1 2 9 0

UNIVERSIDADE D
COIMBRA

Relatório do Projeto de Sistemas Operativos

Licenciatura em Engenharia Informática [2022-2023]

Autor: Johnny Alexis Lopes Fernandes 2021190668

Justificação das escolhas de sincronização:

Durante a elaboração do projeto, foi imprescindível considerar os sistemas de sincronização utilizados para o acesso tanto das *threads* como dos processos às zonas de memória partilhadas que estavam a ser acedidas de forma rotineira. Nesse sentido, há três elementos fundamentais precisaram ser levados em conta:

- **Acesso à memória partilhada principal (processos)**

Na memória de acesso principal, onde foram armazenados todos os dados relativos aos sensores, bem como os alertas, foi implementado um *mutex* em conjunto com uma variável de condição, destinados ao aviso do processo responsável pela verificação dos parâmetros de alerta.

Assim, para que haja acesso de qualquer um dos processos *worker* (ou *alerts watcher*) à memória partilhada, é necessário primeiro garantir o *lock* desse mesmo *mutex*. Após o processamento dos dados do sensor, é também feita uma verificação pela lista de alertas para verificar se as chaves a que esses valores se referem têm alertas configurados. Se tal for o caso, o próprio processo *worker* avisa - através de sinalização da variável de condição - que há um novo conjunto de dados a ser verificado, nomeadamente a verificação se os valores guardados ultrapassam os *thresholds* configurados no alerta onde, se assim for o caso, enviará uma mensagem pela *message queue* à consola responsável. Assim, não só garantimos a integridade dos dados através da sincronização de acessos como prevenimos uma espera ativa por parte do *alerts watcher* na pesquisa dos dados na memória partilhada (conforme as boas práticas e também conforme o enunciado).

- **Acesso à memória partilhada secundária (processos)**

Foi criada uma memória partilhada secundária, a fim de verificar quais os *workers* que estão disponíveis para alocar novas tarefas. Deste modo, quer o *dispatcher* quer os *workers* têm acesso a este segmento de memória partilhada, que inclui também um *mutex* e uma variável de condição. Esses dois elementos são usados numa função de *enqueue* e *dequeue*, sendo que a função *enqueue* é usada pelos *workers* e a função *dequeue* é usada pelo *dispatcher*. Um *worker* que tenha executado uma tarefa irá, então, inserir o seu próprio ID num *array* circular, para que o *dispatcher* possa posteriormente dar *dequeue* do elemento que está na frente do *array*. Caso não exista nenhum elemento, o *dispatcher* irá ficar à espera de um sinal de forma indeterminada, até que um *worker* insira o seu próprio ID e sinalize a variável de condição. Mais uma vez e em

comparação, o *mutex* serve apenas o propósito de sincronização do acesso à zona de memória partilhada - em particular, o *enqueueing* e *dequeueing* dos elementos do *array*.

- **Acesso à *Internal Queue* (*threads*)**

Foram também usadas duas variáveis de condição e um *mutex*. Em comparação com o dito anteriormente, novamente o *mutex* foi usado para o acesso ao *array* com os dados que são passados entre o *console_thread* e o *sensor_thread*, ao *dispatcher*. Por outra via, neste caso em particular, existem duas variáveis de condição que são usadas ao longo do código, havendo também uma função de *enqueueing* e *dequeueing*. No entanto, convém ressalvar que neste caso tem uma variável para os casos em que a fila está cheia, situação na qual as *threads* que lêem os dados dos *named pipes* (*sensor_reader* e *console_reader*) descartam qualquer tipo de informação até poderem colocar novas informações na *internal queue*. Pela mesma lógica, assim que a *internal queue* estiver vazia, o *dispatcher* aguarda uma sinalização no momento da colocação de nova informação para que possa prosseguir.

Outras considerações finais e conclusão:

Optou-se pela criação de uma *shared memory* adicional para a sincronização e libertação dos processos filhos (*workers*) para novas tarefas. Foi tido em conta que é necessária uma nova página (processo de paginação) para a criação desta *shared memory*, no entanto, considerou-se mais elegante como forma de não iterar todos os *workers* à procurar de algum disponível e sim por ter um *array* circular que dá uma tarefa a todos os *workers* e divide através de uma lógica de *load balancing*, percorrendo todos processos de igual forma.

A não utilização de semáforos no presente projeto deve-se ao facto de não se ter considerado um ganho significativo em detrimento do modelo escolhido. Mais se justifica com o facto de que, por exemplo, na memória partilhada que diz respeito à sincronização dos processos *worker*, o único ganho seria a maior simplicidade na contabilização de processos disponíveis em fila para a utilização. Contudo, continuaria a ser necessário o *array* da memória para indicar quais os processos disponíveis, motivo pelo qual ter um semáforo ou uma variável de condição acabariam por apresentar o mesmo resultado e eficiência prática.

O esquema finalizado em anexo indica todo o processo e lógica inerente à estrutura criada em código e poderá ser consultado.

O trabalho foi desenvolvido na totalidade pelo autor supracitado, pelo que o número de horas aplicado ao projeto, apesar de não ter sido contabilizado, ultrapassa em larga escala as 150 horas de desenvolvimento.

Creio que o desenvolvimento e estrutura escolhida adapta-se ao projeto em questão, tendo o programa sido testado nas mais diversas situações e panoramas.