

Comunicação entre processos e sincronização: Semáforos e Queues

Cristiano André da Costa
(cac@unisinis.br)

Roteiro

- Semáforos
 - Estruturas
 - Obtendo, Controlando, Operando
 - Simplificando uso de semáforos
- Queues (Filas de Mensagens)
 - Estruturas
 - Abrindo, Criando, Enviando, Recebendo
 - Exemplo
- Exercícios

2

Semáforos

- Contador usado para fornecer acesso a dados compartilhados por múltiplos processos
- Para usar um recurso compartilhado:
 1. Testar o semáforo que controla o recurso;
 2. Se o valor do semáforo é positivo, o processo usa o recurso. Nesse caso, o processo decrementa o semáforo é decrementado em 1, indicando seu uso;
 3. Caso contrário, se o semáforo é 0, o processo dorme até o valor ser > 0. Quando acorda retorna ao passo 1.

3

Semáforos (cont.)

- Quando o processo termina de usar o recurso compartilhado, o semáforo é incrementado em 1. Se algum processo está dormindo, esperando o semáforo, ele é acordado.
- Para implementar semáforo o teste do valor e o decremento desse valor devem ser uma operação atômica
 - Normalmente implementados pelo kernel
- Semáforos binários e contadores
- Uso de semáforo no Unix é mais complicado...

4

POSIX XSI Semaphores

- Consiste de um vetor de elementos semáforos
- Os elementos semáforos são parecidos, mas não iguais aos clássicos semáforos apresentados em SisOP (do Dijkstra)
- Cada conjunto de semáforo mantém uma estrutura `semid_ds`
- Cada elemento semáforo inclui algumas informações: valor, processos esperando...

5

Estrutura `semid_ds`

- Para cada conjunto de semáforos:

```
struct semid_ds {
    struct ipc_perm sem_perm; /* permissões */
    unsigned short sem_nsems; /* número de semáforos no
                               conjunto */
    time_t sem_otime; /* tempo da última semop */
    time_t sem_ctime; /* tempo da última semctl */
    ...
}
```

- A estrutura está definida em `sys/sem.h`

6

Estrutura do elemento semáforo

- Para cada elemento semáforo tem-se:

```
struct {
    unsigned short semval; /* valor do semáforo */
    pid_t sempid; /* último PID a manipulá-lo */
    unsigned short semncnt; /* número de processos
                             esperando incremento */
    unsigned short semzcnt; /* número de processos
                             esperando semval = 0 */
    ...
}
```

- A seguir serão apresentadas as funções para manipular semáforos definidas em `sys/sem.h`

7

Obtendo ID: `semget`

- `semget` obtém um identificador de um semáforo:

```
int semget(key_t key, int nsem, int flag);
```

- Retorna ID se ok, -1 erro
- `key` - identificador inteiro não negativo
- `nsem` - número de semáforos no conjunto (pode ser 0 se `key` identifica conjunto existente)
- `flag` - define permissão de acesso aos semáforos e outras flags como `IPC_CREAT`

8

Controlando: semctl

- semctl dispara uma operação de manipulação de semáforo:

```
int semctl(int semid, int semnum, int cmd,...
           /*união semun com demais args*/ );
```

- semid - identificador de semáforo

semnum - quando for o caso, número do semáforo a ser operado (inicia em 0)

- cmd - comando a ser executado

- Retorno: para comandos GET (com exceção de GETALL) retorna valor correspondente; para os demais comandos 0.

9

Controlando: semctl (cont.)

- O quarto parâmetro (união semun) é opcional e depende do comando solicitado

- Se presente tem o seguinte formato:

```
union semun {
    int          val; /* para SETVAL */
    struct semid_ds *buf; /*para IPC_STAT/IPC_GET*/
    unsigned short *array; /*para GETALL/SETALL*/
}
```

- Quarto argumento, quando passado, é a própria união

- Não um ponteiro para a união

10

Comandos de controle

- cmd é um comando de controle:

IPC_STAT - obtém a estrutura semid_ds para o conjunto, armazenando em arg.buf

IPC_SET - altera permissões associadas ao conjunto

IPC_RMID - remove o conjunto de semáforos do sistema. Remoção é imediata

GETVAL - retorna semval para o membro semnum

SETVAL - altera semval para o membro semnum
Valor é especificado por arg.val

GETPID - retorna sempid do membro semnum

GETNCNT - retorna semncnt do membro semnum

GETZCNT - retorna semzcnt do membro semnum

11

Comandos de controle (cont.)

