Trabalho de Implementação – Sistemas Operacionais 2024/1

Elaborar um programa no Linux em linguagem C onde o processo pai cria uma área de memória compartilhada (shmem) e 4 processos filhos (através da system call fork). A área de memória compartilhada indicada acima deve ser suficiente para conter duas variáveis do tipo inteiro. A primeira das variáveis será iniciada com o valor 120 (cento e oitenta) e outra com 0 (zero).

Dois processos filho vão manipular diretamente a área de memória compartilhada, e os outros processos filho criarão duas threads cada um, e essas threads irão manipular a área de memória compartilhada. A manipulação da área de memória compartilhada compreende os passos indicados abaixo, e será realizada em um loop de 30 iterações (30 para o processo filho1, 30 para o processo filho 2, 30 para a thread A/B do processo filho 3, 30 para a thread A/B do processo filho 4) com a seguintes rotinas:

1. Copia o valor da 1a variável da área de memória compartilhada para uma variável local;

2. Decrementa de forma local o valor da variável copiada;

3. "Dorme" (sleep) um tempo aleatório (variando entre 200ms e 2s);

4. Armazena o valor decrementado na 1a variável da memória compartilhada;

5. Incrementa o valor da 2a variável (observe que esta 2a variável não é copiada para a memória local do processo filho/thread, e sim alterada diretamente na memória compartilhada);

6. Exibe na tela uma mensagem contendo o seu PID (no caso dos processos filhos que manipulam a memória compartilhada) ou TID (no caso das Threads), um número de série sequencial iniciado em 1 (cada processo/thread terá uma contagem própria, os valores atualizados das duas variáveis da memória compartilhada e outras informações que o programador achar relevante;

7. "Dorme" (sleep) um tempo aleatório (variando entre 200ms e 2s).

Para cada um dos filhos 3 e 4, que criam 2 threads (A e B), deve haver um sorteio aleatório em cada uma das 30 iterações do loop para decidir qual thread vai acessar a área de memória compartilhada e executar os passos 1 a 7 indicados acima. Ou seja, em cada iteração do loop, apenas 1 thread que ganhar o sorteio realizará esses passos, e a que perder o sorteio deverá ficar bloqueada, e ser desbloqueada apenas quando ganhar um sorteio, e vice-versa. Desta forma, os filhos 3 e 4 mesmo com 2 threads cada, realizarão os mesmos 30 acessos à área de memória compartilhada que os filhos 1 e 2. A diferença é que os filhos 1 e 2 acessarão a memória compartilhada diretamente, e os filhos 3 e 4 acessarão a memória compartilhada via suas respectivas threads vinculadas A e B, onde a quantidade de acesso de cada thread vinculada é definida aleatoriamente por meio do sorteio mencionado. Por exemplo, em uma execução do programa a thread A do filho 3 fez 23 acessos, e a thread B desse filho 3 fez 7 acessos. Em outra execução a thread A fez 12 acessos, e a thread B fez 18 acessos, e assim sucessivamente.

Quando o loop terminar, cada filho deve avisar ao processo "pai" que já terminou. O pai então informa que está ciente de que o filho 1, 2, 3 ou 4 terminou (imprimir na tela uma mensagem informativa quando cada um desses 4 eventos ocorrer). No caso dos processos criados através de fork () vamos utilizar a troca de mensagens como mecanismo de comunicação entre processos filhos com o pai. Ou seja, os processos filho vão enviar uma mensagem para o processo pai (msgsnd), onde o processo pai já deve estar aguardando de forma bloqueante as mensagens dos processos filho (msgrcv). Observe que os processos filhos que criam suas threads A e B devem ficar aguardando elas terminarem, para que então o respectivo filho possa enviar uma mensagem ao processo pai. Desta forma, o processo pai tem que ficar bloqueado aguardando o recebimento de uma mensagem de cada filho. Quando todos os processos filhos terminarem o processo pai imprime duas últimas mensagens, informando que o programa será finalizado e exibindo os valores atuais das duas variáveis da memória compartilhada. Após essas 2 últimas mensagens, o processo pai é finalizado.

Observe que a memória compartilhada deve ser manipulada de forma consistente, ou seja, é preciso garantir exclusão mútua no acesso à mesma por meio de semáforos POSIX. Desta forma, a exclusão mútua deve ser iniciada imediatamente antes do passo 1 do loop, e ser finalizada imediatamente após passo 6. Observar que o passo 7 (uma nova dormida entre 200ms e 2s) é feito fora da região crítica, portanto sem proteção de semáforos para exclusão mútua. Para que haja um escalonamento aleatório do processo e threads que manipulam a memória compartilhada, o tempo de "dormir" desses elementos (passos 3 e 7 do loop) deve ser efetivamente aleatório (variando entre 200ms e 2s). Caso contrário eles podem executar em sequência.

Os manuais disponíveis no Linux sobre semáforos POSIX (sem_open, sem_init, sem_wait, sem_post, sem_close, sem_unlink, 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 dos processos e threads programa, para que não haja "lixo" de memória no sistema após a finalização do programa, prejudicando assim novas execuções do programa caso o computador não seja reiniciado ou desligado.

Prazo para entrega: Até as 23:59 do dia 23/05.

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

Nosso retorno a aulas presenciais será no dia 29/05 (semana seguinte ao prazo de entrega), ou após o término da greve. Caso a greve seja finalizada antes de 23/05 (prazo de entrega), não teremos aulas presenciais de SO para que a carga horária da disciplina nessas 4 semanas possa ser dedicada exclusivamente à realização deste trabalho.

Artefatos de entrega na tarefa do Teams: - Arquivos de código fonte (.c ou .h) em formato .zip; - Link de um vídeo de gravação de tela com a explicação do código fonte e a execução do programa mostrando o funcionamento do mesmo.

Qq dúvida é só chamar no Teams.

Bons estudos!