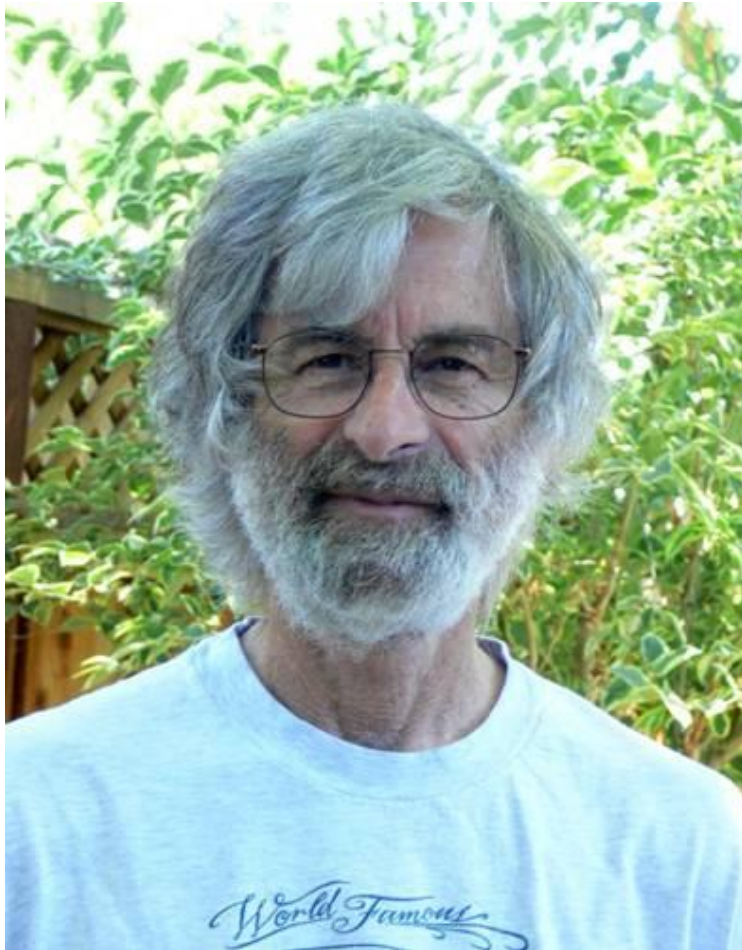




Systemes d'exploitation

Synchronisation de
exécution

Lesslie Lamport



- Américain
- MIT
- LaTeX
- L'algorithme Lamport bakery
- Tolérance aux fautes Byzantines
 - Utilise sur blockchain
- L'algorithme Paxos
- Signature de Lamport

- course
- espace critique
- opérations atomique
- busy waiting
- semaphore
- mutex



Bibliographie pour aujourd'hui

- Modern Operating Systems
 - Chapitre 4
 - 4.4
 - 4.5
 - 4.6
- Operating Systems Concepts
 - Chapitre 9
 - 9.1 – 9.4
 - 9.6
 - 9.7

COURSE

Processus 1

// shared variable a

```
int a = 0;
if (a == 0)
{
    a++;
}
printf ("%d\n", a);
```

Processus 2

// shared variable a

```
int a = 0;
if (a == 0)
{
    a++;
}
printf ("%d\n", a);
```

Course – variante séquentiel

Processus 1

// shared variable a

```
int a = 0;
if (a == 0) {
    a++;
}
printf ("%d\n", a);
```

1

Processus 2

// shared variable a

```
int a = 0;
if (a == 0) {
    a++;
}
printf ("%d\n", a);
```

1

Course – variante intercalé

Processus 1

// shared variable a

```
int a = 0;  
if (a == 0) {
```

```
    a++;
```

```
}
```

```
printf ("%d\n", a);
```

2

Processus 2

// shared variable a

```
int a = 0;  
if (a == 0) {
```

```
    a++;
```

```
}
```

```
printf ("%d\n", a);
```

2

Course – désiré

Processus 1

// shared variable a

```
int a = 0;
```

```
if (a == 0) {  
    a++;  
}
```

```
printf ("%d\n", a);
```

