Fundamentos de Sistemas de Operação

FCT/UNL - 2009/2010

Trabalho prático 3
Sincronização de processos - Semáforos
(2 aulas)

Objectivos

Este trabalho pretende ilustrar aspectos apresentados nas aulas teóricas relacionados com a programação concorrente, concretamente o uso de semáforos para a sincronização de processos e a definição de zonas de memória partilhada entre processos.

Bibliografia:

- slides e apontamentos das aulas teóricas
- capítulo 6 do livro Sistemas Operativos (secções 6.1 a 6.6)
- capítulo 8 do livro Unix System Programming
- manuais *on-line*

Base de trabalho

A base para este trabalho é uma aplicação que cria um conjunto de processos cujo comportamento imprime no terminal 5 mensagens do tipo:

$< pid >$ mensagem i

onde *pid* é o identificador do processo e i o número da mensagem. Por exemplo, um processo com *pid* 100 produzirá o seguinte resultado:

<100> mensagem 1
<100> mensagem 2
<100> mensagem 3
<100> mensagem 4
<100> mensagem 5

Nota: O número de processos a criar é definido pelo utilizador na linha de comando.

Teste da Aplicação

Decarregue o código fonte da aplicação de

`http://asc.di.fct.unl.pt/~herve/aulas/fso/2009-2010/t3/t3.tgz`

compile-o e execute-o várias vezes, analisando a ordem das mensagens produzidas.

Sincronização com Semáforos

Fase 1

Utilize semáforos, com a semântica de Dijkstra dada nas aulas teóricas, para alterar o comportamento do programa dado de forma a que não haja intercalação das mensagens escritas. Ou seja, para garantir que as mensagens escritas por um dado processo aparecem todas de seguida.

Por exemplo, a execução da configuração `trab3 2` tem de produzir um dos seguintes resultados:

Resultado 1	*Resultado 2*
<100> mensagem 1	<101> mensagem 1
<100> mensagem 2	<101> mensagem 2
<100> mensagem 3	<101> mensagem 3
<100> mensagem 4	<101> mensagem 4
<100> mensagem 5	<101> mensagem 5
<101> mensagem 1	<100> mensagem 1
<101> mensagem 2	<100> mensagem 2
<101> mensagem 3	<100> mensagem 3
<101> mensagem 4	<100> mensagem 4
<101> mensagem 5	<100> mensagem 5

Nota: Assuma que os processos criados têm como pid os valores 100 e 101.

Fase 2

Altere o programa dado para que haja uma sincronização dos processos a cada mensagem, ou seja, cada processo só escreve a sua mensagem i quando todos os tiverem escrito as suas mensagens com identificador $i - 1$.

Nesta fase, a configuração `trab3 2` poderá dar origem a vários resultados distintos. Seguem-se dois exemplos:

Resultado 1	*Resultado 2*
<100> mensagem 1	<100> mensagem 1
<101> mensagem 1	<101> mensagem 1
<101> mensagem 2	<100> mensagem 2
<100> mensagem 2	<101> mensagem 2
<100> mensagem 3	<100> mensagem 3
<101> mensagem 3	<101> mensagem 3
<100> mensagem 4	<101> mensagem 4
<101> mensagem 4	<100> mensagem 4
<101> mensagem 5	<101> mensagem 5
<100> mensagem 5	<100> mensagem 5

Sugestão: Implemente uma *barreira*, um método de sincronização em que cada processo só continua a sua execução quando todos tiverem chegado à barreira.

Relatório

O relatório deve apresentar de forma sucinta (no máximo 3 páginas) os problemas a resolver e quais foram as soluções implementadas. Juntamente com este relatório deve entregar o código desenvolvido comentado.

Bibliotecas Auxiliares

Para facilitar a utilização dos semáforos e legibilidade dos programas sugere-se o uso das seguintes bibliotecas dadas.

Manipulação de semáforos

Ficheiro cabeçalho: `sem.h`
Ficheiro de implementação: `sem.c`
Interface:

- **int** createSem(**int** sem_value) - cria um semáforo anónimo com o contador inicializado a sem_value retornando o identificador do semáforo. Em caso de erro o programa termina.
- **void** deleteSem(**int** sem_id) - destrói o semáforo com identificação sem_id. Em caso de erro o programa termina.
- **void** P(**int** sem_id) - faz P (decrementa) sobre o semáforo cujo identificador é sem_id - se o valor do semáforo for zero bloqueia o processo até poder decrementar. Em caso de erro o programa termina.
- **void** V(**int** sem_id) - faz V (incrementa) sobre o semáforo cujo identificador é sem_id. Em caso de erro o programa termina

Áreas de memória partilhada

Ficheiro cabeçalho: `shm.h`
Ficheiro de implementação: `shm.c`
Interface:

- **int** createSharedArea(**int** size) - cria um segmento de memória partilhada de dimensão size retornando o identificador da região de memória partilhada. Em caso de erro o programa termina.
- **void** $*$ attachArea(**int** shm_id) - junta o segmento de memória partilhada com identificação shm_id ao espaço de endereçamento do processo invocador. Devolve um apontador para a zona partilhada. Em caso de erro o programa termina.
- **void** deleteSharedArea(**int** shm_id) - destrói o segmento de memória partilhada com identificação shm_id. Em caso de erro o programa termina.

Nota: Quando é criado um segmento de memória partilhada este continua a existir mesmo quando o processo que o criou deixa de existir. Para evitar o esgotamento das tabelas do sistema os segmentos de memória partilhada devem ser destruídos explicitamente. Isto pode ser conseguido de duas maneiras:

- Antes do programa concorrente terminar: chamando a função deleteShareArea, que usa as operações do sistema de operação para a destruição do respectivo IPC;
- Usando os comandos `ipcs` e `ipcrm` disponíveis a partir do shell. Veja a sintaxe destes comandos no manual do sistema.