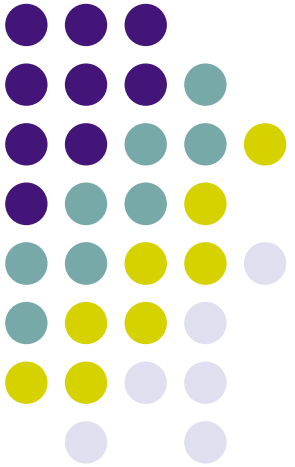
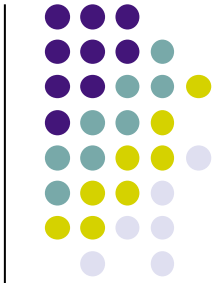
Sistemas de Computação

Semáforos



Semáforos



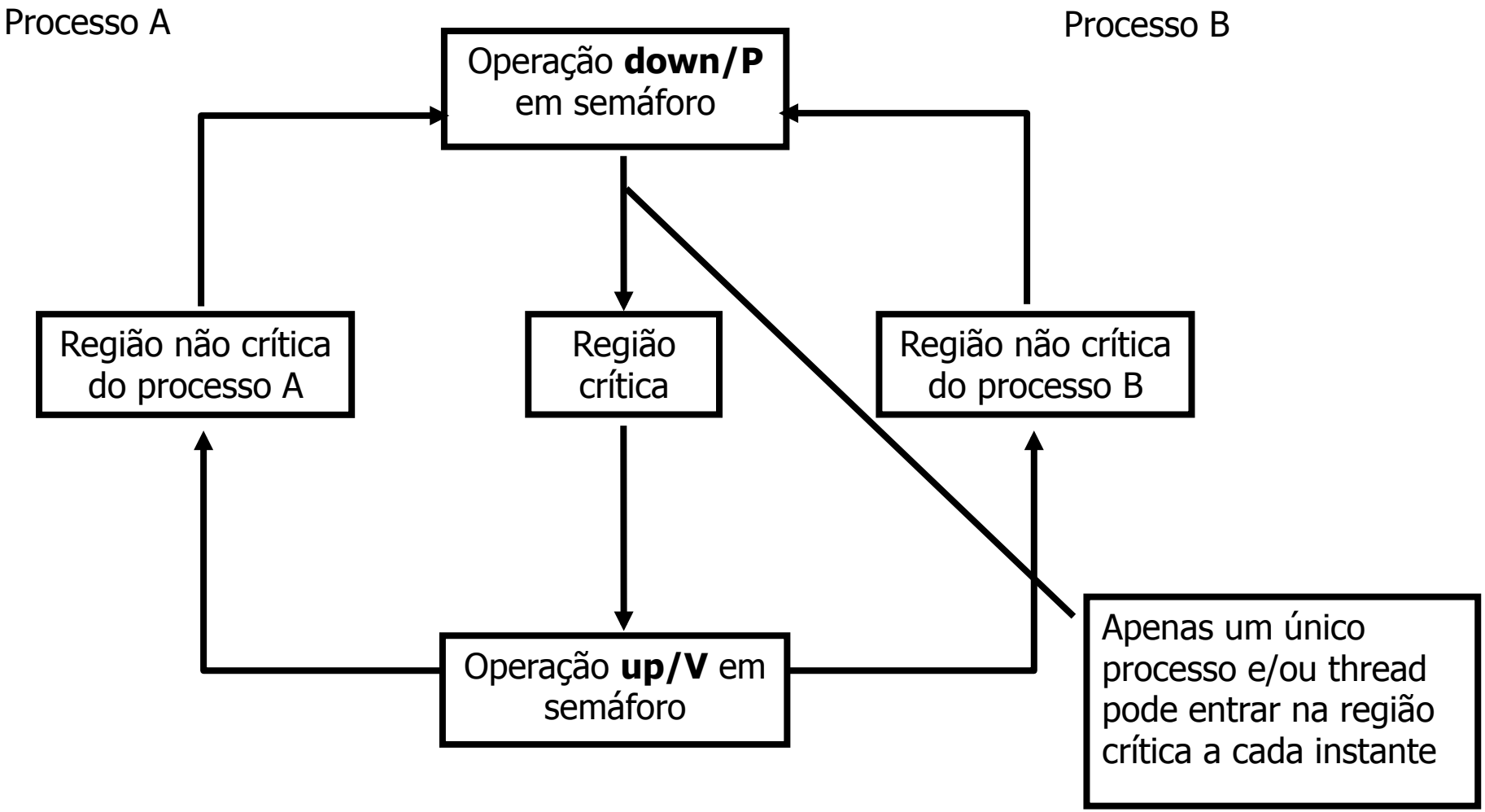
- Um semáforo simples é uma variável que pode assumir os valores 0 e 1 (semáforo binário). Esta é a forma mais comum

