Sistemas Operacionais, 1^0 semestre/2017

Experimento #3

1. Introdução

No experimento anterior foi utilizado um mecanismo de IPC (Inter-Process Communication) baseado em fila de mensagens. Usando este mecanismo foi possível compartilhar dados entre diferentes processos. Outro mecanismo de IPC é a memória compartilhada. Quando uma região de memória é compartilhada entre dois processos que podem ter acesso “simultaneamente” para modificação por, pelo menos, um deles, esta região de memória deve ser protegida.

Um mecanismo que pode ser usado para solucionar o problema de acesso simultâneo a uma mesma área de memória, se chama semáforo. Semáforos são mecanismos compartilháveis entre processos que garantem sincronismo, se utilizados de maneira adequada, por exemplo, para que um processo tenha acesso exclusivo a um recurso crítico em uma localização de memória.

Neste experimento, deseja-se explorar o conceito de exclusão mútua e perceber o funcionamento de uma implementação de semáforos.

Este exercício foi definido a partir dos experimentos existentes em http://www.rt.db.erau.edu/experiments/unix-rt que pertencem ao Laboratório Embry-Riddle de tempo-real.

2. Objetivos

A seguir estão os objetivos deste experimento com relação ao aluno:

- Observar e entender por que um recurso crítico deve ser protegido contra acessos “simultâneos”.
- Esclarecer o conceito de exclusão mútua, de maneira a ser capaz de explicá-lo e descrevê-lo.
- Entender como e quando usar semáforos.

3. Tarefas

A seguir são discutidos conceitos diretamente relacionados com o experimento. Sua leitura é importante e será cobrada quando da apresentação. Caso o entendimento desses conceitos não se concretize procure reler e, se necessário, realizar pequenas experiências de programação até que ocorra o entendimento.

Às vezes é necessário manter um recurso crítico de maneira exclusiva para impedir que outros processos tenham acesso a esse recurso “simultaneamente”.

Em um SO multitarefa, a execução de processos de maneira simultânea ou mesmo intercalada pode ocasionar situações imprevisíveis (condições de corrida). Quando o acesso e o uso do recurso dependem de eventos externos (tal como a interação com o usuário, um time-slice ou uma interrupção externa), fica mais difícil para um projetista do SO predizer quando um dos processos tentará ter acesso ao recurso.

Dois ou mais processos que são executados separadamente, se tentam ter acesso para modificação de um recurso "simultaneamente", mesmo que seja só um deles querendo a modificação, necessitam de um método que permita que apenas um dos processos tenha acesso ao recurso, por vez.

A solução teórica para o problema do recurso crítico é simples: antes de entrar na região crítica de código, o recurso é tornado indisponível, através de um mecanismo de exclusão mútua, que também é usado para disponibilizá-lo ao sair da região crítica.

Deixando o recurso indisponível, apenas um processo pode entrar na sua região crítica. Uma vez que o processo acaba de ter sua região crítica executada, o mesmo disponibiliza o recurso, de forma que um outro processo possa ter acesso. Um processo desejando acessar o recurso compartilhado, necessita apenas verificar a disponibilidade do mesmo antes de ter sua região crítica executada.

Exclusão Mútua

Exclusão mútua é o princípio em que só um processo pode ter acesso a sua região crítica em um determinado momento.

Pergunta 1: Uma região por ser crítica tem garantida a exclusão mútua? Justifique.

Pergunta 2: É obrigatório que todos os processos que acessam o recurso crítico tenham uma região crítica igual?

Semáforos

Semáforos são mecanismos que podem resolver o problema da exclusão mútua. Um semáforo pode ser visto como um mecanismo que pode sofrer uma operação que ocasiona aquilo que significaria o trancamento ou aquilo que significaria o destrancamento da execução de instruções (estas operações de trancar e destrancar são comumente chamadas com outros nomes, por exemplo como up e down, ou V e P).

As operações sobre um semáforo são atômicas (ou seja, não são interrompidas até sua completa finalização).

Pergunta 3: Porque as operações sobre semáforos precisam ser atômicas?

