RAPPORT – Travail Pratique No 2

Travaux avancés avec les threads Linux.

*Résultat:* \_\_\_\_\_ / 100

(Note : ce rapport est écrit de façon à vous faciliter la vie. En cas d’omission ou de différence entre ce rapport vierge et l’énoncé du TP, l’énoncé a priorité).

# 1. Niveaux de priorités des threads dans Linux ( /20 pts)

## 1.1 Programmation de threads avec niveaux de priorités

**Code Source : nosprioprites.c**

﻿#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <errno.h>

#include <string.h>

#include <sys/syscall.h>

#include <sys/resource.h>

#define N\_THREADS 5

typedef struct

{

int ThreadNum;

}Parametres;

void \*FonctionThread(void \*data)

{

Parametres \*pParam = (Parametres \*)data;

printf("Je suis le thread %d et je demarre !!! \n", pParam->ThreadNum);

int ThreadID = syscall(SYS\_gettid);

int ret = setpriority(PRIO\_PROCESS,ThreadID,pParam->ThreadNum);

printf("Code retour de setpriority() pour processus %d : %d. \n", pParam->ThreadNum, ret);

printf("Code retour de errno pour processus %d : %s. \n", pParam->ThreadNum,strerror(errno));

while(1);

pthread\_exit(NULL);

}

int main()

{

pthread\_t threads[N\_THREADS];

Parametres myParam[N\_THREADS];

int i;

for(i=0; i<N\_THREADS;i++)

{

printf("creation thread %d! \n", i);

myParam[i].ThreadNum = i;

pthread\_create(&threads[i],NULL,FonctionThread,(void \*)&myParam[i]);

}

for (i=0;i<N\_THREADS;i++)

{

pthread\_join(threads[i],NULL);

}

exit(0);

}

a) **Aucun changement de priorité**:

﻿ ﻿creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Je suis le thread 4 et je demarre !!!

Code retour de setpriority() pour processus 4 : 0.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Code retour de setpriority() pour processus 3 : 0.

Code retour de errno pour processus 3 : Success.

Je suis le thread 2 et je demarre !!!

Code retour de setpriority() pour processus 2 : 0.

Code retour de errno pour processus 2 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de setpriority() pour processus 1 : 0.

Code retour de errno pour processus 1 : Success.

Je suis le thread 0 et je demarre !!!

Code retour de setpriority() pour processus 0 : 0.

Code retour de errno pour processus 0 : Success.

b) **Test 2**:

﻿creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Je suis le thread 4 et je demarre !!!

Code retour de setpriority() pour processus 4 : 0.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Code retour de setpriority() pour processus 3 : 0.

Code retour de errno pour processus 3 : Success.

Je suis le thread 2 et je demarre !!!

Code retour de setpriority() pour processus 2 : 0.

Code retour de errno pour processus 2 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de setpriority() pour processus 1 : 0.

Code retour de errno pour processus 1 : Success.

Je suis le thread 0 et je demarre !!!

Code retour de setpriority() pour processus 0 : 0.

Code retour de errno pour processus 0 : Success.

c) **Test 3**:

creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Je suis le thread 4 et je demarre !!!

Code retour de setpriority() pour processus 4 : 0.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Code retour de setpriority() pour processus 3 : 0.

Code retour de errno pour processus 3 : Success.

Je suis le thread 2 et je demarre !!!

Code retour de setpriority() pour processus 2 : 0.

Code retour de errno pour processus 2 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de setpriority() pour processus 1 : 0.

Code retour de errno pour processus 1 : Success.

Je suis le thread 0 et je demarre !!!

Code retour de setpriority() pour processus 0 : 0.

Code retour de errno pour processus 0 : Success.

d) **Test 4**:

creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Je suis le thread 4 et je demarre !!!

Code retour de setpriority() pour processus 4 : 0.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Je suis le thread 2 et je demarre !!!

Code retour de setpriority() pour processus 2 : -1.

Code retour de errno pour processus 2 : Permission denied.

Code retour de setpriority() pour processus 3 : 0.

Code retour de errno pour processus 3 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de setpriority() pour processus 1 : -1.

