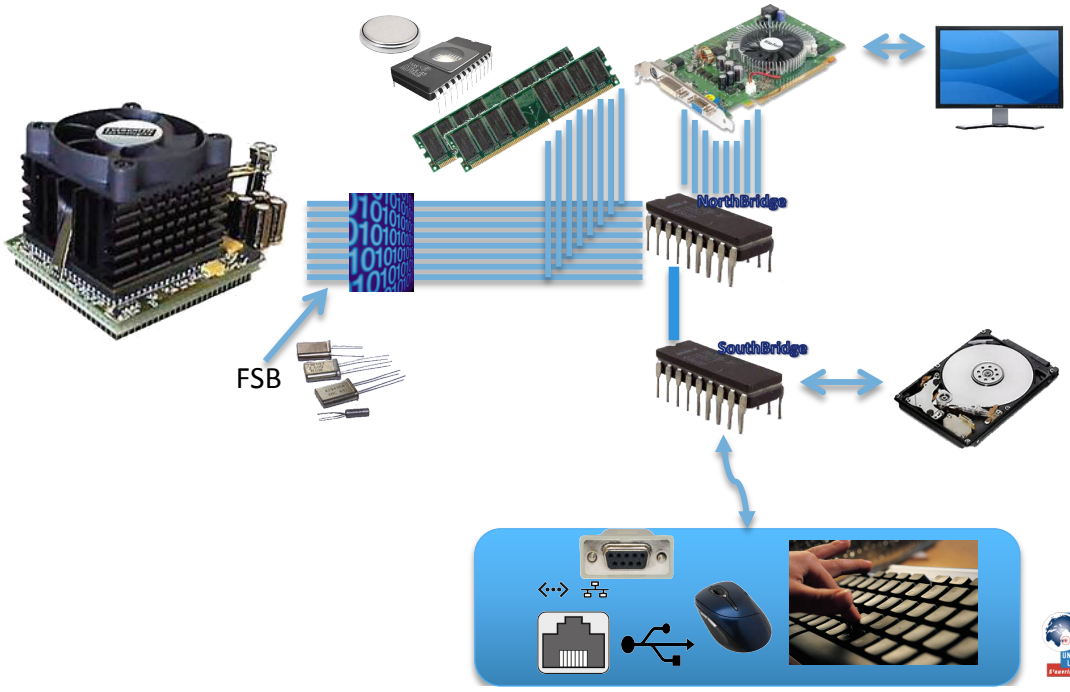
OBJECTIFS

- Comprendre les problématique de communications inter-processus
- être capable de créer un programme tirant parti des dernières innovations technologiques
- programmer à l'aide d'une librairie de threads en mode « pool »

L'ARCHITECTURE MATERIELLE

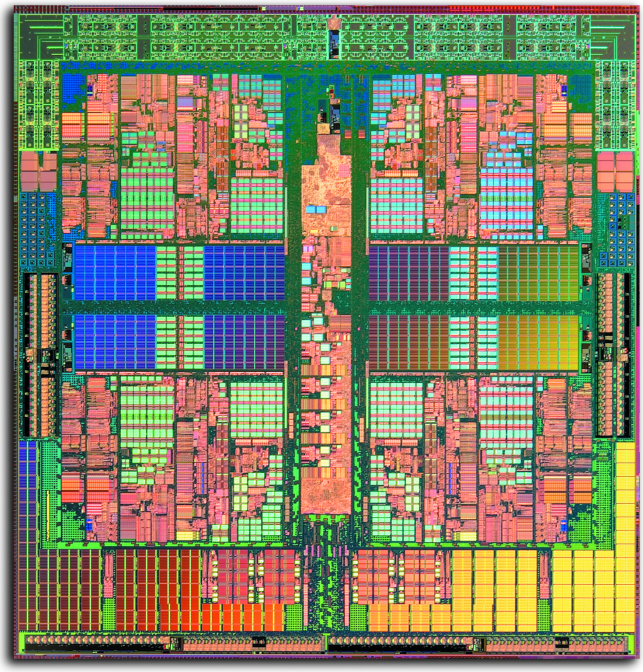
ARCHITECTURE MATERIELLE

Schéma d'un ordinateur



ARCHITECTURE MATERIELLE

Photo d'un microprocesseur Quad-Core



LE PARALLELISME

ARCHITECTURE MATERIELLE

CISC vs RISC

CISC : complex instruction set computer (Famille x86 et 680x0)

Avantages & Inconvénients

RISC : Reduced instruction set computer (Transputer, powerPC, MIPS, RS/6000, Sparc, Alpha, ARM …)
Caractéristiques….
Inconvénients & Avantages

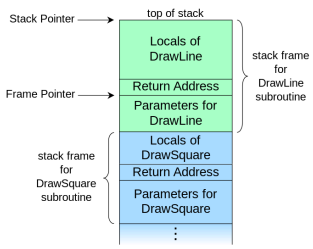
et actuellement ?

