Sistemas Operacionais I

Semáforos

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
    - O valor 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.

Semáforos

- 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
- Tipos de semáforos: binários (mutex) e contadores

Semaforos POSIX

- Consiste de um vetor de elementos semáforos
- Os elementos semáforos são parecidos, mas não iguais as clássicos semáforos apresentados em Sistemas Operacionais (do Dijkstra)
- Cada conjunto de semáforo mantém uma estrutura semid_ds
- Cada elemento semáforo inclui algumas informações: valor, processos esperando e etc…

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

Estrutura do elemento semáforo

- Para cada elemento semáforo tem-se:

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

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

Obtendo um Semáforo: 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

Controlando: semctl

- semctl dispara uma operações de manipulação de semáforo:

int semctl(int semid, int semnum, int cmd,...

/*união semun com demais args*/);

- semid - identificador de semáforo
- semnun - 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.

Controlando: semctl

- O quarto parâmetro (uniõ 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_SET/IPC_GET*/
    unsigned short *array;/*para GETALL/SETALL*/
}
```

- Quarto argumento, quando passado, é a própria união
- Não um ponteiro para a união

Comandos de Controle

- cmd é um comando de controle:
  - IPC_STAT - obtém a estrutua 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

Comandos de Controle

- 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 apondados 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

Operação com Semáforos

- 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

struct sembuf

- Estrutura sembuf:

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

struct sembuf

- 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

struct sembuf

- 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

Simplificando o uso de Semáforos

- Biblioteca estática para manipular semáforos
- Funções e estruturas

int set_semaforo(int valor, int sem_id);
void del_semaforo(int sem_id);
int P(int sem_id);
int V(int sem_id);

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

Exercícios

1. Adicione semáforos ao programa com memória compartilhada apresentado na aula passada
2. Resolver o problema do produtos e consumidor apresentado em sala de aula usando memória compartilhada e semáforos.