- Semáforos que podem assumir diversos valores são chamados de **semáforos contadores**

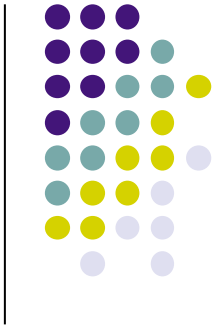
- Semáforos binários (mutex) são utilizados para garantir que somente um processo (ou thread) tenha acesso a uma região crítica ou recurso a cada instante

- Semáforos contadores permitem limitar a quantidade de processos e/ou threads que utilizarão o recurso

Acesso a região crítica



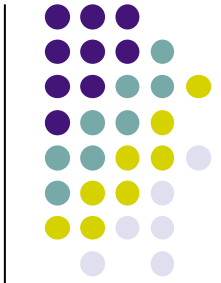
Semáforos – Um exemplo



```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE is the constant 1 */
          item = produce_item( );                /* generate something to put in buffer */
          down(&empty);                         /* decrement empty count */
          down(&mutex);                         /* enter critical region */
          insert_item(item);                    /* put new item in buffer */
          up(&mutex);                           /* leave critical region */
          up(&full);                            /* increment count of full slots */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* infinite loop */
          down(&full);                          /* decrement full count */
          down(&mutex);                         /* enter critical region */
          item = remove_item( );                /* take item from buffer */
          up(&mutex);                           /* leave critical region */
          up(&empty);                           /* increment count of empty slots */
          consume_item(item);                   /* do something with the item */
     }
}
```

O problema Produtor-Consumidor usando 1 mutex e 2 semáforos

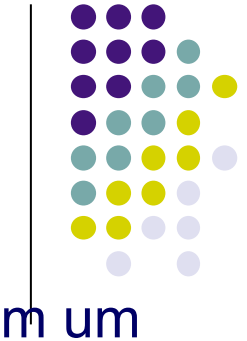
Semáforos em Unix



- Todos as funções de semáforos operam sobre vetores (arrays) de semáforos contadores
- Assim, múltiplos recursos podem ser alocados/ controlados simultaneamente
- As funções de semáforo são:

```
#include <sys/sem.h>

int semget(key_t key, int num_sems, int sem_flags);

int semop(int sem_id, struct sembuf *sem_ops, size_t
num_sem_ops);

int semctl(int sem_id, int sem_num, int command, ...);
```

Função semget()

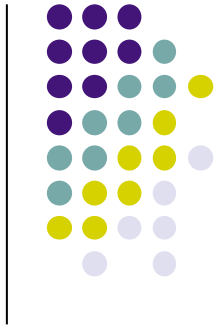


- A função semget() cria um novo (array de) semáforo ou obtém um identificador para já existente
  - Definida em <sys/sem.h>

    int semget(key_t key, int num_sems, int sem_flags);

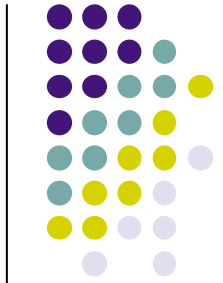
  - Onde:
    - key : um valor inteiro utilizado para permitir que processos não relacionados entre si acessem o mesmo semáforo
    - num_sems : número de semáforos a serem criados, normalmente 1
    - sem_flags : permissões (0xxx) e modo de criação do semáforo
      - IPC_PRIVATE : somente o processo que criou o semáforo pode utilizá-lo
      - IPC_CREAT : cria o semáforo, se não existir
      - IPC_EXCL : permite verificar se o semáforo já existe
  - Retorna:
    - Em caso de sucesso, retorna ipc_id, o identificador do (array de) semáforo
    - Em caso de falha, -1

Função semget( )



- Todos os semáforos do array são acessados de forma atómica indicando o identificador retornada por semget()
- A identificação somente pode ser gerada por semget()
- Todas as outras funções de semáforo utilizam este identificador retornado por semget()

Resumo das flags



- Com IPC_CREAT | IPC_EXCL
  - Cria um conjunto de semáforos (se este não existir);
  - Permite verificar se um dado conjunto existe ou não.