Nota : DSP & ASIC

LES REGISTRES PROCESSEURS

Processeurs: registres & StackPointer

Registres spécialisés : entiers, flottants
+ PC: PointeurCounter
+ PSW : Registre d'état
+ SP : Stack Pointer

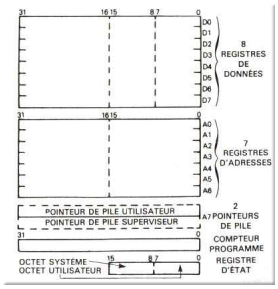


ALpha de DEC contient 32 registres (R0 à R31) de 64 bits pour la manipulation des entiers et 32 registres (F0 à F31) de 64 bits pour la manipulation des flottants

ARM 2 et 3 ont 27 registres 32 bits,
ARM 610 et suivants ont 31 registres 32 bits (dont 6 registres d'état).

MIPS dispose de 32 registres de 4 octets chacun

68000



SPARC v8

- %g0 à %g7
- %g0 vaut toujours 0
- %g5 à %g7 sont réservés

– %i0 à %i7 : recevoir des arguments
– %l0 à %l7 : registres locaux
– %o0 à %o7 : passer des arguments

i86

31	15	7	0
EAX	AX (AH)	AX (AL)	
EBX	BX (BH)	BX (BL)	
ECX	CX (CH)	CX (CL)	
EDX	DX (DH)	DX (DL)	
ESI	SI		
EDI	DI		
EBP	BP		
ESP	SP		