Semáforos são implementados no SO e são considerados uma forma de IPC (semáforos também podem ser usados para sincronização, tão bem como para obtenção de exclusão mútua). Da mesma maneira que SOs diferentes implementam versões diferentes de memória compartilhada e filas de mensagens, há várias implementações de semáforos. O POSIX.1b define semáforos com identificadores e sem identificadores. O System V também implementa semáforos e estes são os usados neste experimento.

Semáforos do System V

As seguintes chamadas de sistema são usadas para operar semáforos no System V:

- semget () cria um **conjunto** de semáforos com base em uma determinada chave e retorna o handle do conjunto;
- semop () permite ao processo testar ou operar o conjunto de semáforos (pode ser equivalente a trancar e destrancar);
- semctl () permite executar operações gerais sobre o conjunto de semáforos, tal como remover ou reinicializar.

Semelhante ao mecanismo de fila de mensagens, os semáforos no System V usam uma chave única para identificar um conjunto de semáforos. Isto significa que um conjunto contém um ou mais semáforos, que individualmente são identificados por números sequenciais começando em 0, ou seja, os semáforos de um mesmo conjunto são indexados. Para detalhes sobre a operação e uso de chave associada a mecanismo de IPC, olhar o experimento anterior.

Para cada conjunto deve ser associada uma chave para se obter um descritor (handle), o qual é passado para as operações que podem ser executadas sobre membros escolhidos do conjunto.

Neste experimento, todos os conjuntos de semáforos criados conterão apenas um semáforo e as operações usadas se aplicam a esse semáforo individual. Para mais informação sobre conjuntos de semáforos, leia as páginas correspondentes usando o comando man.

O exemplo de código seguinte mostra como criar um semáforo no System V:

```
#define SEM_KEY 1304
#define NO_OF_SEMS 1
#define SEM_PERMS 0666
int id;
if ((id = semget(SEM_KEY, NO_OF_SEMS, SEM_PERMS | IPC_CREAT)) == -1) {
     perror ("chamada de semget falhou !");
     exit(1);
}
```

No código acima, a chamada semget () recebe três argumentos. SEM_KEY é a chave única para o IPC do System V. NO_OF_SEMS é o número de semáforos no conjunto, ou seja, um (lembrar que esta opção permite associar múltiplos semáforos com um único par key/id). O último argumento corresponde às permissões de acesso ao conjunto e ao flag para criação, se ainda não criado.

semget () devolverá o descritor (id) para o conjunto de semáforos, se terminar bem, e -1 caso contrário.

O exemplo de código seguinte mostra como usar um semáforo no System V para trancar e destrancar o acesso a um recurso:

```
int id;
struct sembuf semb[1];
semb[0].sem_num = 0;
semb[0].sem_op = -1; / * use 1 para destrancar * /
semb[0].sem_flg = 0;
if (semop(id, semb, 1) == -1)
{
     perror ("chamada de semop falhou !");
     exit(1);
}
```

O código acima pode confundir um pouco, portanto um exame detalhado deve ser feito.

Como descrito anteriormente, semop () pode ser usado para trancar e destrancar o acesso a um recurso. A chamada semop () recebe três argumentos, o descritor para o conjunto de semáforos, um vetor de estruturas do tipo sembuf e o número de elementos no vetor que deve ser considerado. Isto permite que a chamada semop () opere sobre vários semáforos em uma única operação atômica, necessariamente todos do mesmo conjunto.

A estrutura sembuf contém três elementos: sem_num, sem_op, e sem_flg. sem_num é o número do semáforo dentro do conjunto sobre o qual será realizada a operação, ou seja, neste experimento sem_num será sempre 0 (que representa o primeiro semáforo no conjunto). sem_flg é uma flag que pode ser usada para executar operações especiais. Neste experimento, será sempre 0. sem_op corresponde à operação a ser executada. Use -1 para “trancar” o semáforo e 1 para “destrancar” (valor que será somado ao semáforo).

NOTA, para mais informação sobre os valores para os elementos destas estruturas, use o comando man. As operações sobre semáforos têm muito mais detalhes que o explicado.

semctl () pode ser usado para remover um conjunto de semáforos. Como a chamada para semctl () é semelhante a uma chamada IPC "ctl", também não será explicada em detalhes aqui, ver experimento anterior.

