Théorie et pratique de la concurrence – Master 1 Informatique
TP 4 : Programmation concurrente en C (suite)

Les mutexs

L s *mutexs* sont s s map or s b na r s (n pouvant pr n r qu a va ur ou) a b b ot qu `Pthreads` On s app ga m nt v rrous our s ut s r, aut nc ur a b b ot qu

```
#include <pthread.h>
```

La c arat on 'un *mutex* s a t a açon su vant

```
pthread_mutex_t verrou;
```

our n t a s r un *mutex*, on proc comm su t

```
pthread_mutex_init (&verrou , NULL);
```

L pr m r argum nt corr spon au v rrou qu 'on sou a t n t a s r, quan au s con , qu st m s a `NULL`, pr c s s attr buts u v rrou
our "pr n r un v rrou" (a r sur v rrou on a t

```
pthread_mutex_lock (&verrou);
```

Et pour b r r

```
pthread_mutex_unlock (&verrou);
```

ar cons qu nt, un s ct on cr t qu pourra tr a t a açon su vant

```
pthread_mutex_lock (&verrou);
/*section critique*/
pthread_mutex_unlock(&verrou);
```

Les variables de condition

La b b ot qu `Pthreads` propos (((3 (v ene10-622(a)-(r-5.8013(r)-1.66(e)-13"pre 1.4.50084(>)-4.980
putd_w_ 47077(e)-4.97954(x)-4.98077(_)-4

```
pthread_mutex_lock(&verrou);
pthread_cond_wait (&varcond,&verrou);
pthread_mutex_unlock(&verrou);
```

Pour r v r un *thread* b oqu sur un var ab con t on, un autr t r a p ut a r

```
pthread_cond_signal(&varcond);
```

La auss st **recommandé** 'app r c tt onct on au s n 'un s ct on cr t qu prot g par *mutex* ut s par t r a n att nt (ans ' x mp pr c nt s'ag t u *mutex* `verrou`) On p ut auss r v r tous s *threads* n att nt grac a a onct on

```
pthread_cond_broadcast(&verrou);
```

IMPORTANT : Pour compiler vos programmes, n'oubliez pas de mettre l'option `-lpthread`

ous trouv r un x mp s progr amm s ut sant s onct onna t s pr s nt s sur a pag w b s D
`http://www.liafa.jussieu.fr/~sangnier/enseignement/concurrence.html`

Exercices

Exercice *Salle de bain unisexe*

uppos qu'on spos 'un s u sa ba n qu p ut tr ut s par s omm s ou s mm s ma s pas au m m mom nt

Progr amm n ut sant s s map or s un so ut on a c prob m I aut p rm ttr un nombr qu conqu ' omm s O mm s tr ans a sa ba n au m m mom nt La so ut on o t assur r ' xc us on mutu man t 'abs nc ' nt r b ocag

o votr so ut on pour p rm ttr au p us p rsonn s (u m m s x) n m m t mps

3 On sou a t ma nt nant garant r qu' y aura un a t rnanc ntr omm s t mm s (s y a r gu r m nt s omm s t s mm s qu ont a qu u pour c a, v opp un so ut on qu a ss au p us ntr r mm s [r sp ct v m nt omm s] ans a sa ba ns, s un omm [r sp un mm st n tra n 'att n r]

Exercice *Programmation des sémaphores avec les variables de condition*

Dans c t x rc c vous st man progr amm r vos propr s s map or s pour c a r , vous pouv cr r un typ s map or a orm su vant

```
typedef struct sema
pthread_mutex_t verrou;
pthread_cond_t varcond;
int compteur;
 semaphore;
```

Intu t v m nt v rrou s rt a prot g r a va ur u compt ur assoc au s map or t a var ab con t on st ut s pour b oqu r s *threads* qu t nt nt pr n r s map or a ors qu a va ur u compt ur st a

Progr amm s onct ons `int semaphore_init(semaphore *sem,int val)`, `int semaphore_post(seamphore *sem)` t `int semaphore_wait(semaphore *sem)` c s map or (qu corr spon nt aux onct on qu va nt s pour s s map or a bra r `Pthreads`)

st votr v rs on s s map or s n s ut sant pour pr m r x rc c u pr c nt

Exercice 3 *Problème de la liste partagée*

D s vo tur s arr vant u nor ou u su o v nt pass r un pont a vo un qu L s vo tur s ayant a m m r ct on p uv nt pass r pont au m m mom nt, ma s s vo tur s ayant s r ct ons oppos s n p uv nt pas

rogramm co s proc ssus vo tur , a ns qu co n c ssa r a ur sync ron sat on