Relatório

Memória Partilhada

Usando os comandos específicos da “shared memory” como o “shmget” e o “shmat” criámos, no main, uma memória partilhada de uma estrutura de dados contendo informações sobre estatísticas de sensores, dados de alertas, variáveis de contagem e um “end” (inicializado a 0) para quando for acionado o SIGINT, este passe para 1 e deixe as threads acabarem em segurança. Alocámos memória dinâmica e inicializámos todas as variáveis e estruturas na memória partilhada, para saber quando estas estavam vazias.

Threads

Nas threads “Console_Reader” e “Sensor_Reader” é lida uma string que vem dos programas “user_console” e “sensor”, respetivamente, através de um named pipe criado corretamente no main antes de serem criadas as threads. Logo após, esta string que contém informações tanto sobre o sensor como os pedidos da consola (informação dividida pelo caracter “#”) é colocada num vetor de strings (“INTERNAL_QUEUE”) com o tamanho máximo referido no ficheiro de configuração.

A última thread criada é o “dispatcher” que pecorrendo este vetor analisa a origem e dá prioridade aos pedidos que vêm da consola e por consequente envia-os para os processos “worker” através de unnamed pipes. Para saber quais os processos que estão disponíveis temos um vetor de números inteiros na memória partilhada (“worker_state”), para que possa ser acessada por todos os processos e caso o valor do índice do processo seja 0, este está disponível e recebe o pedido.

Processos

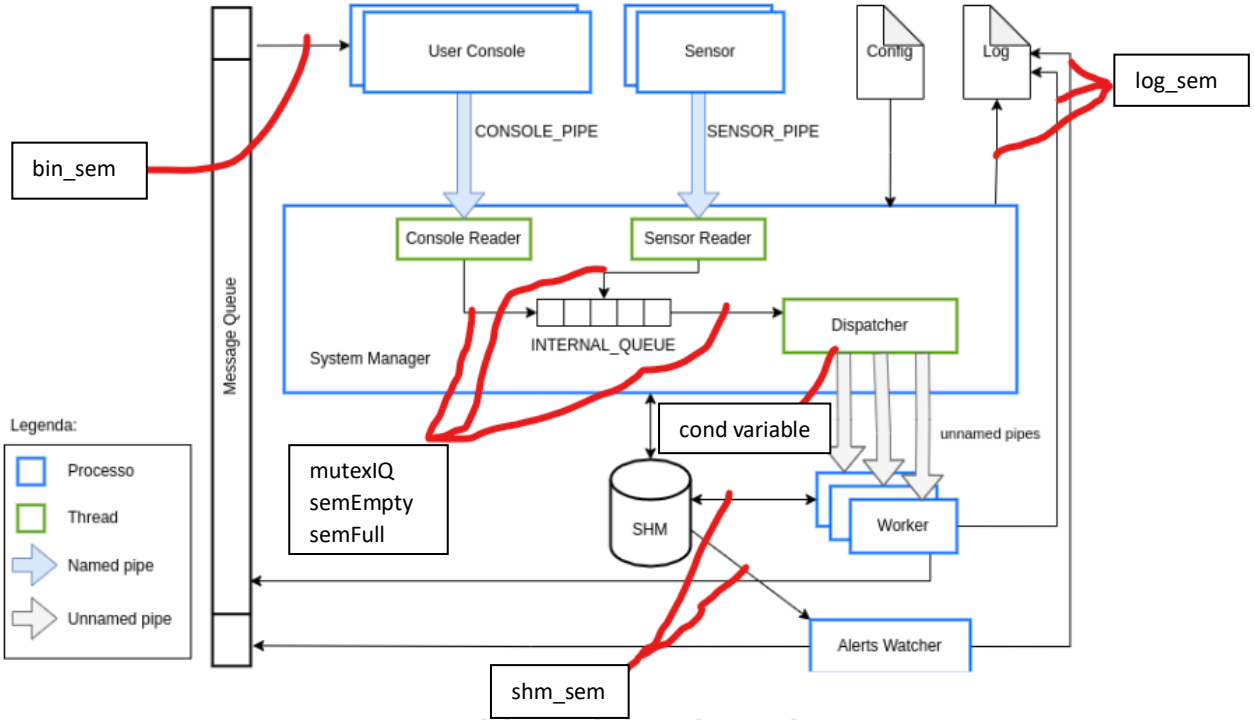
Usando o comando “fork()” criámos o número de processos que constavam no ficheiro de configuração, cada um executando a função “worker()” e mais um para exercer a função “alerts_watcher()”.

A função “worker()” recebe da thread “dispatcher” a “string” e depois divide-a para entender o que tem de fazer e caso seja proveniente da consola faz o que é pedido e devolve o respetivo retorno para a “user_console” através de uma “message queue”. Esta foi criada no main usando a mesma chave que na consola para se poderem conectar. Para fazer acontecer esta conexão usamos “msgsnd” para enviar e “msgrcv” para receber. Caso seja do sensor apenas acessa a memória partilhada para inserir novas informações ou atualizar as mesmas.

Término do programa

Temos uma função chamada “clean_resources()”, onde são esperadas e terminadas as “threads”, a memória partilhada é apagada de maneira correta, são fechados todos os pipes (“named” e “unnamed”) e onde são esperados os processos terminarem.

Diagrama atualizado



Para todos os processos escreverem no ficheiro log de forma sincronizada, usámos um named semáforo.
Para não haver corrupção de dados na shared memory usamos também um named semáforo. Desta forma os processos acedem e fazem alterações na shared memory de forma sincronizada.
Usámos um mutex para que a internal queue seja acedida apenas por uma thread de cada vez. Também usámos os semáforos semEmpty, inicializado com o tamanho da IQ, e semFull inicializado a 0. Desta forma o espaço disponível na IQ vai sendo guardado nos semáforos sendo que decrementamos o semEmpty no console reader e sensor reader e incrementamos o semFull, já no dispatcher decrementamos o semFull e incrementamos o semEmpty.
Usámos uma variável de condição no dispatcher para o caso em que todos os worker estão ocupados.
Por fim, na user console usámos um named semáforo para sincronizar os reads da message queue, visto que criámos um processo que está sempre à espera de alertas vindos do alerts_watcher.

Realizado por:

Pedro Brites Nº2021226319 – 60 horas
Marcelo Gomes Nº2021222994 – 60 horas