EFLAGS	FLAGS
EIP	IP

15	0
CS	
DS	
SS	
ES	
FS	
GS	

Processeurs: registres & StackPointer

Fenêtres de Registres

Fenêtres de registres

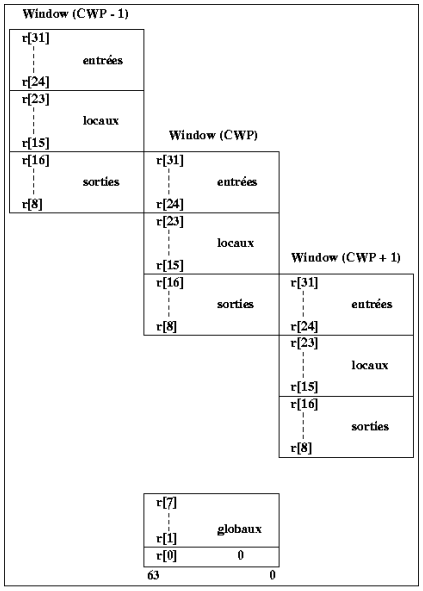


FIG. 4.1 – *Exemple de fenêtres de registres sur l'architecture SPARC-V9*

LES PROCESSUS

Programme en cours d'exécution

ξ(d'instructions) ← ROM / RAM
Espace d'adressage
Ressources

Etats particuliers
Zombie
Swappé
Préempté

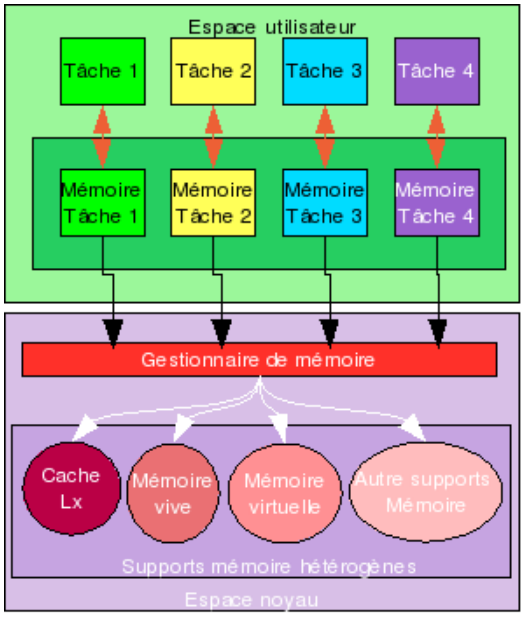
Espace
utilisateur vs noyau

Contexte

LE PARALLELISME

LES PROCESSUS

Espace d'exécution mémoire



AEF d'un processus

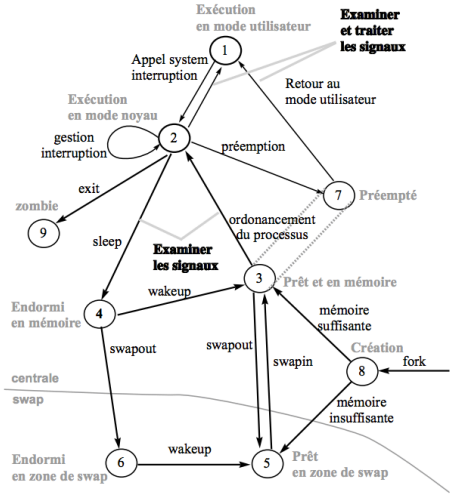


FIG. 6.8 – Diagramme d'état des processus

Dominique Revuz
http://www-igm.univ-mlv.fr/~dr/NCSPDF

Le Swap

- Pages vs Frame
- Mécanisme « swap »
- Swapfile vs swapdisk

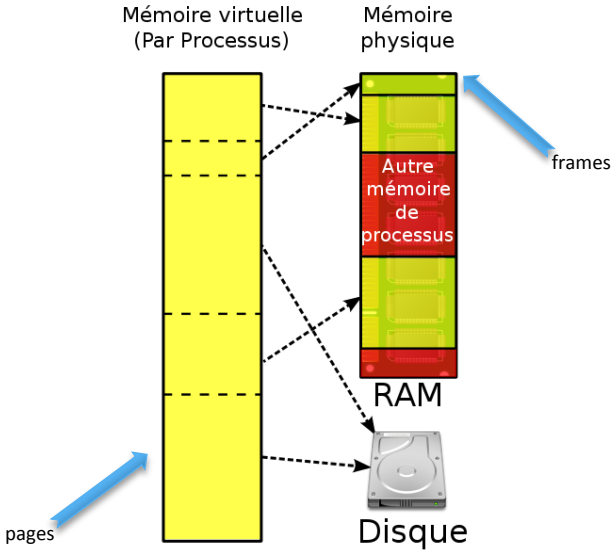


Schéma Wikipédia

HISTORIQUE

BSD : Berkeley System Distribution
socket (Internet, unix, raw) = stream, datagram, bpf

HISTORIQUE

SYS V :
mémoire partagée, les files de messages et les sémaphores

mutex : primitive de synchronisation

LE PARALLELISME

PROBLEMATIQUE

Interblocage / deadlock

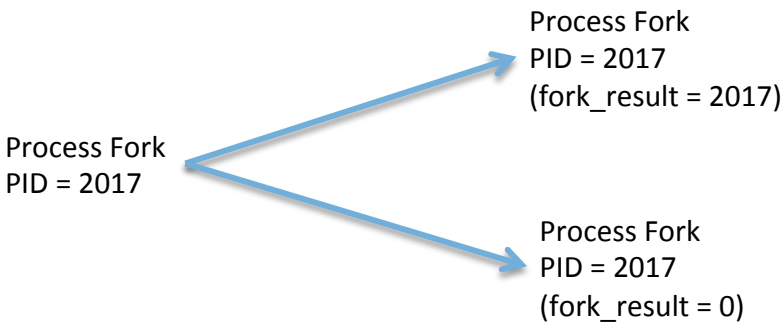
se produit, par exemple, lorsqu'un thread T1 ayant déjà acquis la ressource R1 demande l'accès à une ressource R2, pendant que le thread T2, ayant déjà acquis la ressource R2, demande l'accès à la ressource R1. Chacun des deux threads attend alors la libération de la ressource possédée par l'autre. La situation est donc bloquée.

FORK - DEFINITION

La fonction fork fait partie des appels système standard d'UNIX (norme POSIX1).

Cette fonction permet à un processus (un programme en cours d'exécution)
de donner naissance à un nouveau processus qui est sa copie conforme,
par exemple en vue de réaliser un second traitement parallèlement au premier.

Un bon moyen de visualiser l'effet d'un fork sur un processus est
d'imaginer une bactérie qui se coupe en deux.

