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>

#include <sched.h>

#define N\_THREADS 5

#define CHOIX\_PRIORITE 2 //Changer ce paramètre pour modifier la priorité des threads.

int arr\_thread\_priorite[5][5] = {

        {0,0,0,0,0},

        {1,2,3,4,5},

        {9,7,5,3,1},

        {0,-4,-2,3,4},

        {0,-3,-2,3,4}};

typedef struct

{

    int ThreadNum;

} Parametres;

void \*work(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,

                            arr\_thread\_priorite[CHOIX\_PRIORITE][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, work, (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 :

À 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** | -51 | 5427 | 95% |

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!

Je suis le thread 4 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur pour processus 5427 : Success.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur pour processus 5426 : Success.

Code retour de errno pour processus 3 : Success.

Je suis le thread 2 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur pour processus 5425 : Success.

Code retour de errno pour processus 2 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur pour processus 5424 : Success.

Code retour de errno pour processus 1 : Success.

Je suis le thread 0 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur pour processus 5423 : Success.

Code retour de errno pour processus 0 : Success.

Terminal avec la commande Top -H

top - 14:05:22 up 1:45, 4 users, load average: 5.12, 5.04, 5.39

Threads: 347 total, 6 running, 341 sleeping, 0 stopped, 0 zombie

%Cpu(s): 93.4 us, 0.0 sy, 0.0 ni, 6.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 1023812 total, 324116 free, 205080 used, 494616 buff/cache

KiB Swap: 1046524 total, 1046524 free, 0 used. 648928 avail Mem

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

5427 root -51 0 43296 592 536 R 95.0 0.1 1:02.34 nospriorite+

1183 root 20 0 158520 81988 32136 S 0.3 8.0 0:21.14 Xorg

1608 etu1 20 0 37460 17412 15412 S 0.3 1.7 0:00.57 xfce4-panel

1613 etu1 20 0 68192 29840 20216 S 0.3 2.9 0:04.11 xfdesktop

5311 etu1 20 0 8216 3600 2956 R 0.3 0.4 0:01.42 top

1 root 20 0 24096 5244 3896 S 0.0 0.5 0:01.26 systemd

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

#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>

#include <sched.h>

#define N\_THREADS 5

typedef struct

{

    int ThreadNum;

} Parametres;

void scheduler\_to\_fifo()

{

    struct sched\_param Param;

    int maxPriority = sched\_get\_priority\_max(SCHED\_FIFO);

    int minPriority = sched\_get\_priority\_min(SCHED\_FIFO);

    int meanPriority = (maxPriority + minPriority) / 2;

    Param.sched\_priority = meanPriority;

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

    int ThreadID = syscall(SYS\_gettid);

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

}

void \*work(void \*data)

{

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

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

    scheduler\_to\_fifo();

    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, work, (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** | -51 | 5751 | 18.9 | 0 |
| **1** | -51 | 5752 | 18.4 | 0 |
| **2** | -51 | 5753 | 18.9 | 0 |
| **3** | -51 | 5754 | 18.9 | 0 |
| **4** | -51 | 5755 | 21.2 | 0 |

Terminal ou le processus est lancé :

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 errno pour changement d'ordonnanceur(2) pour processus 5755 : Success.

Code retour de errno pour processus 4 : Success.

Je suis le thread 3 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur(2) pour processus 5754 : Success.

Code retour de errno pour processus 3 : Success.

Je suis le thread 0 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur(2) pour processus 5751 : Success.

Code retour de errno pour processus 0 : Success.

Je suis le thread 2 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur(2) pour processus 5753 : Success.

Code retour de errno pour processus 2 : Success.

Je suis le thread 1 et je demarre !!!

Code retour de errno pour changement d'ordonnanceur(2) pour processus 5752 : Success.

Code retour de errno pour processus 1 : Success.

Terminal avec la commande Top -H

top - 14:52:50 up 2:32, 4 users, load average: 4.67, 2.81, 1.64

Threads: 347 total, 8 running, 339 sleeping, 0 stopped, 0 zombie

%Cpu(s): 93.4 us, 0.5 sy, 0.0 ni, 6.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 1023812 total, 205876 free, 209544 used, 608392 buff/cache

KiB Swap: 1046524 total, 1046524 free, 0 used. 643820 avail Mem

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

5755 root -51 0 43296 596 544 R 21.2 0.1 0:05.08 nospriorite+

5751 root -51 0 43296 596 544 R 18.9 0.1 0:04.14 nospriorite+

5753 root -51 0 43296 596 544 R 18.9 0.1 0:03.90 nospriorite+

5754 root -51 0 43296 596 544 R 18.9 0.1 0:03.80 nospriorite+

5752 root -51 0 43296 596 544 R 18.4 0.1 0:04.04 nospriorite+

1183 root 20 0 161052 84352 32344 S 0.3 8.2 0:35.37 Xorg

**Code source nosprioritesrr.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>

#include <sched.h>

#define N\_THREADS 5

typedef struct

{

    int ThreadNum;

} Parametres;

void change\_scheduler(int scheduler)

{

    struct sched\_param Param;

    int maxPriority = sched\_get\_priority\_max(scheduler);

    int minPriority = sched\_get\_priority\_min(scheduler);

    int meanPriority = (maxPriority + minPriority) / 2;

    Param.sched\_priority = meanPriority;

    sched\_setscheduler(0, scheduler, &Param);

    int ThreadID = syscall(SYS\_gettid);

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

}

void \*work(void \*data)

{

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

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

    change\_scheduler(SCHED\_RR);

    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, work, (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 |  |
| BadLock | Real : 0.09221 s |  |
| AtomicLock | Real : 0.57284 s |  |
| pthread\_mutex | Real : 0.44224 s |  |

2.5 Taux d’erreur **(    /4 pts)**

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