

Sistemas Operacionais II N

Primitivas de Sincronização e Semáforos

Hoje

- Mutex begin / Mutex end / locks
- Sleep / Wakeup (segundo Tanenbaum)
- Semáforos
 - Evolução
 - Semântica
 - Implementação
 - Semáforos nomeados POSIX
 - Padrões básicos de uso
 - Aplicações

Sincronização de threads

- **pthread_join()** garante que o processo pai espere até a thread em argumento terminar sua execução.
- **Mutexes:** Semáforos binários.
 - Garante que não haja corrida.
 - Deveria ser usado em cada dado que é compartilhado.
 - `pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;`
 - `pthread_mutex_lock(&mutex);`
 - `pthread_mutex_unlock(&mutex);`

Sleep & wakeup perdendo sinais

```
#define N 100
int count = 0;

void producer(void)
{
    while (true)
    {
        produce_item();
        if (count == N)
            sleep();
        enter_item();
        ++ count;
        if (count == 1)
            wakeup(consumer);
    }
}

void consumer(void)
{
    while (true)
    {
        if (count == 0)
            sleep();
        remove_item();
        -- count;
        if (count == N-1)
            wakeup(producer);
        consume_item();
    }
}
```

Semáforos

sleep() P() down() sem_wait()

wakeup() V() up() sem_post()

Semáforos - Semântica

P(s) : espera até $s > 0$
decrementa s

V(s) : incrementa s

Semáforos - Implementação

```
P(s) : if (countS==0)
        insere atual na filaS;
    else
        --countS;

V(s) : if (filaS vazia)
        ++countS;
    else
        retira primeiro da filaS
        e insere na ReadyList
```

Semáforos POSIX

- Named semaphores
- Kernel-persistent
- Need open/close, and init

- testes

Aplicações Problemas e Soluções com Semáforos

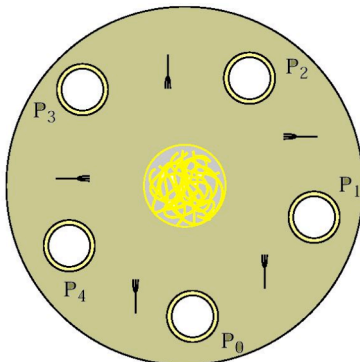
Produtor e Consumidor em Buffer Limitado

```
#define N 100
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer(void)
{
    while (true)
    {
        produce_item(&item);
        down(&empty);
        down(&mutex);
        enter_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void)
{
    while (true)
    {
        down(&full);
        down(&mutex);
        remove_item(&item);
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

Problema dos filósofos



Problema dos filósofos

```
#define N 5
#define LEFT(i) (i+N-1)%N
#define RIGHT(i) (i+1)%N
#define THINKING 0
#define HUNGRY 1
#define EATING 2
int state[N];
sema_t mutex; // = 1
sema_t Sem[N]; // = 0
void philosopher(int i) {
    while(TRUE){
        think();
        take_forks(i);
        eat();
        put_forks(i);
    }
}
void take_forks(int i) {
    sema_wait(&mutex);
    state[i] = HUNGRY;
    test(i);
    sema_post(&mutex);
    sema_wait(&Sem[i]);
}
void put_forks(int i)
{
    sema_wait(&mutex);
    state[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    sema_post(&mutex);
}
void test(int i)
{
    if ( state[i] == HUNGRY &&
        state[LEFT(i)]!=EATING &&
        state[RIGHT(i)]!=EATING )
    {
        state[i] = EATING;
        sema_post(&Sem[i]);
    }
}
```

Problema dos leitores e escritores

```

#include <stdio.h>
#define TRUE 1

sema_t mutex; // = 1
sema_t db; // = 1
int rc = 0;

void writer(void)
{
    while (TRUE)
    {
        make_data();
        sema_wait(&db);
        write_data();
        sema_post(&db);
    }
}

void reader(int i)
{
    while (TRUE)
    {
        sema_wait(&mutex);
        ++ rc;
        if (rc == 1)
            sema_wait(&db);
        sema_post(&mutex);

        read_database();

        sema_wait(&mutex);
        -- rc;
        if (rc == 0)
            sema_post(&db);
        sema_post(&mutex);
    }
}

```

INF01151 - Sistemas Operacionais II N - Marcelo Johann - 2009/1

Aula 07 : Slide 13

```

sema_t mutex; // = 1
sema_t dbw; // = 1
int nr = 0;
int nw = 0;
sema_t mx; // = 1
sema_t dbr; // = 1

void writer(void) {
    while (TRUE) {
        sema_wait(&mx);
        ++ nw;
        if (nw == 1)
            sema_wait(&dbr);
        sema_post(&mx);

        sema_wait(&dbw);
        write_data();
        sema_post(&dbw);

        sema_wait(&mx);
        -- nw;
        if (nw == 0)
            sema_post(&dbr);
        sema_post(&mx);
    }
}

void reader(int i)
{
    while (TRUE)
    {
        sema_wait(&dbr);
        sema_wait(&mutex);
        ++ rc;
        if (rc == 1)
            sema_wait(&dbw);
        sema_post(&mutex);
        sema_post(&dbr);

        read_database();

        sema_wait(&mutex);
        -- rc;
        if (rc == 0)
            sema_post(&dbw);
        sema_post(&mutex);
    }
}

```

INF01151 - Sistemas Operacionais II N - Marcelo Johann - 2009/1

Aula 07 : Slide 14

```

sema_t mutex; // = 1
sema_t dbw; // = 1
int nr = 0;
int nw = 0;
sema_t mx; // = 1
sema_t dbr; // = 1
sema_t mz; // = 1
void writer(void) {
    while (TRUE) {
        sema_wait(&mx);
        ++ nw;
        if (nw == 1)
            sema_wait(&dbr);
        sema_post(&mx);

        sema_wait(&dbw);
        write_data();
        sema_post(&dbw);

        sema_wait(&mx);
        -- nw;
        if (nw == 0)
            sema_post(&dbr);
        sema_post(&mx);
    }
}

void reader(int i)
{
    while (TRUE)
    {
        sema_wait(&mz);
        sema_wait(&dbr);
        sema_wait(&mutex);
        ++ rc;
        if (rc == 1)
            sema_wait(&dbw);
        sema_post(&mutex);
        sema_post(&dbr);
        sema_post(&mz);
        read_database();
        sema_wait(&mutex);
        -- rc;
        if (rc == 0)
            sema_post(&dbw);
        sema_post(&mutex);
    }
}

```

INF01151 - Sistemas Operacionais II N - Marcelo Johann - 2009/1

Aula 07 : Slide 15