UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Semáforos

Walter Fetter Lages

w.fetter@ieee.org

Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Engenharia Elétrica

ELE213 Programação de Sistemas de Tempo Real

Copyright (c) Walter Fetter Lages

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Introdução

- Conceito introduzido por Dijkstra (1968)
- Simplificam os protocolos para sincronização
- Eliminam a necessidade de espera-ocupada
- Um semáforo é uma variável inteira, não negativa na qual se pode fazer apenas duas operações (além da inicialização):
  - `P(s)=wait(s)`
  - `V(s)=signal(s)`

Copyright (c) Walter Fetter Lages – p.2

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Operações nos Semáforos

- `P(s)=wait(s)`
  - Se `s` > 0 decrementa `s`, senão suspende a tarefa
- `V(s)=signal(s)`
  - Se existe alguma tarefa suspenso em `s`, libera um deles, senão incrementa `s`
    - Não é especificado qual proceso é liberado
- As operações nos semáforos são atômicas
- Semáforos binários
- Semáforos contadores

Copyright (c) Walter Fetter Lages – p.3

Exemplo de Sincronização

```
void p1(void)
{
    ...
    wait(s1);
    ...
}
```

```
void p2(void)
{
    ...
    signal(s1);
    ...
}
```

Copyright (c) Walter Fetter Lages – p.4

Exemplo de Exclusão Mútua

```
void p1(void)
{
    for(;;)
    {
        wait(s1);
     /*seção crítica*/
        signal(s1);
      ...
    }
    ...
}
```

```
void p2(void)
{
    for(;;)
    {
        wait(s1);
      /*seção crítica*/
        signal(s1);
      ...
    }
    ...
}
```

Copyright (c) Walter Fetter Lages – p.5

Semáforos POSIX

- POSIX 1003.1b
- Semáforos com nome
  - na forma `/nome`
  - até `NAME_MAX-4` (251) caracteres
- Semáforos sem nome
- Operações `P(s)` e `V(s)`:

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

Copyright (c) Walter Fetter Lages – p.6

Semáforos POSIX com Nome

- Criação e destruição

```
#include <semaphore.h>

sem_t *sem_open(const char *name,int oflag);
sem_t *sem_open(const char *name,int oflag,
        mode_t mode,unsigned int value);
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
```

- *flags*
  - O_CREAT
  - O_EXCL
- Semáforos com nome devem ser removidos quando não mais necessários

Copyright (c) Walter Fetter Lages – p.7

Exemplo de Sincronização

```
#include <semaphore.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdio.h>

void *task1(void *ptr)
{
        sem_t *sem;

        sem=sem_open("/semsync",O_CREAT,S_IRUSR|S_IWUSR,0);
        sem_wait(sem);
        printf("Mensagem de task1.\n");
        sem_close(sem);
        sem_unlink("/semsync");
        return NULL;
}
```

Copyright (c) Walter Fetter Lages – p.8

Exemplo de Sincronização

```
void *task2(void *ptr)
{
        sem_t *sem;

        sem=sem_open("/semsync",O_CREAT,S_IRUSR|S_IWUSR,0);
        printf("Mensagem de task2.\n");
        sem_post(sem);
        sem_close(sem);
        return NULL;
}
```

Copyright (c) Walter Fetter Lages – p.9

Semáforos POSIX sem Nome

- Exigem memória compartilhada entre as tarefas
- Criação e destruição

```
#include <semaphore.h>

int sem_init(sem_t *sem,int pshared,
      unsigned int value);
int sem_destroy(sem_t *sem);
```

Copyright (c) Walter Fetter Lages – p.10

Exemplo de Exlusão Mútua

```
#include <pthread.h>
#include <semaphore.h>
#include "sections.h"

void *task1(void *ptr)
{
        for(;;)
        {
                sem_wait(ptr);
                critical1();
                sem_post(ptr);
                noncritical();
        }
}
```

Copyright (c) Walter Fetter Lages – p.11

Exemplo de Exlusão Mútua

```
void *task2(void *ptr)
{
        for(;;)
        {
                sem_wait(ptr);
                critical2();
                sem_post(ptr);
                noncritical();
        }
}
```

Copyright (c) Walter Fetter Lages – p.12

Exemplo de Exlusão Mútua

```
int main(int arc, char *argv[])
{
        pthread_t task1hdl;
        pthread_t task2hdl;
        sem_t sem;

        sem_init(&sem,0,1);
        pthread_create(&task1hdl,NULL,task1,&sem);
        pthread_create(&task2hdl,NULL,task2,&sem);
        pthread_join(task1hdl,NULL);
        pthread_join(task2hdl,NULL);
        sem_destroy(&sem);
        return 0;
}
```

