Relatório do projeto de SO
2019/2020

Com o objetivo de otimizar o funcionamento do programa desenvolvido, decidimos recorrer à utilização de listas ligadas para guardar as informações necessárias ao desempenho das funções de cada processo, e para representar as filas de espera cruciais para o funcionamento deste programa. A razão desta escolha deve-se ao facto de ser possível ordenar a informação nas listas, de forma a que, pesquisas, inserções e eliminações de dados em memória tenha o menor overhead possível. Mantendo-se apenas na memória os dados necessários em cada momento. Isto é, após um conjunto de dados deixar de ser relevante para qualquer processo do programa, é imediatamente removido.

Para auxiliar nesta gestão de processamento de dados, usamos diversos IPC's dados nas aulas: semáforos POSIX named para controlar acessos a zonas de memoria partilhadas por processos; semáforos MUTEX juntamente com variáveis de condição, para evitar esperas ativas; um pipe para estabelecer cominação entre o programa principal e outro auxiliar; uma estrutura de MSQ para que possa haver comunicação entre processos secundários, dentro do processo principal; criação de zonas de memória partilhada entre processos pai-filho.

Funcionamento do programa

Logo após ser iniciado, a primeira coisa que o programa faz é executar a função main. Começa por obter o id do gestor, limpa o ficheiro "log.txt" e chama a função "inicializa" cujo objetivo passa por alocar memória, criar e mapear a memória partilhada, inicializar semáforos, variáveis de condição e criar a message queue. De seguida mostra ao user e escreve no log o "PID" do gestor de simulação e cria a thread do tempo. O gestor (main) depois de inicializar os recursos, tem apenas a função de receber continuamente comandos do pipe onde sempre que recebe um comando primeiro faz a sua validação, validando a sintaxe e os valores, se for válido procede então à inserção de um novo nó na sua lista com todas as informações do comando.

A função PIPE-comandos.c após o envio dos comandos existentes no ficheiro do qual são lidos, permite que sejam escritos e posteriormente enviados novos comandos. Se detetar que o file descriptor está cheio, apaga o conteúdo do ficheiro donde carrega os comandos para evitar sobrecarregar o pipe.

Para criar voos o gestor cria a thread gere_tempo que está encarrega de incrementar a variável do tempo, e de comparar os inits dos nós da lista do gestor com o tempo atual, criando uma thread para um voo sempre que um init é igual ao tempo.

Cada voo, assim que é criado acede ao seu nó da lista do gestor, pois recebe como parâmetro um ponteiro para o seu próprio nó dessa mesma lista, guarda a informação lá contida em variáveis locais, eliminando depois o nó da lista, De seguida enviam um pedido à torre de controlo pela message queue, aguardando depois a resposta da mesma. Nessa resposta já vem o valor do seu slot na shm, sendo que vão então aceder ao seu slot para verificar se já tem manobra a executar, caso não tenham ficam bloqueados na variável de condição à espera de ser notificados pela thread gere_voos (mencionada adiante). Uma vez notificado um voo vai proceder a execução da sua manobra notificando no fim a thread gere_voos, que já terminou a manobra, de seguida termina.

Passando agora para o funcionamento da torre de controlo. A torre começa por apresentar o seu id, cria ponteiros para inserir nós na sua lista, para a sua criação e para a ordenação de prioridades.
De seguida cria duas threads, a gere_emergencias e a gere_voos.

O objetivo da primeira passa por receber as mensagens de emergência e inserir o voo na lista da torre, e na fila de espera de ARRIVALS, com prioridade máxima (alterando o valor da variável emergencia da estrutura TORRE_struct). Começa por entrar num ciclo while(1) onde verifica duas condições, se o número total máximo de voos já foi atingido, e nesse caso entra noutro ciclo while donde só sai quando houver slots livres na shm, ou seja quando algum voo terminar a sua execução, rejeitando entretanto, todos os voos que enviam pedidos.

Para atribuir o valor do slot a um voo, bloqueia o acesso ao array slots_livres através dum semáforo (mutex_slots_livres), verificando se existe algum slot reutilizável se existir é lhe atribuído o mesmo, senão é lhe atribuído um novo valor de slot dado por um contador. Depois o nó é introduzido sempre sobre o controlo doutro semáforo (mutex_torre), uma vez que há três processos, (torre, thread gere_emergencias, thread gere_voos), a inserir/remover nós das listas das filas e da torre.

A torre de controlo tem um funcionamento semelhante ao da thread gere_emergencias, mas apenas recebe os pedidos normais de voos de partida ou chegada, não atribuído prioridades a nenhum

deles, inserindo nós na sua própria lista, mas também nas listas ligadas das duas filas de espera. O processo de atribuição do slot é igual ao da thread anterior.

