Trabalho Prático – Sistemas Operativos

Curso - Engenharia Informática

Unidade Curricular: Introdução à Inteligência Artificial

Filipe Paradela Silveira – 2021139291

André Maltez Nunes -2023145292

Ano Letivo- 2024/2025

Comunicação entre programas

Para a realização deste trabalho, utilizamos um único struct genérico chamado *packet* para transferência de dados entre feeds e o manager.

Este *packet* é separado em 2 componentes: um header, que contém dados para auxiliar na leitura (pid – o pid do programa que enviou o packet original, tipo-msg – um valor que indica como é que a mensagem deve ser lida ou o que é que o recetor deve fazer; e tam_msg – indica quantos bytes do *buffer* devem ser enviados) e um *buffer*, um array de bytes que contém informações extra que não podem ser enviadas apenas com o *header*, como texto ou outros valores adicionais.

O tam_msg foi definido como um unsigned int de 16bits (uint16_t encontrado em "stdint.h") para garantir que tem sempre o mesmo tamanho, independentemente do sistema em que o programa for compilado. Este tipo de dados foi escolhido porque é o tamanho mínimo necessário para enviar os maiores packets, dado que um int de 8bits não seria suficiente para enviar as maiores mensagens com 300 caracteres + outros dados necessários no mesmo packet.

Para tomar partido da capacidade de endereçamento do tam_msg, o tamanho máximo do buffer foi definido como 64KB. Este pode parecer um tamanho exagerado, mas medidas foram tomadas para garantir que apenas as informações necessárias são enviadas entre processos, pelo que, sob circunstâncias normais, um packet deste tamanho nunca será enviado.

Para garantir que apenas enviamos os bytes necessários entre programas, tiramos partido do tam_msg, que nos indica o tamanho do buffer. Assim quando fazemos o write() de um packet, indicamos no tamanho da mensagem sizeof(packetHeader) + packet.head.tam_msg. Assim, para um packet com, por exemplo, 25 bytes no buffer, apenas enviamos 8 bytes (sizeof(packetHeader)) + 25 bytes do buffer = 33 bytes.

Para garantir que uma leitura correta de um packet é feita, fazemos 2 read()'s consecutivos. O primeiro lê apenas um packetHeader. O segundo têm em consideração o tam_msg indicado no header e lê apenas esse número de bytes. Assim, obtemos todos os dados do packet enviado sem qualquer descoordenação de um named pipe.

Named Pipes

Neste trabalho, existem no máximo 11 named pipes em simultâneo e, no mínimo, 1 named pipe.

Estes são:

1 named pipe para o manager;

1 named pipe para cada feed.

Selects e Threads

Neste trabalho utilizamos selects no feed e threads no manager.

Ao todo temos 3 threads no manager:

- Thread do administrador
    - Esta thread permite a um administrador interagir com o manager através de uma interface semelhante ao feed. Esta thread não trata de nenhum dos dados do manager, em vez disso, todos os pedidos realizados pelo administrador são enviados sobre a forma de packets para a thread do servidor (explicada a seguir) da mesma maneira que os feeds interagem com o manager. No entanto, esta thread nunca recebe resposta da thread do servidor, dado que ambas as threads encontram-se no mesmo processo, evitando assim a abertura de um named pipe desnecessário.
- Thread do servidor
    - Esta thread controla todos os dados do manager. Esta thread processa os packets recebidos pelo manager, tanto vindos de feed como do administrador. Quando um packet é processado, novos packets são enviados para os feeds que forem necessários.
- Thread do contador
    - Esta thread é responsável por fazer a contagem decrescente das mensagens persistentes. No entanto, esta thread não tem acesso a quaisquer dados do manager. Em vez disso, a cada segundo, um packet é enviado à thread do servidor a pedir que o tempo de vida das mensagens persistentes seja decrementado. Assim garantimos que todos os acessos e alterações aos dados do manager são realizados num único sítio: a thread do servidor.

Sinais

Para garantir a saída correta de ambos os programas, foi alterado o funcionamento do sinal de interrupção (<Ctrl-c>), passando este a fazer o mesmo que os respetivos comandos de término de cada programa, ou seja:

- No feed é enviado um pedido de saída ao manager. Quando este pedido é garantido pelo manager, o feed termina ordeiramente;
- No manager, são enviados packets a indicar o fechamento do manager a todos os feeds ligados antes deste terminar ordeiramente.