Processus 2

// shared variable a

```
int a = 0;
```

```
if (a == 0) {  
    a++;  
}
```

```
printf ("%d\n", a);
```

1

1

espace critique

Espace critique

- une partie du code qui doit être exécuté seul
 - l'accès aux variables (ressources) qui doit être exécuté seul



```
if (a == 0) {  
    a++;  
}
```



```
if (a == 1) {  
    a--;  
}
```

Le raison est l'accès de variable a

Operation atomique

- Un opération qui est exécute sans interruption
- Interruptions possible
 - interruption matériel
 - signaux
 - changement de contexte
- Les instructions de CPU sont atomique

Exemple

```
int main ()  
{  
    int a = 0;  
    a++;  
}
```

Plus des exemples sur <https://godbolt.org/>

Exemple – x86-64

```
int main ()  
{  
    int a = 0;  
  
    a++;  
}
```

```
; rbp-4 address of variable a  
mov DWORD PTR [rbp-4], 0  
  
add DWORD PTR [rbp-4], 1
```

a++ est atomique

Exemple – ARM

```
int main ()  
{  
    in a = 0;  
  
    a++;  
}
```

; fp-8 address of variable a

```
mov r3, #0  
str r3, [fp, #-8]  
  
ldr r3, [fp, #-8]  
add r3, r3, #1  
str r3, [fp, #-8]
```

a++ n'est pas atomique

Instruction pour synchronisation

; rbp-4 address of variable test

```
if (test == 0)
{
    test = 1;

    // do some work
```

```
test = 0;
}
```

```
cmp DWORD PTR [rbp-4], 0
jne .L2
```

```
mov DWORD PTR [rbp-4], 1
```

; do some work

```
mov DWORD PTR [rbp-4], 0
.L2:
```

N'est pas atomique

Test and set

- Une instruction qui
 - Fait un comparaison
 - Fait un attribution en dépendent de la comparaison
- TSL

```
if (r == 0)  
    r = value;
```

```
tsl (r, value)
```

```
tsl r, value
```

Instruction pour synchronisation

; rbp-4 address of variable test

```
if (tsl(test, 1))  
{  
  
    // do some work
```

```
    test = 0;  
}
```

```
mov eax, 0  
lock cmpxchg DWORD PTR  
            [rbp-4], 1  
jne .L2
```

; do some work

```
mov DWORD PTR [rbp-4], 0  
.L2:
```

est atomique

Fair le code executer atomique

- Spinlock
- Semaphore
- Mutex

SPINLOCK

Spinlock

```
while (test != 0)  
    continue;
```



Spinlock

```
test = 1;
```

Atomique?

```
// do some work with
```

```
test = 0;
```

Spinlock x86

; rbp-4 address of variable test

```
while (test != 0)
```

```
    continue;
```

```
test = 1;
```

```
// do some work with
```

```
test = 0;
```

```
.L2:
```

```
cmp DWORD PTR [rbp-4], 0
```

```
jne .L2
```

```
mov DWORD PTR [rbp-4], 1
```

```
; do some work
```

```
mov DWORD PTR [rbp-4], 0
```

N'est pas atomique

Spinlock

```
while (tsl(test, 1))  
    continue;
```



Spinlock

```
// do some work with
```

```
test = 0;
```

Spinlock x86

; rbp-4 address of variable test

```
while (tsl(test, 1))  
    continue;
```

```
// do some work with
```

```
test = 0;
```

```
mov eax, 0
```

```
.L2:
```

```
lock cmpxchg DWORD PTR  
                [rbp-4], 1
```

```
jne .L2
```

```
; do some work
```

```
mov DWORD PTR [rbp-4], 0
```

Est atomique

lock et unlock

- lock
 - prend le spinlock
- unlock
 - libère le spinlock

lock et unlock

```
void lock (s)
{
    while (tsl(s, 1));
}
```

```
void unlock (s)
{
    s = 0;
}
```

// s - spinlock

Exemple

Processus 1

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
lock (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
unlock (s);
```

```
printf ("%d\n", a);
```

Processus 2

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
lock (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
unlock (s);
```

```
printf ("%d\n", a);
```

Exemple

Processus 1

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
lock (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
unlock (s);
```

```
printf ("%d\n", a);
```

Processus 2

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
lock (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
unlock (s);
```

```
printf ("%d\n", a);
```

Deadlock

Que se passe-t-il si nous oublions d'utiliser le unlock?

