Operating Systems : Synchronisation

Gabriel Laskar <gabriel@lse.epita.fr>

LSE Security System

Exercices

- When fork(2) can fail?
- What are the errno values for close(2)?
- What are the differences between uid/euid/gid/egid/tid/tgid?

LSE Security System

Why do we need synchronisation?

- Communication between processes:
  - dependencies
  - serialization
- Concurrent access to data needs to be synchronized to avoid data corruption
- 2 types of synchronisation:
  - inter-process
  - intra-process

LSE Security System

Critical Section

```
int func()
{
   non_critical_stuff();

   enter_critical_section();
      critical_section();
   exit_critical_section();

   non_critical_stuff();
}
```

LSE Security System

Critical Section (continued)

- only one task is executing critical section at one time
- ready tasks must not be blocked by non-asking tasks
- task must not wait indefinitely to enter inside a critical section

Lamport's Bakery Algorithm

```
// declaration and initial values of global variables
Entering: array [NUM_THREADS] of bool = {false};
Number: array [NUM_THREADS] of integer = {0};

lock(integer i) {
      Entering[i] = true;
      Number[i] = 1 + max(Number[1], ..., Number[NUM_THREADS]);
      Entering[i] = false;
      for (j = 0; j < NUM_THREADS; j++) {
            // Wait until thread j receives its number:
            while (Entering[j]) { /* nothing */ }
            // Wait until all threads with smaller numbers or with the same
            // number, but with higher priority, finish their work:
            while ((Number[j] != 0) && ((Number[j], j) < (Number[i], i))) { /* nothing */ }
      }
}

unlock(integer i) {
      Number[i] = 0;
}
```

LSE Security System

Hardware support for critical sections

- forbid interruptions inside a critical section
  - can’t be done safely in userland
  - don't work on multi-core systems
  - disables clocks
- We must have atomic instructions
  - Test And Set (TAS)
  - Swap

Implementation with TAS

```
lock() {
    wait[i] = true;
    is_locked = true;
    while (wait[i] && is_locked) {
        is_locked = tas(lock);
    }
    wait[i] = false;
}
```

```
unlock() {
    j = (i + 1) % N;
    while (i != j && !wait[j]) {
        j = (j + 1) % N;
    }

    if (i == j)
        lock = false;
    else
        wait[j] = false;
}
```

LSE Security System

Issues

- Busy waiting: spinning lock
- need for priority inversion support

LSE Security System

Other synchronisation mechanisms

- Semaphores (Dijkstra, 1965)
- Monitors (Hoare, 1974)
- Mutexes
- Condition Variables
- Barriers

Applications

- Limited buffer size
- Counting semaphores
- Producer/Consumer
- …

LSE Security System

Mutexes

```c
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

POSIX Semaphores - sem_overview(7)

```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag);
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
```

LSE Security System

POSIX Message Queues - mq_overview(7)

```
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag);
int mq_send(mqd_t mqdes, const char *msg_ptr,
      size_t msg_len, unsigned int msg_prio);
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr,
      size_t msg_len, unsigned int *msg_prio);
```

LSE Security System