Copyright (c) Walter Fetter Lages – p.13

Outras Funções

- degeneram o conceito de semáforo
- devem ser usadas com cautela

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
int sem_trywait(sem_t *sem);
int sem_timedwait(sem_t *sem,
        const struct timespec *abs_timeout);
int sem_post(sem_t *sem);
int sem_getvalue(sem_t *sem,int *sval);
```

Copyright (c) Walter Fetter Lages – p.14

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

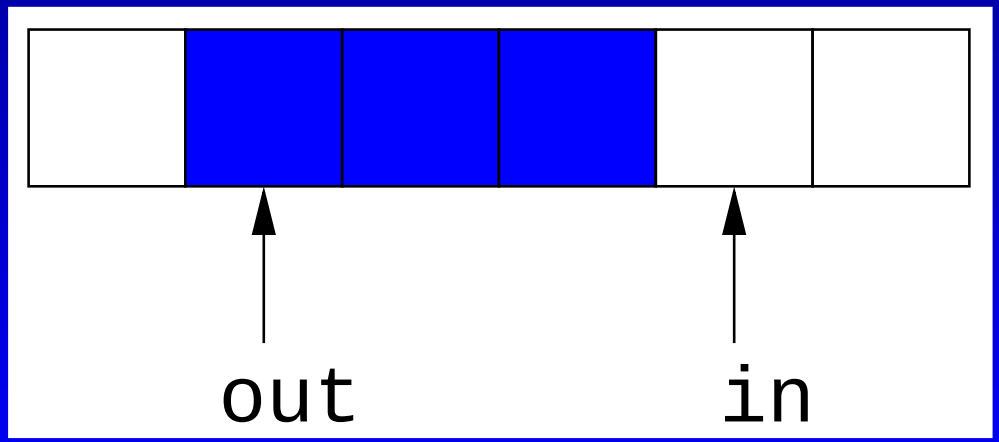
Problema Produtor-Consumidor

- Uma (ou mais) tarefa(s) produz(em) dados
- Outra(s) tarefa(s) consome(m) os dados
- Produtores e consumidores podem operar em velocidades instantâneas diferentes
- É conveniente ter um *buffer* entre os dois
  - Mantém ambos funcionando na velocidade ótima

Copyright (c) Walter Fetter Lages – p.15

Buffer Circular

- Gerenciado através de dois ponteiros
- Ambiguidade quando os dois ponteiros são coincidentes
  - *buffer* totalmente cheio?
  - *buffer* totalmente vazio?
  - solução: no mínimo 1 *slot* sempre vazio ou variável de contagem



Copyright (c) Walter Fetter Lages – p.16

Arquivo de Cabeçalho

```
#ifndef BUFFER_H
#define BUFFER_H

#define BUFFER_SIZE 20

typedef struct
{
        int in;
        int out;
        int buf[BUFFER_SIZE];
} buffer;

extern void buffer_init(buffer *b);
extern int buffer_count(const buffer *b);
extern int buffer_append(int item,buffer *b);
extern int buffer_take(int *item,buffer *b);
#endif
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Funções do *Buffer* Circular

```
#include "buffer.h"

int buffer_count(const buffer *b)
{
        int count;

        count=b->in-b->out;
        if(b->in < b->out) count+=BUFFER_SIZE;
        return count;
}

int buffer_append(int item,buffer *b)
{
        if(buffer_count(b) >= BUFFER_SIZE-1) return -1
        b->buf[b->in]=item;
        b->in=(b->in+1) % BUFFER_SIZE;
        return 0;
}
```

Funções do *Buffer* Circular

```
int buffer_take(int *item,buffer *b)
{
        if(b->in == b->out) return -1;
        *item=b->buf[b->out];
        b->out=(b->out+1) % BUFFER_SIZE;
        return 0;
}

void buffer_init(buffer *b)
{
        b->in=0;
        b->out=0;
}
```

Copyright (c) Walter Fetter Lages – p.19

Headers e Semáforos

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include "buffer.h"

static sem_t exclusion;
static sem_t empty_slot;
static sem_t full_slot;
```

Copyright (c) Walter Fetter Lages – p.20

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Produtor

```
void *producer(void *ptr)
{
        int data=0;

        for(;;)
        {
                data++;                                  /* pro
                sem_wait(&empty_slot);
                sem_wait(&exclusion);
                buffer_append(data,ptr);
                sem_post(&exclusion);
                sem_post(&full_slot);
        }
}
```

Consumidor

```
void *consumer(void *ptr)
{
        int data;

        for(;;)
        {
                sem_wait(&full_slot);
                sem_wait(&exclusion);
                buffer_take(&data,ptr);
                sem_post(&exclusion);
                sem_post(&empty_slot);
                printf("data=%d\n",data);        /* cons
        }
}
```

Copyright (c) Walter Fetter Lages – p.22

Criação das Tarefas