Que se passe-t-il si nous utilisons plusieurs de fois lock sans unlock?

Que se passe-t-il si le program s'arrête avant unlock?

Oublier unlock

Processus 1

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
lock (s);
```

Processus 2

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
lock (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
printf ("%d\n", a);
```

... waits a lot

...

Plusieurs de lock

Processus 1

// shared variables a and s

int a = 0;

int s;

lock (s);

... waits a lot

...

Processus 2

// shared variables a and s

int a = 0;

int s;

lock (s);

lock (s);

...waits a lot

...

Arrête avent unlock

Processus 1

// shared variables a and s

int a = 0;

int s;

lock (s);

Processus 2

// shared variables a and s

int a = 0;

int s;

lock (s);

if (a == 0)

{

 a++;

}

... waits a lot

...

erreur

SÉMAPHORE

Spinlock

```
while (tsl(test, 1))  
    continue;
```

```
// do some work with
```

```
test = 0;
```

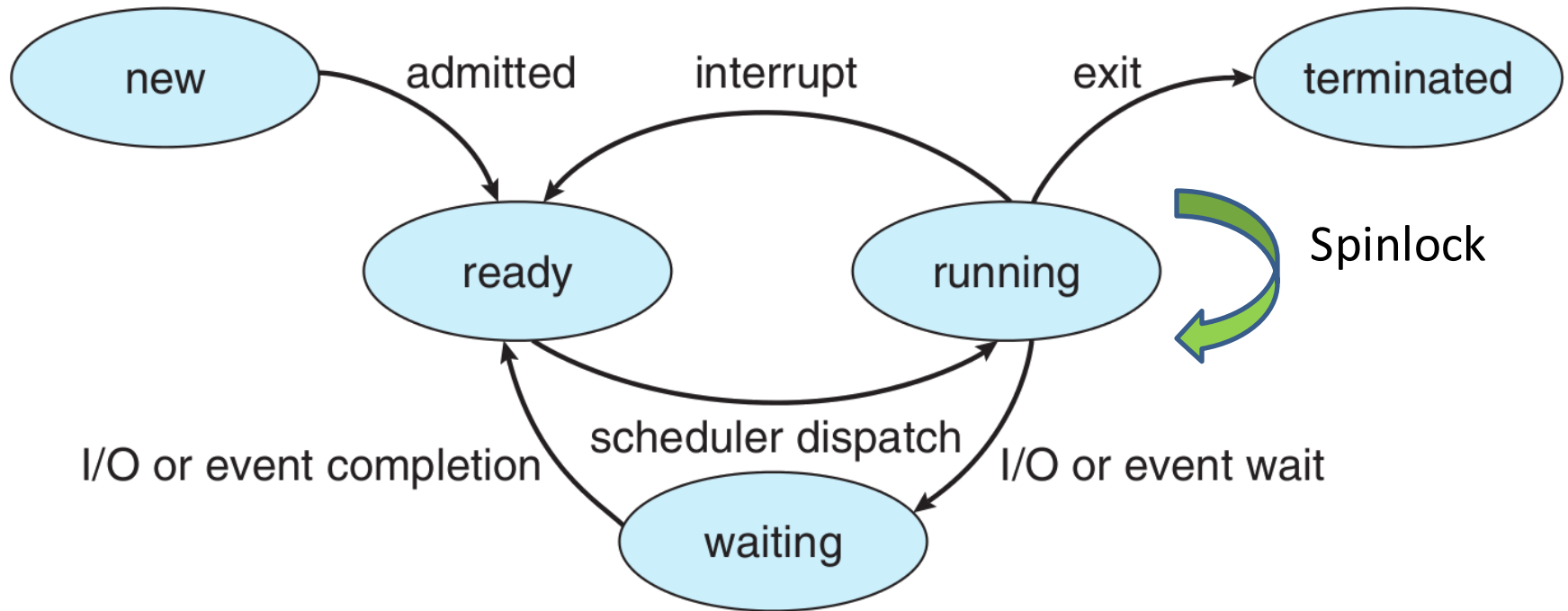


Spinlock
Busy Waiting

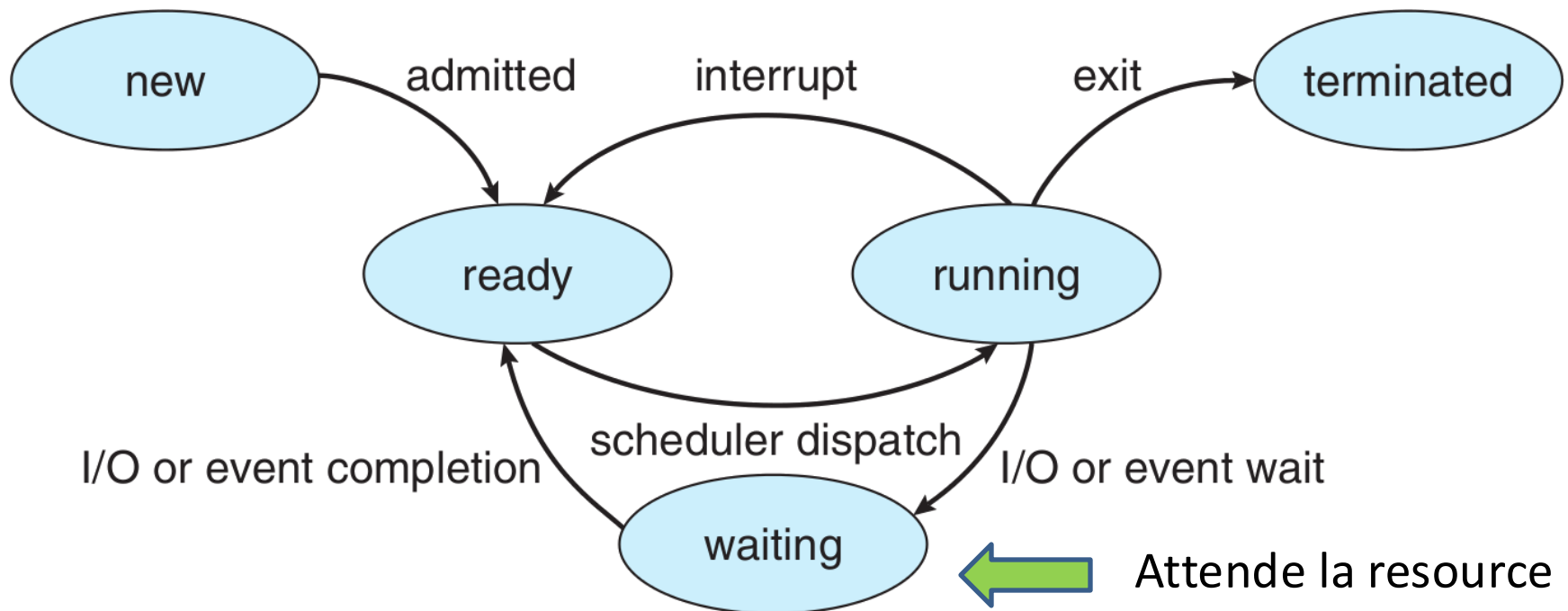
Busy Waiting

- Essayer en permanence de tester la valeur de la variable
- Le CPU fait des instructions inutiles
- Le processus va consommer tout le quantum de temps pour tester la valeur

Les états de processus



Les états de processus



Sémaphore

- Object du Système d'Exploitation
- Partage entre plusieurs processus
- Fonctions exécuté atomique
 - P (attendre) / wait / down
 - V (signaler) / signal / up
- Value initiale
- Queue de processus qui attende