- Sem as flags IPC_CREAT e IPC_EXCL (sem ambas)
  - Obtém o conjunto de semáforos (se existir);
  - Se já existir o conjunto e o número de semáforos for inferior a *nsems*, retorna erro.

- Com IPC_CREAT e sem IPC_EXCL
  - Obtém o conjunto de semáforos (se existir), cria se não existir;
  - Se já existir o conjunto e o seu número de semáforos for inferior a *nsems*, retorna erro.

- Nota: Na operação de abertura de um array de semáforos, devem ser respeitadas as permissões de acesso, isto é, as *flags* de permissão presentes no argumento *flags* (escritas na forma 0xxx) devem conter aquelas presentes no conjunto de semáforos quando definidas na sua criação.

Função semop()

- A função semop() é utilizada para modificar o valor de um semáforo:
  - Definida em <sys/sem.h>

```
int semop(int semid, struct sembuf *sops, size_t
  nsops);
```

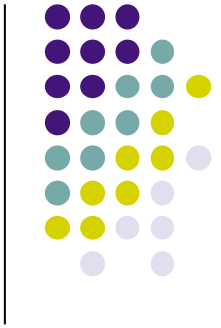
Onde:

- semid: identificador do semáforo retornado por semget()
- sops: ponteiro para o vetor de estruturas
- nsops: número de elementos do vetor de estruturas sembuf:

```
struct sembuf
{
        short sem_num;          // valor do semáforo
        short sem_op;           // operação do semáforo
        short sem_flg;          // flags da operação
}
```

