Projeto de Sistemas Operacionais - Comunicação entre Processos

Lucas Monteiro do Amaral[1], Francisco Matheus Fernandes Freitas[1], Alberto Luian De Lima Marques[1]

[1]Universidade Estadual do Ceará (UECE) - Fortaleza - CE - Brasil

{lucas.amaral, fernandes.matheus, alberto.luian}@aluno.uece.br

Resumo. *O intuito desse relatório é explicar o processo de desenvolvimento de uma solução que tinha como proposta estabelecer uma comunicação entre processos por meio de uma memória compartilhada, com o foco em três cenários específicos (todos utilizando uma Shared Memory (SHM) como meio de comunicação): uma troca de mensagens entre dois processos; uma troca de arquivos entre dois processos e a comunicação de um processo enviando mensagens para vários outros processos, onde o texto será expirado após ser lido por todos os N processos que irão recebê-lo.*

1. Introdução

Um dos principais desafios da comunicação entre processos é a sincronização, especialmente pela iminência de uma condição de corrida, que trata-se da tentativa de acesso a um recurso compartilhado entre vários processos, de uma só vez, o que pode gerar conflito de sobrescrita de dados, prejudicando sua manipulação nos campos que são utilizados.

Para evitar tal problema, faz-se necessário a presença de um mediador responsável por controlar o acesso a SHM, que no caso, se trata do Mutex. Trata-se de um conjunto de sinais que coordena o acesso a regiões críticas (como a SHM, por exemplo).

Outro desafio enfrentado foi a forma com que iríamos coordenar a leitura da SHM sem que um processo o fizesse mais de uma vez ou então realizasse a leitura de sua própria mensagem.

Com base no que foi dito, é notória a importância de uma implementação efetiva de um Mutex para que a comunicação seja estabelecida com sucesso, e esse artigo tem como objetivo relatar o processo que nós enfrentamos para desenvolver tal solução.

2. Estratégias

Esta seção tem por finalidade descrever a metodologia e as ferramentas que nós utilizamos para o desenvolvimento da solução (Bibliotecas, linguagens utilizadas, SO utilizado, dentre outras informações).

2.1. Cenário

O sistema operacional escolhido foi o Windows, majoritariamente pela familiaridade dos integrantes do grupo, e pela maior compatibilidade com as bibliotecas utilizadas.

Um dos motivadores que nos levou a optar pela utilização desse SO, é o seu suporte a Mapped Names, que são simplificações no mapeamento das memórias compartilhadas no ambiente Windows. Essa funcionalidade não se faz presente no Linux, pois ele apenas suporta a criação de SHM's a partir de arquivos temporários, que por sua vez adicionava uma complexidade maior no objetivo de suportar os dois ambientes. Por conta disso, optamos por utilizar apenas o ambiente Windows.

2.2. Ferramentas

As linguagens de programação escolhidas foram C# e Python, e o motivo de termos optado por duas linguagens foi para executarmos a integração entre elas, além da facilidade oferecida pelo Python na execução de processos distintos, o que se provou uma barreira ao tentar executar o mesmo no C#.

As bibliotecas utilizadas (além das padrões de cada linguagem), foram a "subprocess" (usada para realizar chamadas de sistema que criam processos), a "time.sleep" (usada para dar um pequeno intervalo de tempo ao gerar os processos) e a "PySimpleGUI" (usada para criar a interface gráfica de abertura dos processos), todas do Python. Já no C#, utilizamos a "MappedMemoryFiles" (usada para atuar como a shared memory, mapeando um bloco de memória que será compartilhada entre os dois processos).

3. Código

Nesse trecho iremos explicar a estruturação e fluxo de execução do código.

3.1 Estruturação

3.1.1 P2P

Para o P2P, fizemos uso de 2 Threads em cada um dos processos, onde uma thread ficou responsável por verificar constantemente a existência de mensagens ou arquivos na SHM, já a outra, pela espera de uma entrada do usuário e, subsequentemente, a escrita desta mensagem ou arquivo na memória compartilhada.