Fonctions de sémaphore

```
void wait (s)
{
    while (s <= 0);
    Busy waiting

    s = s - 1;
}
```

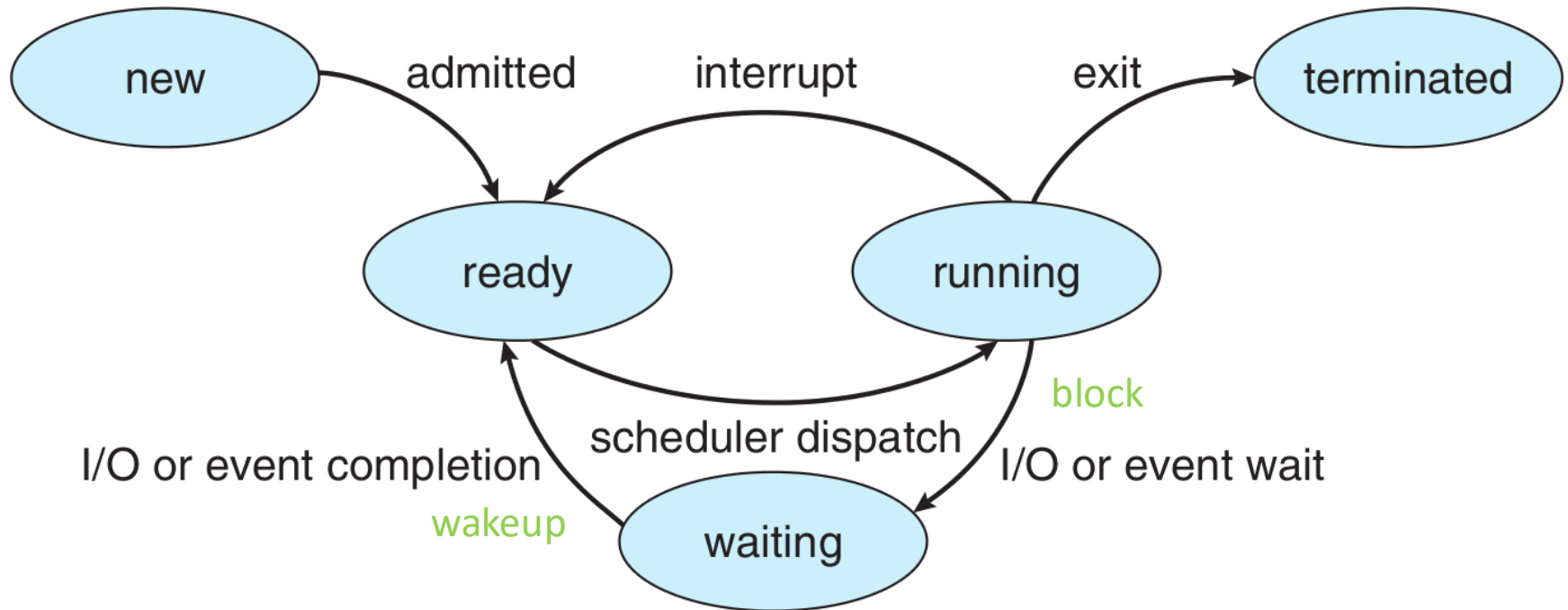
```
void signal (s)
{
    s = s + 1;
}
```

// s - semaphore

block et wakeup

- block
 - arrête le processus curenent
 - état WAITING
- wakeup
 - démarre tous les processus qui sont bloque sure le sémaphore
 - état READY

Les états de processus



Fonction de sémaphore

```
void wait (s)
{
    while (s <= 0)
        block (s);
    s = s - 1;
}
```

```
void signal (s)
{
    s = s + 1;
    wakeup (s);
}
```

// s - semaphore

Types des sémaphores

- binaire
 - la valeur peut être seulement 0 ou 1
- comptage
 - La valeur est une valeur entier