É importante notar que o valor padrão de inicialização de um semáforo é 0. No contexto acima, 0 pode ser usado para indicar que um recurso está trancado. Então, imediatamente depois de criar o semáforo, **para se obter exclusão mútua**, o valor do semáforo deve ser modificado para significar destrancado. Isto é equivalente ao init visto em sala de aula.

Programa Exemplo

O programa exemplo apresenta como podem ser usados semáforos para proteger recursos compartilhados. Nele, o recurso compartilhado é um inteiro, usado como índice para acessar uma string de caracteres que é exibida na tela e com o qual vários processos tentarão exibir letras do alfabeto e dígitos cooperativamente, de maneira que não ocorra repetição dos caracteres, algumas vezes de maneira desordenada. O programa é projetado para ser compilado de dois modos, um em que é considerado o uso do semáforo para exclusão mútua e um outro sem o uso. Ou seja, há uma diretiva ao

compilador que, se definida, permite a compilação dos comandos que protegem o recurso e, se não definida, não há a inclusão desses comandos no código executável.

Pergunta 4: O que é uma diretiva ao compilador?

Examine como o programa se comporta em cada um dos dois modos.

Um único processo pai inicia vários processos filhos, seguindo o algoritmo:

1. Inicialize as estruturas de semáforo.
2. Crie e inicialize o semáforo (coloque 1 no seu valor).
3. Crie, associe e inicialize o segmento de memória compartilhada.
4. Inicie o número especificado de filhos de forma que cada um chama PrintAlphabet.
5. Espere por alguns milisegundos.
6. Mate os filhos.
7. Destrua o semáforo.
8. Destrua o segmento de memória compartilhada.
9. Termine.

Os filhos usam o seguinte algoritmo:

1. Durma por um período pequeno de tempo para dar a todos os irmãos uma chance de terem começado.
2. Adquira a hora do dia e gere um número pseudo aleatório.
3. (Se PROTECT estava especificado em tempo de compilação) tranque a região crítica (somando -1 ao valor do semáforo).
4. Obtenha o índice atual do segmento de memória compartilhada.
5. Execute um loop um número variável de vezes (pseudo aleatório) e escreva na tela esse número de caracteres, a partir do valor existente no índice no segmento de memória, a cada iteração. Atualize o índice.
6. Durma por microsegundos para dar aos outros processos uma chance para executar.
7. Atualize o índice no segmento de memória compartilhada.
8. Verifique se o índice atual, no segmento de memória, não está além da fronteira do vetor e, se estiver, escreva um caractere de retorno na tela e coloque zero no índice.
9. (Se PROTECT estava especificado em tempo de compilação) destranque a região crítica (somando 1 ao valor do semáforo).
10. Repita o loop

Pergunta 5: Porque o número é pseudo aleatório e não totalmente aleatório?

A string dos caracteres que estarão sendo exibidos pelos processos é:

```
char g_letters_and_numbers [ ] = "abcdefghijklmnopqrstuvwxyz 1234567890
                                  ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

Com PROTECT definido em tempo de compilação, a saída deve parecer com algo do tipo:

```
Filho 1 iniciado
Filho 2 iniciado
Filho 3 iniciado
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 12345
```

Com PROTECT não definido em tempo de compilação, a saída é dramaticamente diferente (NOTA: A barra invertida indica uma mudança de linha que não foi produzida através de um caractere de retorno):

```
Filho 1 iniciado
Filho 2 iniciado
Filho 3 iniciado
abacbdcceddfeegffhggihhjiikjjlkkmllnmmonnpooqpprqqsrrtssuttvuuwvvxwwyxxzyy
zz1 \
2113224335446557668779880 99                                          00A
BACADBBECCFDDGEEHFFIGGJHHKIILJJMKKNLLOMMPNNQO \
ORPPSQQTRRUSSVTTWUUXVVYWWZXX YY ZZ
a b c
da
eabfcbgcdhediefjgfkghlihmijnkjoklpmlqmnronsoptqpuqrvsrwstxutyuvzwv
wx1yx2yz3 z4 \
1521623743845965067                    87A89B09C0                      DA
EABFCBGCDHEDIEFJGFKGHLIHMIJNKJOKLPMLQMNRONSO \
PTQPUQRVSRWSTXUTYUVZWV WX YX
aYZb cd de Z
a
abefcbcdghedefijgfghklihijmnkjklopmlmnqronopstqpqruvsrstwxutuvyzwvwx
1yxyz23 z \
14521236743458965670                   8789AB090                      CDA
ABEFCBCDGHEDEFIJGFGHKLIHIJMNKJKLOPMLMNQRONO \
PSTQPQRUVSRSTWXUTUVYZWVWX YXYZ
```

Em seguida são apresentadas várias seções do código onde se pode perceber o que o programa em execução está fazendo exatamente. As operações sobre semáforos são examinadas em detalhe. O código seguinte cria e inicializa um semáforo e várias estruturas usadas em operações de semáforo:

```
#define SEM_KEY 0x1304
...
struct sembuf g_lock_sembuf[1];
struct sembuf g_unlock_sembuf[1];
...
g_lock_sembuf[0].sem_num = 0;
g_lock_sembuf[0].sem_op = -1;
g_lock_sembuf[0].sem_flg = 0;
g_unlock_sembuf[0].sem_num = 0;
g_unlock_sembuf[0].sem_op = 1;
g_unlock_sembuf[0].sem_flg = 0;
...
if ((g_sem_id = semget (SEM_KEY, 1, IPC_CREAT | 0666)) == -1) {
   fprintf(stderr, "chamada semget () falhou, nao pode criar semaforo !");
   exit(1);
}
if (semop (g_sem_id, g_unlock_sembuf, 1) == -1) {
   fprintf(stderr, "chamada semop () falhou, nao pode inicializar semaforo
" );
   exit(1);
}
...
if (semctl (g_sem_id, 0, IPC_RMID, 0) != 0) {
   fprintf(stderr," nao foi possivel remover o semaforo!\n ");
   exit(1);
}
```

A primeira sequência de código cria dois vetores de estruturas do tipo sembuf, com um elemento (poderiam ser vários).

A seguir estão as definições para as operações que serão executadas no semáforo. Uma é usada para trancar e a outra para destrancar. Estas estruturas são inicializadas no processo pai.

Para ambas as estruturas, sem_num é inicializado com zero declarando que se quer operar no primeiro semáforo do conjunto (único criado no conjunto!) e sem_flg é inicializado com zero declarando que nenhuma "ação especial" é necessária. sem_op é inicializado com -1 para trancamento e 1 para o destrancamento.

Em seguida, o conjunto de semáforos com a chave SEM_KEY é criado usando a chamada semget (). As permissões são fixadas em 0666 e o conjunto é criado se não existe ainda. Como declarado previamente, imediatamente depois da criação, o semáforo deve ser inicializado como destrancado. A chamada semop () faz isto.

A chamada semctl () remove o conjunto de semáforos do sistema.

O código seguinte nos filhos permite o trancamento e destrancamento do recurso:

```
...
#ifdef PROTECT
if (semop (g_sem_id, g_lock_sembuf, 1) == -1) {
```

```
     fprintf(stderr, "A chamada semop () falhou, nao pode trancar o
recurso!");
     exit(1);
}
#endif
...
#ifdef PROTECT
if (semop (g_sem_id, g_unlock_sembuf, 1) == -1) {
     fprintf(stderr, " A chamada semop () falhou, nao pode destrancar o
recurso!");
     exit(1);
}
#endif
...
```

Note o par #ifdef PROTECT #endif ao redor da chamada semop () para permitir ou não compilar a proteção antes e depois da região crítica.

Esta é uma prática comum na indústria para adicionar ou remover código baseado em opções em tempo de compilação.

Note o motivo pelo qual o programa acima pode violar a exclusão mútua, analisando duas linhas de código:

```
...
tmp_index = *g_shm_addr;
...
*g_shm_addr = tmp_index + i;
...
```

A primeira linha estabelece a leitura do índice atual a partir do segmento de memória compartilhada e o armazena em uma variável local. A segunda linha, algum tempo depois, estabelece a escrita do novo índice de volta à localização da memória compartilhada.

Condição de corrida ocorre quando um processo lê o índice da memória compartilhada antes de um processo que o leu anteriormente acabe a sua operação de atualização do novo índice. Quando isto acontece, ambos os processos podem ter o mesmo valor de índice e ambos os processos acabam escrevendo os mesmos caracteres na tela. Olhe o resultado para verificar isto. Um semáforo pode assegurar que dois processos não acessam o recurso crítico “ao mesmo tempo”.

4. Resultado

Cada experimento constitui uma atividade que precisa ser completada através de tarefas básicas. A primeira se refere à compilação e entendimento de um programa exemplo que trata de assuntos cobertos em sala de aula e na teoria. Uma segunda se refere à implementação de uma modificação sobre o exemplo.

Este experimento deve ser acompanhado de um relatório com as seguintes partes obrigatórias:

- Introdução, indicando em não mais do que 20 linhas o que fazem o programa exemplo e os programas modificados;
- Apresentação de erros de sintaxe e/ou lógica que o programa exemplo possa ter, juntamente com a sua solução;
- Respostas às perguntas que se encontram dispersas pelo texto do experimento e pelo código fonte exemplo;
- Resultados da execução do programa exemplo;
- Resultados da execução do programa modificado;
- Análise dos resultados (deve-se explicar o motivo da igualdade ou desigualdade de resultados, usando como suporte gráficos com dados);
- Conclusão indicando o que foi aprendido com o experimento.

Entrega

A entrega do experimento deve ser feita em seu escaninho no AVA, em uma pasta com o nome "Experimento3", de acordo com o cronograma previamente estabelecido, **até a meia-noite** do dia anterior à respectiva apresentação.

Em todos os arquivos entregues deve constar **OBRIGATORIAMENTE** o nome e o RA dos integrantes do grupo.

Devem ser entregues os seguintes itens:

i. *os códigos fonte;*
ii. os executáveis;
iii. o relatório final do trabalho, em formato pdf.

Solicita-se que **NÃO** sejam usados compactadores de arquivos.

Não serão aceitas entregas após a data definida. A não entrega acarreta em nota zero no experimento.

5. Tarefas

A tarefa para este experimento é: primeiro, compilar duas vezes o programa exemplo, sem os erros existentes, uma com proteção, outra sem proteção. Salvar em um único arquivo os resultados de cinco execuções de cada um dos executáveis obtidos nessas duas compilações, ou seja, dez execuções.

Analise os resultados obtidos e explique as eventuais diferenças.

Modifique o programa exemplo de maneira que existam oito filhos, quatro dos quais são iguais aos existentes no exemplo, ou seja, são produtores de caracteres, e quatro serão consumidores.

Um produtor acessa o vetor de caracteres e, ao invés de exibi-los no monitor, os coloca em um buffer em memória compartilhada.

Um consumidor qualquer tem um índice (global) para o buffer compartilhado diferente do índice usado por um produtor (também global). Devem ser dois índices.

Cada vez que se vai produzir ou consumir, sorteia-se quantos caracteres serão envolvidos, de um total de, no máximo, cinco caracteres.

Consumir significa substituir cada caractere produzido pelo caractere “#”.

Ao atingir o final do buffer, tanto um produtor como um consumidor deve exibir o conteúdo de todo o buffer e o respectivo índice deve ser posicionado no início do buffer.

Faça com que cada filho produtor mostre quais caracteres produziu.

Use uma dormência equivalente ao número de caracteres produzidos ou consumidos, respectivamente para produtores e consumidores.

Use o mesmo esquema de proteção e de semáforo usado no programa exemplo. Deverão ser dois conjuntos de semáforos, cada qual para proteger o acesso a cada um dos índices globais.

Compile o programa modificado duas vezes: uma com proteção e outra sem proteção.

Salvar em um único arquivo os resultados de cinco execuções de cada um dos executáveis obtidos nessas duas compilações, ou seja, dez execuções.

Analise os resultados obtidos e explique as diferenças.

6. Apresentação

Os resultados do experimento serão apresentados em sala no dia de aula prática da semana marcada para a entrega, com a presença obrigatória de todos os alunos, de acordo com o cronograma previamente estabelecido.

Serão escolhidos alunos para a apresentação e discussão dos resultados.

Todos os alunos que completaram o experimento devem preparar para a apresentação, em formato digital:

- Os códigos-fonte,
- A introdução e a conclusão,
- Os erros e soluções no programa exemplo,
- As respostas às questões,
- Os resultados e sua análise.

Recomenda-se fortemente a preparação para apresentação.