Se receber um pedido de um voo de chegada, o takeoff é definido como zero, guarda o ETA e o combustível de acordo com os valores recebidos pela MSQ, e insere na fila de ARRIVALS passando um ponteiro para o nó da lista da torre, correspondente ao voo a ser processado, para que seja possível aceder à informação da lista da torre, a partir da fila de espera de ARRIVALS. Se receber um pedido de um voo de partida, guarda o takeoff de acordo com o valor da MSQ, e atribui o valor 0 ao ETA e o combustível inserindo na fila de DEPARTURES e na sua própria lista um novo nó.

A gestão das listas da torre, da fila de ARRIVALS e da fila de DEPARTURES, é feita pela thread gere_voos. Esta está num ciclo while onde em cada iteração verifica se o tempo foi incrementado acedendo ao valor da variável tempo que se encontra na primeira posição do array de estruturas da memória partilhada (ptr_shm[0].tempo), se detetar que foi incrementado então percorre a lista ligada da torre para fazer a atualização dos seus dados, nomeadamente combustível de voos de chegada. Para além disso verifica se no tempo atual existe algum voo com combustível igual a 0, nesse caso redireciona-o para outro aeroporto, reordena a lista ligada da fila de ARRIVALS, e escreve no ficheiro log.txt. De seguida ainda no processo de percorrer a lista ligada da torre, verifica se há algum voo que no tempo atual tenha ficado em situação de emergência, caso isto se verifique é lhe atribuído prioridade máxima, através da variável emergência, para que possa aterrar.

Depois de percorrer a lista ligada da torre, esta thread chama a função “escalonamento”, que percorre em primeiro lugar a fila de DEPARTURES, verificando sempre apenas para o primeiro nó dessa fila, se o seu valor de takeoff é igual ao valor do tempo atual, só vê o primeiro nó porque a fila está ordenada pelos valores de takeoff, logo se o primeiro for diferente do tempo todos os outros também vão ser, e neste caso atribui aos voos de DEPARTURE prioridade mínima (0). Por outro lado se a condição se verificar, atribui aos voos de DEPARTURE prioridade máxima (3), depois percorre também apenas o primeiro nó da fila de ARRIVALS verificando em primeiro lugar se o voo está em situação de emergência, condição que é verificada com recurso a um ponteiro do nó da lista da fila de ARRIVALS, para o respetivo nó desse voo, da lista da torre, verificando o valor da variável (emergencia) , se estiver a (1), a função atribui prioridade máxima (3) aos voos de arrival, caso contrário atribui prioridade média (1).

De seguida a função compara as prioridades, se a prioridade dos DEPARTURES for maior que a dos ARRIVALS, a função devolve o valor (28), para que a thread gere_voos saiba que as pistas foram cedidas a descolagens, caso contrário devolve (1) sendo as pistas cedidas para aterragens. Se as prioridades forem iguais, então é porque há um voo que precisa de descolar (prioridade 3), mas também há um voo que chegou e que está em estado de emergência (prioridade 3), sendo que nesta situação a função devolve (1) para dar prioridade ao voo que quer aterrar.

Ainda na thread gere_voos depois de receber o valor da função que faz o escalonamento entra na condição que percorre a fila de ARRIVALS, se o valor devolvido for (1), caso esse valor seja (28) entra na condição que percorre a fila de DEPARTURE. Então no caso da fila de DEPARTURES percorre apenas o primeiro nó, notificando o voo através da variável de condição que esta no slot de memória partilhada do respetivo voo, e fica depois à espera que o voo termine e a notifique através da mesma variável de condição. Após receber a notificação do voo, a thread gere_voos vai então proceder à libertação dos nós da lista da torre e da fila de DEPARTURES correspondentes a este voo, guardando também no array slots_livres o valor do slot em shm usado por este voo, para que a torre saiba que o pode reutilizar.

Caso a thread gere_voos entre na condição que percorre a fila de ARRIVALS, vai percorrer a fila toda notificando sempre apenas o primeiro nó da fila para que aterre, usando o mesmo mecanismo que anteriormente, mas desta vez continua a percorrer a fila porque se detetar que há mais do que 5 voos, ordena todos os voos que estão para lá da quinta posição a fazer um hold, esse hold é calculado gerando um valor aleatório entre o min_hold e o max_hold definidos na configurações de entrada, prosseguido depois à reordenação da fila de ARRIVALS pois acrescenta esse tempo de hold ao ETA dos voos que o fizeram.

Quando o ^C é pressionado o gestor redireciona esse sinal para a função cleanup onde espera que todos os voos terminem, enviando um sinal à thread tempo para que esta termine e pare de criar threads.

Por sua vez, a torre também redireciona este sinal para a mesma função, chama a função que escreve as estatísticas no ecrã e no log da aplicação, e envia um sinal para as duas threads que criou terminando a sua execução.

Tempo despendido em horas: Eurico José (entre 110h – 120h) Nuno Duarte (12h – 18h).