Module 2: Fils d'exécution

ÉCOLE POLYTECHNIQUE MONTRÉAL

INF8601: Systèmes Informatiques Parallèles

Michel Dagenais

Fils d'exécution

École Polytechnique Montréal

- Continuer le traitement avec un autre fil lorsqu'un fil est bloqué; utiliser plusieurs processeurs en parallèle;
- Processus conventionnel: mémoire séparée par défaut; mémoire partagée peut être ajoutée;
- Fils d'exécution en mode usager, coopératif, préemptif;
- Processus légers en mode usager;
- Processus légers gérés par le système d'exploitation;
- Solaris: mélange de processus légers en mode usager et en mode noyau;
- Linux: tout est un processus, différents attributs (mémoire, descripteurs...) peuvent ou non être partagés.

Le vrai coût en performance?

Processeur Intel i5, cycle de 0.4ns, LMBench

Opération	Temps ns		Opération	Temps ns
syscall trivial	65		fork+exit	263000
read	170		fork+execve	269400
write	138		pthread_create	~19000
stat	778		changer de processus	~1600
fstat	205		changer de fil	~1400
open/close	1492		iadd	.2
select 10 fd	515		imul	.18
select 500 fd	7022		idiv	9.52
signal	1348		dadd	1.32
segfault	158		ddiv	12.05
délai de tuyau	22716			

Le vrai coût en performance?

Processeur Intel i7, cycle de 0.47ns, LMBench

Opération	Temps ns		Opération	Temps ns
syscall trivial	43		fork+exit	116023
read	120		fork+execve	137390
write	74		pthread_create	
stat	437		changer de processus	
fstat	115		changer de fil	
open/close	895		iadd	.16
select 10 fd	275		imul	.02
select 500 fd	4227		idiv	7.6
signal	861		dadd	0.95
segfault	380		ddiv	4.48
délai de tuyau	4228			

Technologie Hyperthread

- Deux ensembles de registres pour un même processeur;
- Apparaît comme deux processeurs;
- Flots d'instructions entrelacés;
- Lorsqu'un processeur “virtuel” bloque sur une faute de cache, l'autre prend toute la place;
- Peut confondre le système d'exploitation ou les applications parallèles et nuire à leurs stratégies d'optimisation.

Fils d'exécution en mode usager

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Création: allouer pile, insérer entrée dans la table des fils d'exécution, créer un contexte initial qui pointe vers la pile;
- Fin: retirer l'entrée, désallouer la pile et le contexte, ou les conserver pour recyclage;
- “Yield”: sauver le contexte courant (setjmp), changer pour le contexte d'une autre tâche (longjmp);
- Préemption: demander une interruption régulière, sauver le contexte courant, changer pour le contexte d'une autre tâche.
- Appels systèmes bloquants? Multi-processeur?

Processus régulier

- Cloner le processus courant (fork);
- Le processus courant continue et éventuellement attend le résultat de l'enfant (waitpid);
- Le processus enfant ((pid = fork()) == 0) démarre souvent un autre exécutable (exec); il peut allouer des zones de mémoire partagée;
- L'enfant a le même contenu initial de mémoire virtuelle (COW), une copie des descripteurs de fichiers...
- L'enfant n'hérite pas des signaux, temporisateurs, verrous.

Fils d'exécution

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Création: pthread_create(&tid, …);
- Le parent attend souvent après l'enfant: pthread_join(tid, …);
- L'enfant partage la mémoire et toutes les ressources avec le parent mais utilise une partie différente de leur mémoire pour sa pile;
- Qui reçoit le signal envoyé au processus?

Avantages et inconvénients

- Même espace mémoire et table de page pour les fils d'exécution d'un même processus; plus compact, et création et changement de contextes plus rapides.
- Taille moins flexible pour les piles;
- Tout usage concurrent de données partagées doit être protégé par des verrous ou accédé avec des opérations atomiques.

Réentrance versus concurrence

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Réentrance: une interruption peut survenir n'importe quand. La routine d'interruption pourrait accéder des structures de données qui étaient en train d'être modifiées;
- Concurrence: deux fils d'exécution s'exécutent sur deux processeurs simultanément et pourraient accéder en même temps les mêmes structures de données.