Code retour de errno pour processus 1 : Permission denied.

Je suis le thread 0 et je demarre !!!

Code retour de setpriority() pour processus 0 : 0.

Code retour de errno pour processus 0 : Success.

On remarque les permission denied puisque le programme n’a pas été lancé en mode super utilisateur.

e) **Test 5** (mode super user)

creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Je suis le thread 4 et je demarre !!!

Code retour de setpriority() pour processus 4 : 0.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Code retour de setpriority() pour processus 3 : 0.

Code retour de errno pour processus 3 : Success.

Je suis le thread 2 et je demarre !!!

Code retour de setpriority() pour processus 2 : 0.

Code retour de errno pour processus 2 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de setpriority() pour processus 1 : 0.

Code retour de errno pour processus 1 : Success.

Je suis le thread 0 et je demarre !!!

Code retour de setpriority() pour processus 0 : 0.

Code retour de errno pour processus 0 : Success.

## 1.2 Observation du temps d’exécution des threads avec différents niveaux de priorités

a)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** | **NI** |
| **0** | 20 | 21359 | 19.1 | 0 |
| **1** | 20 | 21360 | 19.5 | 0 |
| **2** | 20 | 21361 | 19.1 | 0 |
| **3** | 20 | 21362 | 19.5 | 0 |
| **4** | 20 | 21363 | 19.5 | 0 |

b)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** | **Pourcentage théorique CFS** | **NI** |
| **0** | 21 | 21442 | 29.1 | 29.7 | 1 |
| **1** | 22 | 21443 | 22.8 | 23.7 | 2 |
| **2** | 23 | 21444 | 18.5 | 19.1 | 3 |
| **3** | 24 | 21445 | 15.2 | 15.3 | 4 |
| **4** | 25 | 21446 | 11.9 | 12.1 | 5 |

Détail des calculs pour CFS :

A l’image du laboratoire b de la semaine 5 on prend les poids : 820, 655, 526, 423 et 335 pour les niveaux de priorité respectifs : 1,2,3,4 et 5.

L’équation du CFS est donné par Avec  on trouve : , , , , . Bien sur les pourcentages théoriques sont différents du % CPU utilisé, une des raisons est que nos threads ne sont pas les seules entités utilisant le CPU.

c)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** | **NI** |
| **0** | 29 | 21527 | 6.6 | 9 |
| **1** | 27 | 21528 | 10.2 | 7 |
| **2** | 25 | 21529 | 15.8 | 5 |
| **3** | 23 | 21530 | 24.8 | 3 |
| **4** | 21 | 21531 | 38.9 | 1 |

d)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** | **NI** |
| **0** | 20 | 21593 | 24.5 | 0 |
| **1** | 20 | 21594 | 24.5 | 0 |
| **2** | 20 | 21595 | 24.5 | 0 |
| **3** | 23 | 21596 | 12.6 | 3 |
| **4** | 24 | 21597 | 10.3 | 4 |

e)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** | **NI** |
| **0** | 20 | 21615 | 17.9 | 0 |
| **1** | 17 | 21616 | 34.9 | -3 |
| **2** | 18 | 21617 | 27.9 | -2 |
| **3** | 23 | 21618 | 9.3 | 3 |
| **4** | 24 | 21619 | 7.3 | 4 |

## 1.3 Expérimentations avec différents algorithmes d’ordonnancement

### First In First Out (FIFO)

|  |  |  |  |
| --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** |
| **0** | ? | ? | ? |
| **1** | ? | ? | ? |
| **2** | ? | ? | ? |
| **3** | ? | ? | ? |
| **4** | rt | 21889 | 96% |

Toutes les sorties d’écran

Terminal ou le processus est lancé :

creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 4 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 3 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 2 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 1 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 0 et je demarre !!!

Terminal avec la commande Top -H

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

21889 root rt 0 43296 632 580 R 94.4 0.0 1:08.16 nosprioritesfifo

927 root 20 0 207724 66044 30580 S 0.3 0.8 0:37.91 Xorg

1701 etu1 20 0 58440 22496 18032 S 0.3 0.3 0:06.32 xfce4-terminal

Incluez le code source ici (en plus de le mettre dans le zip).