```
int main(int argc,char *argv[])
{
        pthread_t producerhdl;
        pthread_t consumerhdl;
        buffer buf;

        buffer_init(&buf);
        sem_init(&exclusion,0,1);
        sem_init(&full_slot,0,0);
        sem_init(&empty_slot,0,BUFFER_SIZE-1);
        pthread_create(&producerhdl,NULL,producer,&buf);
        pthread_create(&consumerhdl,NULL,consumer,&buf);
        pthread_join(producerhdl,NULL);
        pthread_join(consumerhdl,NULL);
        sem_destroy(&exclusion);
        sem_destroy(&full_slot);
        sem_destroy(&empty_slot);
        return 0;
}
```

Copyright (c) Walter Fetter Lages – p.23

Semáforos POSIX no RTAI

- No RTAI deve-se utilizar o arquivo de cabeçalho `rtai_posix.h` ao invés de `pthreads.h` e `semaphore.h`

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Semáforos System V

- Semáforos são um conjunto de valores de semáforos
- A criação é independente da inicialização
- Semáforos não são destruidos com os processos
  - Semáforos existem no escopo sistema e não no escopo dos processos que os utilizam
- Grande parte das operações realizadas por uma única função

Copyright (c) Walter Fetter Lages – p.25

Criação

```
int semget(key_t key,int nsems,
           int semflg)
```

- Retorna o identificador dos semáforos associado a `key`
- `key` é uma chave única
  - `IPC_PRIVATE` garante a unicidade
- `nsems` é o número de semáforo no conjunto
- `semflg` *bitwise or* entre
  - `IPC_CREAT` cria um novo semáforo
  - modo de acesso: `rwxrwxrwx`
    - `SEM_R`, `SEM_R >> 3`, `SEM_R >> 6`
    - `SEM_W`, `SEM_W >> 3`, `SEM_W >> 6`

Copyright (c) Walter Fetter Lages – p.26

Controle

```
int semctl(int semid,int semnun,int cmd
        union semun arg)

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

- Executa as operações de controle nos semáforos

Copyright (c) Walter Fetter Lages – p.27

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Controle

- `IPC_STAT` copia informação de controle
- `IPC_SET` altera as permissões na estrutura de controle
- `IPC_RMID` remove o semáforo
- `GETVAL` retorna o valor de `semnum`
- `SETVAL` altera o valor de `semnum`
- `GETPID` retorna o `PID` da última operação em `semnum`

Copyright (c) Walter Fetter Lages – p.28

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Controle

- `GETNCNT` retorna o número de processos bloqueados esperando um incremento em `semnum`
- `GETZCNT` retorna o número de processos bloqueados esperando que o valor de `semnum` seja 0
- `GETALL` obtém os valores dos semáfors no conjunto
- `SETALL` altera os valors dos semáfors no conjunto

Copyright (c) Walter Fetter Lages – p.29

Operações

```
int semop(int semid,struct sembuf sops[],
        size_t nsops)

struct sembuf
{
        short sem_num;
        short sem_op;
        short sem_flg;
};
```

- Executa operações atômicas no conjunto de semáforos

Copyright (c) Walter Fetter Lages – p.30

Operações

- `sem_flg` *bitwise or* entre `IPC_NOWAIT` e `SEM_UNDO`
- `sem_op`
  - > 0
    - Corresponde à operação `signal`
    - `sem_op` é somado ao valor do semáforo
    - Se `SEM_UNDO` for especificado, `sem_op` é subraído do valor de ajuste do semáforo para o processo
  - $= 0$ o processo é bloqueado até que
    - O valor do semáforo seja 0, ou
    - O semáforo seja removido do sistema, ou
    - O processo receba um sinal

Copyright (c) Walter Fetter Lages – p.31

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Operações

- `sem_op` < 0
  - Corresponde à operação `wait`
  - Se o valor do semáforo $\geq$ |`sem_op`|
    - O valor absoluto de `sem_op` é subtraído do valor do semáforo
    - Se `SEM_UNDO` for especificado, o |`sem_op`| é somado ao valor de ajuste do semáforo para o processo
  - Se o valor do semáforo é $<$ |`sem_op`|
    - O processo é bloqueado até que
      - O valor do semáforo $\geq$ |`sem_op`|, ou
      - O semáforo seja removido do sistema, ou
      - O processo receba um sinal

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Ajuste de semáforos

- Quando um processo termina as operações de `wait` em semáforos (alocações de recursos) realizadas por els não são desfeitas
- Quando as operações incluem a `flag` `SEM_UNDO`, o *kernel* rastreia as operações de `wait` realizadas e realiza os ajustes quando o processo termina.

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Problemas com Semáforos

- Primitivas de baixo nível
- Basta a omissão de uma ocorrência para que o programa não funcione corretamente
- Semáforos não implementam a exclusão mútua diretamente, apenas um mecanismo que permite a implementação de exclusão mútua
- Semáforos System V são desnecessariamente complicados
- Semáforos POSIX sem nome exigem memória compartilhada entre os processos

Copyright (c) Walter Fetter Lages – p.34