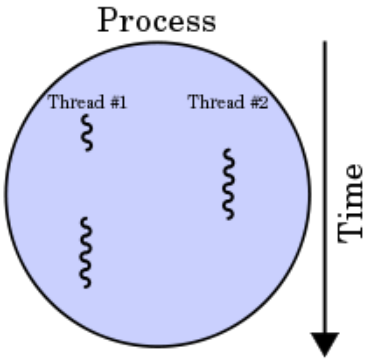


FORK - PROBLEMATIQUE

Size Mem(père) = Size Mem(fils)
performance ↘

THREAD

tâche, processus léger, fil
chaque thread a sa propre pile d'appel



Usages : Applications graphique, calcul intensif, … tout les communications asynchrones !

THREAD - OPTIMISATION

Avantages :
mémoire commune entre threads

accès mémoire en lecture seule : synchronisation inutile….
-> Algorithmes **Lockless**

Inconvénients :
complexité du développement
interblocages possibles
Introduction de variables globales

LA COMPILATION

LA COMPILATION

Soit le bout de code suivant « linktest.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int carre(int x);

static int var_globale = 32;

int main(int argc, char **argv)
{
   int i = 2;

   printf("Dodo\n");
   sleep(2);

   printf("Le carre de %d est %d\n", i,
         carre(i));
   printf("Le carre de %d est %d\n",
         var_globale, carre(i));

   return EXIT_SUCCESS;
}

int carre(int x)
{
   return x * x;
}
```

Après la compilation (**gcc -o linktest linktest.c**),
on utilise commande ***nm*** pour lister les symboles contenus par l'image binaire produite.

nm linktest

```
08049698 A __bss_start
08048344 t call_gmon_start
08048449 T carre
08049698 b completed.5621
0804958c d __CTOR_END__
[...]
08048564 R _IO_stdin_used
08049598 d __JCR_END__
08049598 d __JCR_LIST__
         w _Jv_RegisterClasses
08048460 T __libc_csu_fini
080484b0 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
080483c4 T main
08049690 d p.5619
         U printf@@GLIBC_2.0
         U puts@@GLIBC_2.0
         U sleep@@GLIBC_2.0
08048320 T _start
08049694 d var_globale
```

Analyse des librairies dynamiques

ldd linktest

```
libc.so.6 => /lib/tls/libc.so.6 (0xb7e66000)

/lib/ld-linux.so.2 (0xb7fa2000)

linux-gate.so.1 =>  (0xffffe000) [une fausse librairie : c'est en fait le kernel Linux]
```

LES THREADS

THREAD - Langages

SmallTalk
Java

} utilisent des extensions du langage ou des bibliothèques pour utiliser directement les services de multithreading du système d'exploitation, mais de façon portable

-> Espace Utilisateur

C++11, possède aussi une bibliothèque de gestion des threads : le modèle de classe est **std::thread**.

THREAD – Finalités & Design

Les threads : pour quelle finalité ?

- Maitre/esclaves
- Pool
- peer-to-peer

Pipeline

le thread « massif »:
utilisation de thread en « pool »
-> librairies C toutes faites (cthreadpool -- heber.tomer@gmail.com)
https://sourceforge.net/projects/cthreadpool/files/

THREAD POOL – usage:1 –

```
// création du pool
struct threadpool *pool;
int arr[ARR_SIZE], i, ret, failed_count = 0;

// On lancera 400 threads
#define ARR_SIZE 400

// Initialisation du « pool » de threads avec 10 coworkers.
if ((pool = threadpool_init(10)) == NULL)
{
  printf("Error! Failed to create a thread pool struct.\n");
  exit(EXIT_FAILURE);
}

for (i = 0; i < ARR_SIZE; i++) arr[i] = i;

// lancement de 400 Threads
for (i = 0; i < ARR_SIZE; i++)
{
  /* blocking (➔1). */
  ret = threadpool_add_task(pool, maFonction, arr + i,1);

  if (ret == -1) { printf("An error had occurred while adding a task."); exit(EXIT_FAILURE); }
  if (ret == -2) { failed_count++; }
}

// Vide le pool
threadpool_free(pool,1);
```

LE PARALLELISME

THREAD POOL – spécifications –

Réentrance (pure, synchro+mutex)

THREAD POOL – usage:2 –

```
// Variables Globales
static pthread_mutex_t count_mutex = PTHREAD_MUTEX_INITIALIZER;
static int count;

// Fonction ré-entrante avec mutex
void maFonction (void *ptr)
{
  int monArgument = *((int *) ptr);

  printf("maFonction(%d): count value is %d.\n", monArgument, count);

  pthread_mutex_lock(&count_mutex);
  count++;
  pthread_mutex_unlock(&count_mutex);
}
```