#define N\_THREADS 5

typedef struct

{

int ThreadNum;

}Parametres;

void scheduler\_to\_fifo()

{

struct sched\_param Param;

Param.sched\_priority=sched\_get\_priority\_max(SCHED\_FIFO);

sched\_setscheduler(0,SCHED\_FIFO,&Param);

printf("Code retour de errno pour changement d'ordonnanceur : %s. \n", strerror(errno));

}

void \*FonctionThread(void \*data)

{

Parametres \*pParam = (Parametres \*)data;

scheduler\_to\_fifo();

printf("Je suis le thread %d et je demarre !!! \n", pParam->ThreadNum);

/\* int ThreadID = syscall(SYS\_gettid);

int ret;

switch(pParam->ThreadNum)

{

case 0:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 1:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 2:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 3:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 4:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

}

printf("Code retour de setpriority() pour processus %d : %d. \n", pParam->ThreadNum, ret);

printf("Code retour de errno pour processus %d : %s. \n", pParam->ThreadNum,strerror(errno));

\*/

while(1);

pthread\_exit(NULL);

}

int main()

{

pthread\_t threads[N\_THREADS];

Parametres myParam[N\_THREADS];

int i;

for(i=0; i<N\_THREADS;i++)

{

printf("creation thread %d! \n", i);

myParam[i].ThreadNum = i;

pthread\_create(&threads[i],NULL,FonctionThread,(void \*)&myParam[i]);

}

for (i=0;i<N\_THREADS;i++)

{

pthread\_join(threads[i],NULL);

}

exit(0);

}

### Round Robin

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thread** | **Niveau de priorité PR** | **PID** | **% CPU utilisé** | **NI** |
| **0** | rt | 22171 | 19.9 | 0 |
| **1** | rt | 22172 | 19.9 | 0 |
| **2** | rt | 22173 | 18.9 | 0 |
| **3** | rt | 22174 | 16.9 | 0 |
| **4** | rt | 22175 | 19.3 | 0 |

Terminal ou le processus est lancé :

creation thread 0!

creation thread 1!

creation thread 2!

creation thread 3!

creation thread 4!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 4 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 3 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 2 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 1 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur : Success.

Je suis le thread 0 et je demarre !!!

Terminal avec la commande Top -H

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

22171 root rt 0 43296 632 580 R 19.9 0.0 0:09.01 nosprioritesrr

22172 root rt 0 43296 632 580 R 19.9 0.0 0:09.12 nosprioritesrr

22175 root rt 0 43296 632 580 R 19.3 0.0 0:10.84 nosprioritesrr

22173 root rt 0 43296 632 580 R 18.9 0.0 0:09.56 nosprioritesrr

22174 root rt 0 43296 632 580 R 16.9 0.0 0:09.45 nosprioritesrr

**Code source nosprioritesrr.c**

#define N\_THREADS 5

typedef struct

{

int ThreadNum;

}Parametres;

void change\_scheduler(int scheduler)

{

struct sched\_param Param;

Param.sched\_priority=sched\_get\_priority\_max(SCHED\_FIFO);

sched\_setscheduler(0,SCHED\_RR,&Param);

printf("Code retour de errno pour changement d'ordonnanceur : %s. \n", strerror(errno));

}

void \*FonctionThread(void \*data)

{

int scheduler=SCHED\_RR;

Parametres \*pParam = (Parametres \*)data;

change\_scheduler(scheduler);

printf("Je suis le thread %d et je demarre !!! \n", pParam->ThreadNum);

/\* int ThreadID = syscall(SYS\_gettid);

int ret;

switch(pParam->ThreadNum)

{

case 0:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 1:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 2:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 3:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

case 4:

ret = setpriority(PRIO\_PROCESS,ThreadID,0);

break;

}

printf("Code retour de setpriority() pour processus %d : %d. \n", pParam->ThreadNum, ret);

printf("Code retour de errno pour processus %d : %s. \n", pParam->ThreadNum,strerror(errno));

\*/

while(1);

pthread\_exit(NULL);

}

int main()

{

pthread\_t threads[N\_THREADS];