Para as mensagens, utilizamos a leitura e escrita através de um buffer do tipo “MemoryMappedView”, que coloca a mensagem na primeira linha e o id do processo que enviou a mensagem na segunda linha da SHM(a separação de linhas é definida pela aparição de um ‘\n’), a fim de que a thread de leitura possa verificar se a mensagem que está sendo lida é uma mensagem do seu próprio processo ou se ela deve ser exibida no terminal. Quando a mensagem é exibida, a thread de leitura apaga a mensagem, substituindo os bytes por null, limpando a SHM. Vale ressaltar que todo acesso à memória compartilhada possui uma tentativa de acesso a um mutex, para garantir a exclusividade do acesso à SHM.

Para os arquivos, continuamos com a lógica da escrita ou leitura do id do processo que enviou o arquivo na SHM, substituímos a utilização da mensagem pelo nome do arquivo(a fim de obter um nome para criar o novo arquivo e sua extensão) e adicionamos uma nova linha, que contém uma flag indicando que existe um arquivo a ser lido em uma nova SHM(SHM2), que possui o tamanho do arquivo. A thread de leitura dos arquivos irá ler essa nova SHM2 e usar a função de escrita binária do C# para criar um novo arquivo com o nome e extensão definidos na SHM, após isso ele irá limpar o conteúdo da SHM e da SHM2 de modo semelhante ao feito com as mensagens.

3.1.2 P-N

Para o P-N, temos 2 tipos de processos genéricos, o processo ‘sender’ e o processo ‘receiver’, onde o processo ‘receiver’ é aberto ‘n’ vezes e o ‘sender’, 1 vez.

O processo ‘sender’ cria a SHM e espera uma entrada, para depois escrever uma linha vazia (stringControl), seu id e a mensagem, cada um em sua respectiva linha(bytes separados por um ‘\n’). Após escrever, ele aguarda uma nova mensagem.

Já os processos ‘receiver’, quando são executados, acessam a SHM e verificam a primeira linha, a stringControl, guardando em uma variável qual será seu identificador, de acordo com o tamanho da stringControl, adicionando um “0” na string control e reescrevendo-a, o id do processo ‘sender’ e a mensagem. Após isso ele verifica se existe “1” na stringControl no caractere específico do seu identificador, caso positivo, não executa nada, caso contrário, ele exibe a mensagem e substitui este “0” por “1”, verificando se a stringControl inteira é formada por apenas “1”, se sim, ele escreve novamente a stringControl vazia, repete o id do processo ‘sender’ e apaga a mensagem, indicando no console que a mensagem foi expirada. Em caso de erro de index, ao tentar acessar uma posição inválida da stringControl, que acontecerá quando a mensagem foi apagada e um processo ‘receiver’ tentar verificar se existe “1” no index correspondente ao seu identificador, o seu identificador é resetado e todo o processo de obtenção de um novo identificador é iniciado.

3.2 Fluxo de execução

Existe um menu feito em uma GUI, onde é possível escolher entre P2P e P-N, no caso de P-N é pedido o número “n” de processos.

3.2.1 P2P

No P2P, são abertos 2 terminais, onde é pedida a mensagem ou o número ‘3’, que sinaliza a intenção de envio de arquivo. Quando a mensagem é enviada, ela chega juntamente com o id do processo que enviou, do modo “processo {id}: {mensagem}”. Já no caso de arquivos, ao inserir o sinalizador ‘3’, é pedido o caminho do arquivo e logo após, o processo que irá recebê-lo mostra no console: “arquivo recebido: {nomeDoArquivo.Extensao}”.

3.2.2 P-N

No P-N, são abertos ‘n+1’ terminais, onde um deles espera a mensagem a ser enviada e os outros ‘n’ mostram a mensagem lida, e o último processo a ler exibe um aviso de que a mensagem expirou.

4. Conclusão

Deste modo, podemos perceber melhor a forma com que podemos trabalhar com a comunicação entre processos, desenvolvendo melhor o entendimento nas soluções que envolveram a sincronização entre os processos que, a princípio, não possuíam uma relação. Também passamos a entender melhor como os arquivos são trabalhados e armazenados a nível de bytes.

5. Referências

Tanenbaum, Andrew S.; Bos, Herbert (2016). Sistemas Operacionais Modernos. Pearson Education do Brasil Ltda.

Microsoft Learn - MemoryMappedFiles (https://learn.microsoft.com/pt-br/dotnet/api/system.io.memorymappedfiles.memorymappedfile?view=net-7.0)