João Batanete 2009113460
Beatriz Gonçalves 2012142477

Relatório do projeto de SO

João Batanete 2009113460
Beatriz Gonçalves 2012142477

O projeto destina-se a implementar um servidor HTTP simples que permite aos utilizadores correrem scripts/executáveis guardados em disco e visualizarem o seu output ou a visualizarem o conteúdo de ficheiros html estáticos. As estatisticas de funcionamento do servidor podem ser consultadas num ficheiro chamado “logs.txt” e as configurações usadas neste podem ser alteradas a qualquer altura.

Os requests do cliente são armazenados em estruturas que contém o nome do script/página pretendida, o tipo de request(dinâmico ou estático), a socket do cliente que o fez, a hora de criação e de fim de processamento deste e os parâmetros a passar a este na sua execução no caso dos scripts(no caso de se pretender passar parâmetros, estes devem ser indicados um a um, no browser, separados por virgulas a seguir ao nome do script).

Para lidar com os requests de cada cliente foram usadas duas filas(uma guarda os requests dinâmicos e a outra os estáticos) com recurso a listas ligadas implementadas por nós. Estas estão implementadas para retornar sempre o request mais antigo existente. Consoante a política que estamos a aplicar, a thread de escalonamento ir'a dar prioridade a uma das filas, apenas processando os requests da outra quando esta estiver vazia. Caso estejamos perante a política FIFO, ela vê qual dos requests que estão no topo de cada fila é mais antigo e processa esse em primeiro lugar.

Para processar cada request sao usadas threads que recebem ponteiros para as estruturas dos requests através de pipes e, consoante o tipo destes, os processam(ou recusam) da maneira adequada e enviam(através de uma message queue) uma estrutura que contem dados relativos ao request(hora de receção, hora de fim de processamento,etc).

A “pool de threads” usada por nós é na verdade uma estrutura que contem, para além do array de threads, um array de ints com a mesma dimensao do de threads e que indica se a thread na posição i está livre para processar um request naquele momento. Também é usado um semáforo para impedir que aguarda que uma thread fique livre caso todas estejam ocupadas(a thread de escalonamento tranca esse semáforo cada vez que atribui um request a uma thread, e essa thread destranca-o quando termina, sendo o semáforo inicializado com valor igual ao número de threads no início do programa). As threads recebem como argumento um ponteiro para uma estrutura que lhes dá informações sobre o seu indice, o seu pipe e a possibilidade de dizerem à thread de escalonamento quando estão ocupadas. Quando recebem no pipe delas um ponteiro para NULL em lugar de um ponteiro para um request terminam a execução(isto é usado para terminar as threads quando temos de mudar as configurações do servidor).

Também foram usados semáforos para o processo principal poder esperar que o processo de configurações e de estatísticas iniciassem a execução antes de começarmos a receber requests e para impedir acessos simultâneos à memória partilhada, bem como o envio de mensagens pela message queue em simultâneo por duas threads, visto que as message queues usadas não têm mecanismos de sincronização com threads. Todos os semáforos usados são semáforos POSIX, visto que todos têm de ser usados entre threads(impedindo o uso de semáforos System V) e/ou têm de ser usados de forma não binária(exemplo: o semáforo que controla a pool de threads e impede o acesso a esta quando todas as threads estão ocupadas).

João Batanete 2009113460
Beatriz Gonçalves 2012142477

Para além das situações indicadas no enunciado(uso de sinais SIGHUP no processo de estatísticas e de configurações) também foram usados sinais para informar o processo principal quando as configurações mudam, de forma a este poder atualizá-las. Todos os processos libertam a memória alocada quando recebem um SIGINT da consola(premindo CONTROL+C).