- cmd é um comando de controle (cont.):

GETALL - retorna os valores de todos os semáforos no conjunto. Os valores são armazenados em arg.array

SETALL - altera os valores dos semáforos no conjunto pelos valores apontados por arg.array

- O comando semctl executa apenas uma operação em semáforos

- Para criar as operações wait and signal, são necessários mais de um comando executados como operação atômica

12

Operando: semop

semop dispara diversas operações de manipulação de semáforo atomicamente:

```
int semop(int semid, struct sembuf semoparray[],
          size_t nops);
```

Retorno: 0 se ok; -1 em erro.

semid - identificador de semáforo

semoparray - ponteiro para um vetor de operações em semáforos, representados por sembuf

nops - número de operações no vetor sembuf

13

struct sembuf

Estrutura sembuf:

```
struct sembuf {
    unsigned short sem_num; /*número do semáforo*/
    short          semop;   /*operação*/
    short          sem_flg; /*IPC_NOWAIT, SEM_UNDO*/
}
```

14

struct sembuf (cont.)

semop opera de acordo com o valor do semáforo:

sem_op > 0, então semval += sem_op;
possivelmente desbloqueando outro processo

sem_op == 0, então processo bloqueia até
que semval == 0, não alternado semval

sem_op < 0, processo é bloqueado até que
semval >= sem_op, quando semáforo é
atualizado com semval -= sem_op

15

struct sembuf (cont.)

sem_flg define algumas flags:

IPC_NOWAIT evita que a operação bloqueie o
semáforo. Caso fosse gerar o bloqueio,
retorna EAGAIN

SEM_UNDO automaticamente desfaz operação no
semáforo quando processo termina. Evita
manter o semáforo trancado quando um
processo morre

16

Simplificando uso de semáforos

🔧 Para simplificar o uso de semáforos o livro Advanced Linux Programming propõe um conjunto de funções

🔧 Funções usam os conceitos apresentados:

binary_semaphore_allocation - aloca semáforo e obtendo o ID

binary_semaphore_deallocate - desaloca semáforo. todos os usuários devem finalizar seu uso.

binary_semaphore_initialize - inicializa um semáforo com o valor 1

binary_semaphore_wait - equivalente a um wait em um semáforo

binary_semaphore_post - equivalente a um signal em um semáforo

17

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

/*Define a união semun*/
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short int *array;
    struct seminfo *__buf;
};

/*Obtém um ID para um semáforo binário, alocando se necessário*/
int binary_semaphore_allocation (key_t key, int sem_flags)
{
    return semget (key, 1, sem_flags);
}

/*Desaloca semáforo binário. Todos os usuários devem ter
finalizado seu uso. Retorna -1 em caso de falha*/
int binary_semaphore_deallocate (int semid)
{
    union semun ignored_argument;
    return semctl (semid, 1, IPC_RMID, ignored_argument);
}
```

18

```
/*Inicializa um semáforo binário com o valor 1*/
int binary_semaphore_initialize (int semid)
{
    union semun argument;
    unsigned short values[1];
    values[0] = 1;
    argument.array = values;
    return semctl (semid, 0, SETALL, argument);
}

/*Wait: espera em um semáforo binário. Bloqueia até que o valor
do semáforo seja positivo, então decrementa-o em 1*/
int binary_semaphore_wait (int semid)
{
    struct sembuf operations[1];
    /*Usa o primeiro (e único) semáforo*/
    operations[0].sem_num = 0;
    /*Decrementa em 1*/
    operations[0].sem_op = -1;
    /*Permite desfazer*/
    operations[0].sem_flg = SEM_UNDO;
    return semop (semid, operations, 1);
}
```

19

Simplificando uso de semáforos (cont.)

```
/*Signal ou Post: incrementa o semáforo binário em 1 e
retorna imediatamente*/
int binary_semaphore_post (int semid)
{
    struct sembuf operations[1];
    /*Usa o primeiro (e único) semáforo*/
    operations[0].sem_num = 0;
    /*Incrementa em 1*/
    operations[0].sem_op = 1;
    /* Permite desfazer*/
    operations[0].sem_flg = SEM_UNDO;

    return semop (semid, operations, 1);
}
```

Código disponibilizado na página da disciplina

20

Queues (Filas de Mensagens)

- Lista encadeada de mensagem armazenadas no kernel e identificadas por um queue ID
- Para criar uma nova fila ou abrir uma fila existente usa-se `msgget`
- Para adicionar novas mensagens no fim da fila usa-se `msgsnd`
- Para se obter uma mensagem da fila usa-se `msgrcv`

21

Estrutura da Mensagem

- Cada mensagem contém:
 - tipo - inteiro longo
 - tamanho
 - bytes de dados
- Mensagens não precisam ser obtidas na ordem FIFO, podem ser buscadas pelo seu tipo

```
struct mensagem
{
    long mtipo;
    char mtexto[MSGTAM];
};
```

22

Estrutura da Fila: `msqid_ds`

- Cada fila possui uma estrutura `msqid_ds` associada a ela:

```
#include <sys/msg.h>
struct msqid_ds {
    struct ipc_perm msg_perm; /*permissões */
    msgqnum_t msg_qnum; /*num. mensagens na fila */
    msglen_t msg_qbytes; /*máx. num. de bytes na fila */
    pid_t msg_lspid; /* pid do último msgsnd */
    pid_t msg_lrpid; /* pid do último msgrcv */
    time_t msg_stime; /* tempo último msgsnd */
    time_t msg_rtime; /* tempo último msgrcv */
    time_t msg_ctime; /* tempo última mudança */
    ...
}
```

- A estrutura define o status atual de cada fila

23

Abrindo e Criando: `msgget`

- `msgget` abre uma fila existente ou cria uma nova:
`int msgget(key_t key, int flag);`
- Retorna queue ID se ok, -1 erro
- `key` - identificador inteiro não negativo
- `flag` - define permissão de acesso inicial e flags de controle de criação (como `IPC_CREAT`).

24

Enviando: msgsnd

- msgsnd coloca dados em uma fila:

```
int msgsnd(int msqid, const void *ptr,
           size_t nbytes, int flag);
```

- Retorna 0 se ok, -1 erro
- msqid - especifica queue id
- ptr - aponta para a mensagem: inteiro longo que define o tipo e os dados da mensagem
- nbytes - tamanho da mensagem
- flag - define modificadores com IPC_NOWAIT (não bloqueante)

25

Recebendo: msgrcv

- msgrcv recebe mensagem de uma fila:

```
int msgrcv(int msqid, void *ptr, size_t nbytes,
           long type, int flag);
```

- Retorna tamanho dos dados se ok, -1 erro
- msqid - especifica queue id
- ptr - aponta para a mensagem: inteiro longo que define o tipo e os dados da mensagem
- nbytes - tamanho da mensagem. Se mensagem for maior que nbytes e existe a flag MSG_NOERROR, mensagem é truncada

26

Recebendo: msgrcv (cont.)

- tipo - especifica que mensagem obter:

tipo == 0 a primeira mensagem da fila é retornada
tipo > 0 a primeira mensagem na fila igual a tipo é retornada
tipo < 0 a primeira mensagem na fila em que tipo é o menor valor menor ou igual ao valor absoluto de tipo

- flag - IPC_NOWAIT, MSG_NOERROR, ...

27

Controlando: msgctl

- msgctl realiza várias operações em filas:

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

- Retorna 0 se ok, -1 erro
- msqid - especifica queue id
- cmd - comando a ser executado:
 - IPC_STAT: obtém o msqid_ds da fila em buf
 - IPC_SET: copia dados de buf na fila
 - IPC_RMID: remove a fila do sistema
- buf - utilizado de acordo com o comando

28

Exemplo: Produtor-Consumidor

```
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MSGTAM 128
#define MSGKEY 1234

typedef struct mensagem
{
    long mtipo;
    char mtexto[MSGTAM];
} mensagem_buf;
```

29

Produtor

```
main ()
{
    int msqid;
    int msgflg = IPC_CREAT | 0666;
    mensagem_buf sbuf;
    memset(&sbuf.mtexto, 0, (size_t)MSGTAM);

    if((msqid = msgget((key_t)MSGKEY, msgflg)) < 0)
    {
        perror("msgget");
        exit(1);
    }
    sbuf.mtipo = 1;
    (void) strcpy(sbuf.mtexto, "Voce recebeu isso?\n\0");
    if(msgsnd(msqid, &sbuf, (size_t)(strlen(sbuf.mtexto)), 0) < 0)
    {
        perror("msgsnd");
        exit(1);
    }
    printf("Mensagem enviada...\n");
    exit(0);
}
```

30

Consumidor

```
main ()
{
    int msqid;
    mensagem_buf rbuf;
    memset(&rbuf.mtexto, 0, (size_t)MSGTAM);

    if((msqid = msgget((key_t)MSGKEY, 0666)) < 0)
    {
        perror("msgget");
        exit(1);
    }
    if (msgrcv(msqid, &rbuf, MSGTAM, 1, 0) < 0)
    {
        perror("msgrcv");
        exit(1);
    }
    printf("%s\n", rbuf.mtexto);
    exit(0);
}
```


31


Exercícios

1. Baixe da página, compile e execute os programas exemplos (veja arquivo leiam.txt no próprio zip).
2. Adicione semáforos ao programa com memória compartilhada.
3. Modifique o programa com memória compartilhada para utilizar filas.
4. Modifique o programa produtor-consumidor para usar memória compartilhada e semáforos. O consumidor deve remover a fila após o uso.

32

Bibliografia

 STEVENS, W.R. Advanced Programming in the UNIX Environment. 2nd. Ed., Addison Wesley, 2005

 Advanced Linux Programming
(<http://www.advancedlinuxprogramming.com/>)