- Retorna:
  - Em caso de sucesso: 0 (zero)
  - Em caso de falha: -1

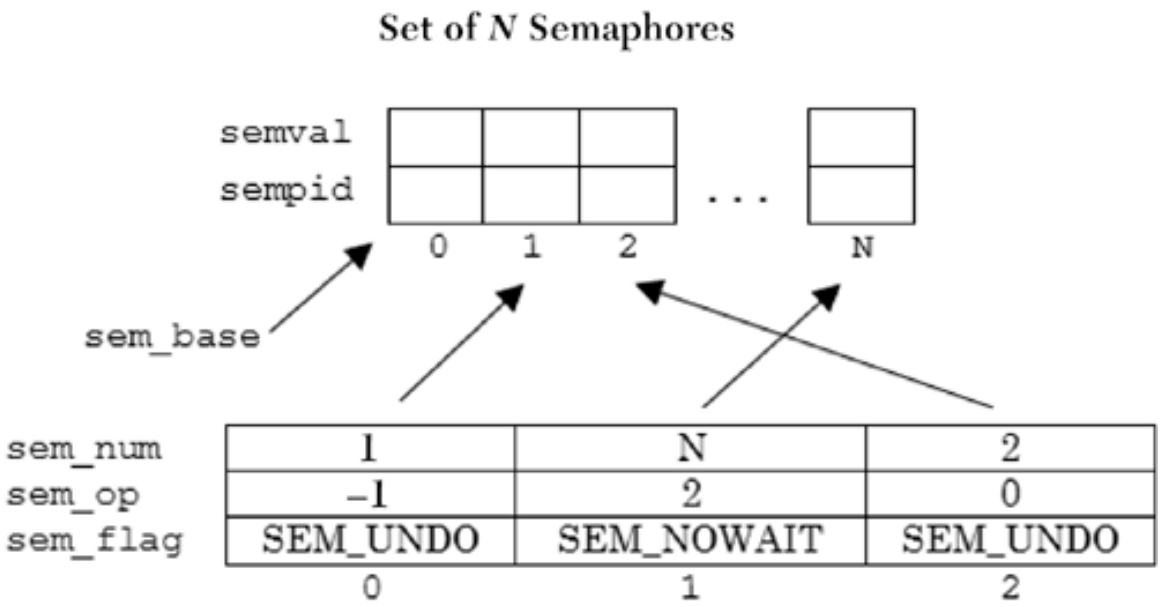
Estrutura sembuf



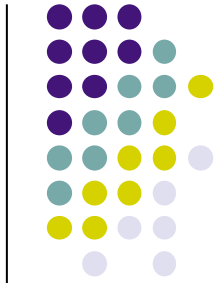
```
struct sembuf
{
  short sem_num; //valor do semáforo
  short sem_op;  //operação do semáforo
  short sem_flg; //flags da operação
}
```

- Onde:
  - sem_num: valor do semáforo, normalmente 0 (zero)
  - sem_op: normalmente -1 para realizar down/P e +1 para realizar up/V
  - sem_flg: geralmente SEM_UNDO para que o sistema operacional verifique o estado do semáforo e mantenha o sistema funcionando mesmo se o processo terminar com o semáforo em lock

Função semop()



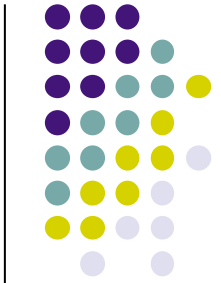
Função semop()



- Se semop() retornar um valor diferente de 0 (zero), o processo ou a thread será suspensa

- O processo ou a thread sairá do estado de suspenso se:
  - O valor do semáforo passar a ser 0
  - Ou o semáforo for removido do sistema

Função semctl()



Opera diretamente o semáforo

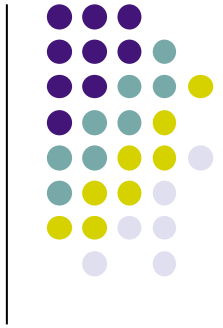
```
int semctl(int semid, int semnum, int cmd);
int semctl(int semid, int semnum, int cmd, union semun arg);
```

Onde:

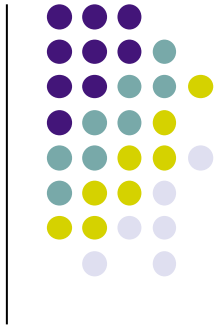
- semid: identificador do semáforo retornado por semget()
- semnum: valor do semáforo, normalmente 0
- cmd: ação a ser tomada
  - Existem vários valores possíveis para **`cmd`**, mas dois são os mais comuns:
    - SETVAL: utilizado para inicializar o semáforo com um valor conhecido.
      O valor é passado no membro **`val`** da **`union semun`**
    - IPC_RMID: utilizado para remover um semáforo que não é mais necessário
  - arg: (opcional) é do tipo **`union semun`**

```
union semun
{
  int val;
  struct semid_ds *buf;
  unsigned short *array;
}
```

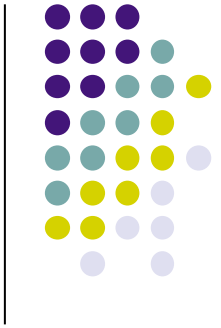
- Retorna:
  - 0 : em caso de sucesso
  - -1 : em caso de erro



```c
/* Exemplo de uso de unico semáforo*/

#include <sys/sem.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

union semun
{
   int val;
   struct semid_ds *buf;
   ushort *array;
};

// inicializa o valor do semáforo
int setSemValue(int semId);
// remove o semáforo
void delSemValue(int semId);
// operação P
int semaforoP(int semId);
//operação V
int semaforoV(int semId);
```

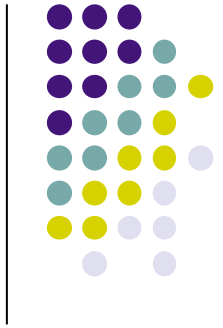


```
int main (int argc, char * argv[])
{
   int i;
   char letra = 'o';
   int semId;

   if (argc > 1)
   {
            semId = semget (8752, 1, 0666 | IPC_CREAT);
            setSemValue(semId);
            letra = 'x';
            sleep (2);
   }
   else
   {
            while ((semId = semget (8752, 1, 0666)) < 0)
            {
                  putchar ('.'); fflush(stdout);
                  sleep (1);
            }
   }
```