Sémaphore binaire

```
void wait (s)
{
    while (s == 0)
        block (s);
    s = 0;
}
```

```
void signal (s)
{
    s = 1;
    wakeup (s);
}
```

// s - semaphore

Exemple

Processus 1

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
wait (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
signal (s);
```

```
printf ("%d\n", a);
```

Processus 2

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
wait (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
signal (s);
```

```
printf ("%d\n", a);
```

Exemple

Processus 1

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
wait (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
signal (s);
```

```
printf ("%d\n", a);
```

1

Processus 2

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
wait (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
signal (s);
```

```
printf ("%d\n", a);
```

1

Deadlock

Que se passe-t-il si nous oublions d'utiliser le signal?

Que se passe-t-il si nous utilisons plusieurs de fois wait sans signal?

Que se passe-t-il si le program s'arrête avant signal?

Oublier signal

Processus 1

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
wait (s);
```

Processus 2

// shared variables a and s

```
int a = 0;
```

```
int s;
```

```
wait (s);
```

```
if (a == 0)
```

```
{
```

```
    a++;
```

```
}
```

```
printf ("%d\n", a);
```

... waits a lot

...

Plusieurs de wait

Processus 1

// shared variables a and s

int a = 0;

int s;

wait (s);

... waits a lot

...

Processus 2

// shared variables a and s

int a = 0;

int s;

wait (s);

wait (s);

...waits a lot

...

Arrête avent signal

Processus 1

// shared variables a and s

int a = 0;

int s;

wait (s);

Processus 2

// shared variables a and s

int a = 0;

int s;

wait (s);

if (a == 0)

{

 a++;

}

... waits a lot

...

erreur

MUTEX

Mutex

- *Mutual Exclusion*
- Similaire avec le sémaphore binaire
- Utilisez pour threads (même processus)

lock et unlock

- lock
 - prend le mutex et mémorise le id de thread qui a le pris
 - la fonction lock est réentrante dans le même thread
- unlock
 - libéré le mutex
 - au fin du program (thread) le mutex est libéré

Deadlock

Que se passe-t-il si nous oublions d'utiliser le unlock?

Que se passe-t-il si nous utilisons plusieurs de fois wait sans lock?

Que se passe-t-il si le program s'arrête avant unlock?

Oublier unlock

Processus 1

```
// shared variables a and s  
int a = 0;  
mutex s;  
lock (s);
```

```
if (a == 0)  
{  
    a++;  
}  
unlock (s);  
printf ("%d\n", a);
```

1

Processus 2

```
// shared variables a and s  
int a = 0;  
mutex s;  
lock (s);  
if (a == 0)  
{  
    a++;  
}  
  
printf ("%d\n", a);
```

1

Plusieurs de lock

Processus 1

// shared variables a and s

```
int a = 0;  
mutex s;  
lock (s);
```

```
if (a == 0)  
{  
    a++;  
}  
unlock (s);  
printf ("%d\n", a);
```

1

Processus 2

// shared variables a and s

```
int a = 0;  
mutex s;  
lock (s);  
lock (s);  
if (a == 0)  
{  
    a++;  
}  
unlock (s);  
printf ("%d\n", a);
```

1

Arrête avant unlock

Processus 1

```
// shared variables a and s  
int a = 0;  
mutex s;  
lock (s);
```

Processus 2

```
// shared variables a and s  
int a = 0;  
mutex s;  
lock (s);  
if (a == 0)  
{  
    a++;  
}
```

```
if (a == 0)  
{  
    a++;  
}  
unlock (s);  
printf ("%d\n", a);
```

Synchronization en Java

```
class Run
{
    private Object a;

    public synchronized void runAlone ()
    {
        // run some work
    }

    public void run ()
    {
        synchronized (a)
        {
            // run some work using the variable a
        }
    }
}
```

- course
- espace critique
- opération atomique
- busy waiting
- spinlock
- lock
- unlock
- sémaphore
- sémaphore binaire
- sémaphore comptage
- deadlock
- wait
- signal
- mutex

Questions

