Rosa Manuela Rodrigues de Faria (nº 2005128014)
Vasco Bruno dos Santos Gouveia (nº 2001106666)

Relatório do Projecto da cadeira de Sistemas Operativos

Descrição da abordagem do problema

Como princípio foi criado o processo pai que representa o servidor. Este processo tem como objectivo criar os processos filhos e gerir as mensagens de e para os pipes, bem como o envio de sinais para o cliente.

Este começa por criar os unnamed pipes (2 por cada processo filho) e os 3 processos (dentro do processo pai) que representam os 3 tipos de rotação. É depois criado o named pipe que irá servir de comunicação do cliente para o servidor. As mensagens enviadas serão compostas por 3 campos: um inteiro com a rotação pretendida, um pid_t com o pid do cliente, e um char[] com o path para o ficheiro a tratar. Esta mensagem é depois reenviada através de um unnamed pipe para o filho correspondente à rotação, e quando for recebida do filho poderá ter flags no campo rotação (descritas mais adiante). Todos os pipes são geridos por um select que detectará actividade nos mesmos.

Em cada um dos processos filhos foram depois criadas 4 threads trabalhadoras, sendo que se considerou que não era necessário criar uma master thread, dado que o processo iria estar idle durante a execução do programa; assim, a master thread é o próprio processo.

Cada worker thread terá o propósito de ir ao buffer buscar uma mensagem de um cliente, fazer a rotação pedida por este, e mandar uma mensagem de confirmação ao servidor pelo unnamed pipe que liga o processo filho ao pai. Caso não seja possível executar o trabalho, envia uma mensagem de erro para o servidor com a flag -1 no campo do valor da rotação, também pelo unnamed pipe. No caso da escrita do pipe falhar manda directamente o sinal ao cliente.

Já a master thread (o processo filho) começa por fazer a inicialização do buffer (inicialização da fila e dos semáforos, criando então depois as threads. É também feito o tratamento de sinais (descrito adiante). Seguidamente, e enquanto não receber o sinal de término, fica à escuta de mensagens do servidor. Quando chega uma mensagem coloca-a no buffer, e faz o sem_post do semáforo sem_full.

O semáforo sem_full representa o número de elementos escritos. Não foi necessário um semáforo que indicasse se o buffer tinha espaço livre por este ser uma lista ligada, havendo sempre espaço para mais (salvo se já não houver de facto espaço em memória, caso em que o malloc devolverá um erro; nesse caso o processo terminará). Quando a worker thread vai buscar um trabalho ao buffer verificará primeiro se o semáforo sem_full não está a 0, através do comando sem_wait. Quando o valor for diferente de 0 fará o lock de um mutex, fazendo o unlock do mesmo depois de retirar o processo da lista. Este mutex servirá para fazer a sincronização entre as worker threads, de modo a que não tentem aceder ao mesmo trabalho.

Quanto ao tratamento de sinais, estes foram definidos através de 4 máscaras: unblock_ctrlc, que apenas permite que seja recebido o sinal SIGINT, unblock_sigusr1, que apenas permite a recepção do sinal SIGUSR1, unblock_sigusr2, com a mesma função mas relativa ao sinal SIGUSR2, e block_all, que bloqueará todos os sinais.
A máscara unblock_ctrlc será usada no processo pai (servidor) permanentemente. Após a recepção desse sinal, e caso o utilizador indique que quer terminar o processo, este envia o sinal SIGUSR2 a

Rosa Manuela Rodrigues de Faria (nº 2005128014)
Vasco Bruno dos Santos Gouveia (nº 2001106666)

todos os seus processos filhos e passa a bloquear todos os sinais.
Já a máscara unblock_sigusr2 será usada pelos processos filhos (mas não pelas threads nestes criadas). Ao receber este sinal o processo envia o sinal SIGUSR1 a cada uma das threads através do comando pthread_kill, fazendo depois o cleanup. Neste cleanup será primeiro feito o join das threads, o encerramento dos semáforos, e a limpeza do buffer. Esta limpeza consiste em percorrer todos os elementos mandando o sinal de SIGUSR2 aos clientes em espera, cujos trabalhos já não serão executados, e respectiva libertação de memória. Finalmente é enviada uma mensagem ao servidor através do unnamed pipe com a flag -2 no campo da rotação, para informar o processo pai de que este processo terminou correctamente.
No caso da máscara unblock_sigusr1, esta será usada pelas worker threads, que receberão o sinal correspondente quando a master thread/processo filho do servidor estiver a terminar, como descrito anteriormente.
Durante a execução de trabalhos pela worker thread e a colocação de novas mensagens no buffer pela master thread é utilizada a máscara block_all, para que estas tarefas não sejam interrompidas. Assim, garante-se que o trabalho que está a ser executado pela worker_thread é terminado, bem como que todo o processo de alocação e colocação no buffer de uma mensagem seja também terminado.

No processo cliente será feito um tratamento de sinais idêntico ao que é feito no servidor, sendo bloqueados todos os sinais excepto o SIGUSR1 e SIGUSR2 (que indicarão o sucesso ou insucesso da rotação, respectivamente). É feito o mapeamento do ficheiro e a abertura do named pipe, sendo depois enviada uma mensagem com a estrutura anteriormente descrita (rotação pretendida, pid do próprio e path do ficheiro – parâmetros dados pelo utilizador quando executa o ficheiro) ao servidor através do named pipe. O cliente fica depois idle até receber um dos sinais SIGUSR do servidor.

Para criar os executáveis é apenas necessário executar o makefile. Para correr o processo servidor basta fazer “./server”. Para correr o processo cliente deve-se usar o comando “./client <path> <rotação>”. Os ficheiros necessários para a compilação são o header.h (que contém as estruturas e protótipos das funções do servidor), rotation.c (que contém as funções de rotação fornecidas pelo professor), server.c e client.c (descritos acima).