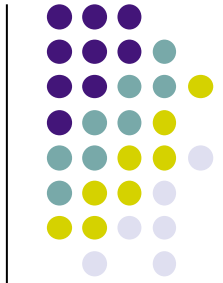


```
   for (i=0; i<10; i++)
   {
            semaforoP(semId);
            putchar (toupper(letra)); fflush(stdout);
            sleep (rand() %3);
            putchar (letra); fflush(stdout);
            semaforoV(semId);
            sleep (rand() %2);
   }

   printf ("\nProcesso %d terminou\n", getpid());

   if (argc > 1)
   {
            sleep(10);
            delSemValue(semId);
   }

   return 0;
}
```

```
int setSemValue(int semId)
{
   union semun semUnion;
   semUnion.val = 1;
   return semctl(semId, 0, SETVAL, semUnion);
}
void delSemValue(int semId)
{
   union semun semUnion;
   semctl(semId, 0, IPC_RMID, semUnion);
}
int semaforoP(int semId)
{
   struct sembuf semB;
   semB.sem_num = 0;
   semB.sem_op = -1;
   semB.sem_flg = SEM_UNDO;
   semop(semId, &semB, 1);
   return 0;
}
int semaforoV(int semId)
{
   struct sembuf semB;
   semB.sem_num = 0;
   semB.sem_op = 1;
   semB.sem_flg = SEM_UNDO;
   semop(semId, &semB, 1);
   return 0;
}
```



Exemplo (semaforo.c):



```
$ ./semaforo &
.[1] 7503
$ ./.semaforo ...1..
OoOoXxOoXxOoXxOoXxOoXxOoXxOoXxOoOo
Processo 7503 terminou
XxXxXx
Processo 7504 terminou
[1]+  Done                    ./semaforo
$
```

- ./semáforo & /* argc < 1*/

  executa em background;

  imprime O ao entrar na região crítica e o ao sair.

- ./semaforo 1 /* argc > 1 */

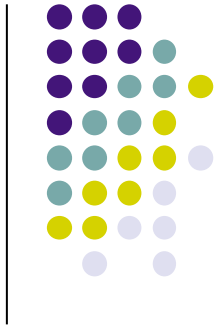
  executa em foreground;

  cria o semáforo (ao fazer isso, dispara a execução do semáforo em background - que indicava erro pois o semáforo não tinha sido criado);

  imprime X ao entrar na região crítica e x ao sair.

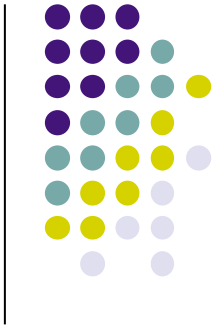
Perguntas?

Exercício



1) Execute o programa dado (exemplo de uso de semáforos) e explique sua execução.

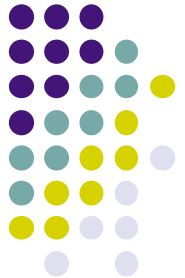
Exercício



2) Produtor-Consumidor

- Escreva um programa formado por dois processos concorrentes, leitor e impressor, que usam memória compartilhada executando um loop infinito. Para sincronizar as suas ações, eles fazem uso de semáforos.
- O processo leitor fica lendo caracteres da entrada padrão e colocando em um buffer de 16 posições. Quando o buffer está cheio o processo impressor deve imprimi-lo na saída padrão.

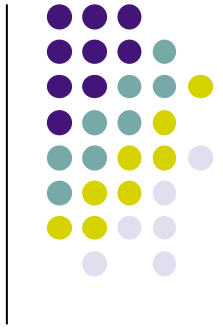
Exercício:



3) Faça programas para alterar um valor de uma variável na memória compartilhada.

   Um programa soma 1 à variável e o outro soma 5 à variável. Utilize semáforos para alterar a variável (região crítica).

Desafio:



4) Faça programas separados que utilizam a memória compartilhada para trocar mensagens. Utilize semáforos para sincronizar a aplicação.