TRABALHO DE IMPLEMENTAÇÃO SO – 2025/1

O trabalho consiste em processos/threads concorrentes, que vão disputar o acesso a uma área de memória compartilhada. A cada execução do programa, teremos uma corrida concorrente para a área de memória compartilhada, e uma ordenação diferente no acesso à mesma (Processo 1 foi o primeiro a acessar, Processo 3 foi o segundo a acessar, e assim em diante...). Para esse cenário, a implementação em linguagem C deve estar aderente à descrição dos passos apresentados a seguir:

1. Um processo pai cria uma área de memória compartilhada destinada a armazenar uma variável do tipo inteira que é sempre inicializada com o valor 1;

2. O processo pai cria 4 processos filhos de forma sequencial utilizando o comando fork. Esses filhos devem ter acesso à área de memória compartilhada criada pelo pai;

3. Cada processo filho deve ser iniciado e ficar imediatamente bloqueado (via algum mecanismo de bloqueio), aguardando a liberação de execução via processo pai (vide passo 4);

4. Após a criação dos 4 processos filho (que estão bloqueados em virtude do passo 3), o processo pai dorme 2s e em seguida libera os 4 filhos de forma simultânea (instruções sequenciais de liberação);

5. Cada processo filho dorme um tempo aleatório entre 200ms e 1s utilizando o comando sleep;

6. Após acordar do passo 5, cada processo filho tenta acessar a área de memória compartilhada com exclusão mútua;

7. Quando o processo conseguir acessar a área de memória compartilhada, ele deve copiar o valor atual da memória compartilhada para uma variável local, depois decrementar em 1 unidade o valor atual na memória compartilhada e por fim dormir um tempo aleatório entre 500ms e 1s utilizando o comando sleep. Lembrando que todas essas atividades do passo 7 devem ser realizadas com exclusão mútua entre os processos;

8. Após conseguir acessar a área de memória compartilhada e fazer os passos do item 7, cada processo filho deve instanciar uma thread para imprimir um conteúdo na tela indicando o valor que foi copiado da memória compartilhada e seu PID (Process ID) (Ex: “Processo de PID 4567 copiou o valor -2”). Após a impressão do conteúdo indicado, a thread deve ser finalizada. Atentar que print na tela do passo 8 deve ser realizado obrigatoriamente pela thread instanciada no processo filho correspondente;

9. Ao término do item 8, cada o processo filho imprime um conteúdo na tela (o próprio processo filho deve fazer isso e não mais a thread, já que a mesma foi encerrada no passo 8) informando da sua finalização e seu PID (Process ID) (Ex: “Processo filho de PID 1234 foi finalizado”) e informa ao processo pai por meio de troca de mensagens a sua finalização;

10. Ao término dos 4 processos filho, o processo pai imprime um conteúdo na tela dizendo que todos os filhos foram finalizados, informando seu PID, e o valor remanescente na memória compartilhada (via acesso direto à área de memória compartilhada, sem cópia de valor para

variável local). (Ex: “Processo pai de PID 6778 informa que todos os filhos foram finalizados e o valor atual da memória compartilhada é -3) e é finalizado.

Considerações:

A) Como o passo 7 é realizado em exclusão mútua, cada processo copia um valor diferente da área de memória compartilhada, já que depois da cópia o processo decrementa em 1 unidade o valor da memória compartilhada. Com isso, o 1º processo que conseguir o acesso à área de memória compartilhada copia o valor 1, o 2º copia o valor 0, o 3º copia o valor -1 e o 4º e último copia o valor -2;

B) A dormida de tempo aleatório no passo 5 é para forçar que a cada execução do programa a ordem de acesso dos processos à memória compartilhada seja diferente. Já a dormida no passo 7 é para forçar uma concorrência no acesso à área de memória compartilhada, ou seja, um processo está fazendo acesso à memória compartilhada, e enquanto dorme lá dentro outro processo tenta fazer o mesmo acesso, o que será negado por causa da exclusão mútua. Desta forma, não esqueçam de implementar a alteração da semente de geração de valores aleatórios no código do programa, para que cada execução utilize uma semente diferente. Caso isso não seja feito, os valores de dormida de cada processo serão os mesmos da execução anterior.

C) Os manuais sobre semáforos POSIX (sem_open, sem_close, sem_wait, sem_post, etc), memória compartilhada (shmget) e troca de mensagens (msgsnd e msgrvc) são uma boa fonte de consulta para se iniciar a programação. De qq forma, vcs podem consultar livremente a internet para pesquisar exemplos de uso e esclarecer dúvidas sobre a implementação dessas rotinas. Atentar para a necessidade de desconexão e exclusão da área de memória compartilhada, semáforos e fila de troca de mensagens ao final da execução do programa, para que não haja “lixo” de memória no sistema após as execuções do mesmo;

D) Além dos conteúdos impressos na tela nos passos 8, 9 e 10, vocês podem imprimir conteúdo de controle à vontade, seja para verificar se um processo/thread passou por um trecho de código específico, ou para indicar informações adicionais que acharem interessantes.

Prazo para entrega: Até as 23:59 do dia 03/07.

Artefatos de entrega na tarefa do Teams:

- Arquivos de código fonte (.c ou .h) em formato .zip;

- Vídeo de gravação de tela (ou link de acesso ao vídeo em repositório público, como YouTube, Google Drive, etc.) com a explicação do código fonte e a execução do programa 5 vezes (gravar um único vídeo para a explicação do código fonte e execuções do programa).

Pontuação do trabalho: 0-10 pontos, sendo 8 pontos para a corretude da solução e 2 pontos para o grau de eficiência da solução;

Qq dúvida é só chamar no Teams ou durante as aulas presenciais. Bons estudos!