Carte de la mémoire

ÉCOLE POLYTECHNIQUE MONTRÉAL

- cat /proc/'pid'/maps;
- Fichier exécutable (text, rodata, rwdata);
- Monceau + garde;
- libgcc;
- Piles des fils et gardes;
- Librairies partagées (text, rodata, rwdata);
- VDSO;
- ld.so;
- Pile principale extensible.

POSIX Threads

ÉCOLE POLYTECHNIQUE MONTRÉAL

- Diviser le travail pour utiliser plusieurs coeurs;
- Poursuivre l'utilisation du CPU lors de bloquages d'E/S;
- Simplifier le traitement asynchrone;
- Gestionnaire qui alimente un bassin de travailleurs (thread pool);
- Pipeline avec un fil d'exécution par station;
- Groupe de pairs, division hiérarchique du travail.

ANSI/IEEE POSIX 1003.1: PThreads

ÉCOLE POLYTECHNIQUE MONTRÉAL

- pthread_create (thread,attr,start_routine,arg);
- pthread_exit (status);
- pthread_cancel (thread);
- pthread_attr_init (attr);
- pthread_attr_destroy (attr);
- pthread_join (threadid,status);
- pthread_detach (threadid);
- pthread_attr_setstacksize (attr, stacksize);

PThreads Mutex

ÉCOLE POLYTECHNIQUE MONTRÉAL

- pthread_mutex_init (mutex,attr);
- pthread_mutex_destroy (mutex);
- pthread_mutexattr_init (attr);
- pthread_mutexattr_destroy (attr);
- pthread_mutex_lock (mutex);
- pthread_mutex_trylock (mutex);
- pthread_mutex_unlock (mutex) ;
- Attention à l'ordre de verrouillage!

PThreads Conditions

ÉCOLE POLYTECHNIQUE MONTRÉAL

- pthread_cond_init (condition,attr);
- pthread_cond_destroy (condition);
- pthread_condattr_init (attr);
- pthread_condattr_destroy (attr);
- pthread_cond_wait (condition,mutex);
- pthread_cond_signal (condition);
- pthread_cond_broadcast (condition);

Exemple pthread

ÉCOLE POLYTECHNIQUE MONTRÉAL

```
// exemple_pthreads.c
#define _REENTRANT
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
void afficher(int n, char lettre)
{       int i,j;
        for (j=1; j<n; j++)
        {       for (i=1; i < 10000000; i++);
                printf("%c",lettre);
                fflush(stdout);
        }
}

void *threadA(void *inutilise)
{       afficher(100,'A');
        printf("\n Fin du thread A\n");
        fflush(stdout);
        pthread_exit(NULL);
}
```

Exemple pthread (suite)