Parametres myParam[N\_THREADS];

int i;

for(i=0; i<N\_THREADS;i++)

{

printf("creation thread %d! \n", i);

myParam[i].ThreadNum = i;

pthread\_create(&threads[i],NULL,FonctionThread,(void \*)&myParam[i]);

}

for (i=0;i<N\_THREADS;i++)

{

pthread\_join(threads[i],NULL);

}

exit(0);

}

# 2. Implémentations de différents verrous (20 pts)

Code source qlock1 : (seulement la partie implémentée)

**void** BadLock()

{

**while** (lock==1);

lock=1;

}

**void** BadUnlock()

{

lock=0;

}

*//2.2*

**void** AtomicLock() {

}

**void** AtomicUnlock() {

}

**void** \*CodeThread(**void** \* a)

{

**long** i;

**for**(i = 0; i < N\_ITER; i++)

{

BadLock();

count++;

BadUnlock();

}

}

Code source qlock2 : (Seulement la partie implémentée)

**void** AtomicLock() {

**while** (xchg(&atomicLock,1));

}

**void** AtomicUnlock() {

xchg(&atomicLock,0);

}

**void** \*CodeThread(**void** \* a)

{

**long** i;

**for**(i = 0; i < N\_ITER; i++)

{

AtomicLock();

count++;

AtomicUnlock();

*// La fonction de unlock à ajouter ici*

}

}

Code source qlock3 :

**void** BadLock() {

}

**void** BadUnlock() {

}

*//2.2*

**void** AtomicLock() {

}

**void** AtomicUnlock() {

}

**void** \*CodeThread(**void** \* a)

{

**long** i;

**for**(i = 0; i < N\_ITER; i++)

{

pthread\_mutex\_lock(&monMutex);

count++;

pthread\_mutex\_unlock(&monMutex);

}

}

*// -------- la fonction main ----------*

**int** main(**int** argc, **char** \*\*argv)

{

pthread\_t T1, T2;

pthread\_mutex\_init(&monMutex,0);

*//pthread\_mutex\_unlock(&monMutex);*

**int** iTrial;

**int** correct = 0;

**for** (iTrial = 0; iTrial < 10000; iTrial++) {

count = 0;

pthread\_create(&T1, 0, CodeThread, 0);

pthread\_create(&T2, 0, CodeThread, 0);

pthread\_join(T1, **NULL**);

pthread\_join(T2, **NULL**);

printf("La valeur finale est %ld\n",count);

**if** (count == 2\*N\_ITER) correct++;

printf("%d/%d de correct!\n",correct,iTrial+1);

}

}

### 2.4 Temps d’exécution

On regarde le temps pour 100 boucles du main puis on divise le temps d’exécution du programme par 100.

|  |  |  |
| --- | --- | --- |
| **Méthode** | **Temps d’exécution moyen par itération (1 CPU)** | **Temps d’exécution moyen par itérations (2 CPU)** |
| Rien | Real : 0.05911 s | Real : 0.07267 s |
| BadLock | Real : 0.09221 s | Real : 0.21023 s |
| AtomicLock | Real : 0.57284 s | Real : 2,14662 s |
| pthread\_mutex | Real : 0.44224 s | Real : 1,6733 s |

2.5 Taux d’erreur **(    /4 pts)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Méthode** | **Taux avec 1 CPU** | **Taux avec 2 CPU** | **Etat Algorithme 1 CPU** |
| Rien | 5,1% | 100% | Non correct |
| BadLock | 0,0797 % | 100% | Non correct |
| AtomicLock | 0.000% | 0% | Correct |
| pthread\_mutex | 0.000% | 0% | Correct |

La méthode sans lock est exécutée pendant 1000 boucles. (Les conflits sont nombreux)

La méthode BadLock est exécutée pendant 5 000 boucles.

La méthode AtomicLock est exécutée pendant 5 500 boucles.

La méthode pthread\_mutex est exécutée pendant 5 000 boucles.

# 3. Producteurs-Consommateurs en utilisant une file FIFO (60 pts)

Incluez le code source ici (en plus de le mettre dans le zip).

Incluez la sortie d’écran du programme.