École Polytechnique Montréal

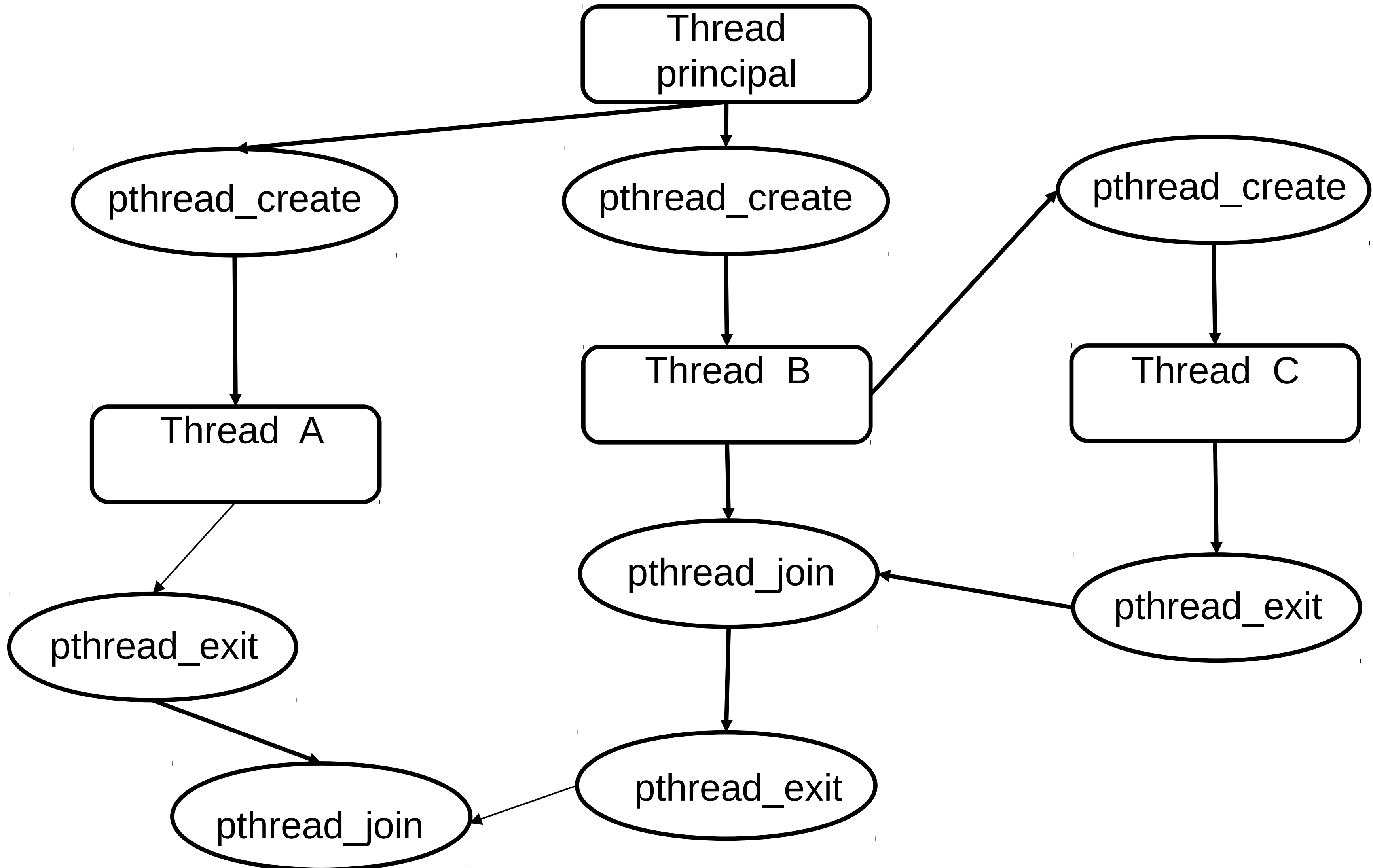
```
void *threadC(void *inutilise)
{
        afficher(150,'C');
        printf("\n Fin du thread C\n");
        fflush(stdout);
        pthread_exit(NULL);
}

void *threadB(void *inutilise)
{
        pthread_t thC;
        pthread_create(&thC, NULL, threadC, NULL);
        afficher(100,'B');
        printf("\n Le thread B attend la fin du thread C\n");
        pthread_join(thC,NULL);
        printf("\n Fin du thread B\n");
        fflush(stdout);
        pthread_exit(NULL);
}
```

Exemple pthread (suite)

ÉCOLE POLYTECHNIQUE MONTRÉAL

```
int main()
{
        int i;

        pthread_t thA, thB;

        printf("Creation du thread A");

        pthread_create(&thA, NULL, threadA, NULL);
        pthread_create(&thB, NULL, threadB, NULL);

        sleep(1);
        //attendre la fin des threads
        printf("Le thread principal attend que les autres se terminent\n");

        pthread_join(thA,NULL);
        pthread_join(thB,NULL);

        exit(0);
}
```

Exemple pthread (suite)



Exemple pthread (suite)

ÉCOLE POLYTECHNIQUE MONTRÉAL

```
-bash-3.2$ ./exemple_threads
Creation du thread
AACBACBACBACBACBACBACBACBACBACBABCABCABCABCABCABCA
BCABCABCABCABCABCABCABCACBACBACBACBLe thread principal
attend que les autres se terminent
ACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBAC
BACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBA
CBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACB
ACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBAC
BACBACBACBA
 Fin du thread A
CB
 Le thread B attend la fin du thread C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
 Fin du thread C

 Fin du thread B
-bash-3.2$
```

Exemple pthread (suite)

-bash-3.2$ ./exemple_threads
Creation du thread
AACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBA
CBACBACBACBACBACBACBACBACBACBACBACBLe thread principal attend que les autres se terminent
ACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBAC
BACBACBACBACBACBACBACBACBCABCABCABCABCABCABCABCABC
ABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCAB
CABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCA
BCABCABCABCA
 Fin du thread A
B
 Le thread B attend la fin du thread C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
 Fin du thread C

 Fin du thread B
-bash-3.2$