Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

Application Server

Client Side

Sistemas Operativos

2º ano, 2º semestre

Turma 6, Grupo 1

Sérgio da Gama up201906690

Rui Moreira up201906355

José Silva up201904775

Índice

1.Tratamento de argumentos

Neste projeto, como o formato do *input* to utilizador não varia, concluiu-se que bastaria uma simples verificação direta do *array* de argumentos (char*argv[]). Inicialmente, é verificado o número de argumentos, sendo que este tem de ser obrigatoriamente igual a 4. Caso contrário, o programa termina e expõe a forma correta da sua utilização, sendo esta a introdução dos argumentos no formato: "./c <-t nsecs> <fifoname>". Se o formato anteriormente referido não for respeitado, o programa também termina e mostra igualmente a forma correta de utilização.

2.*Multithreading*

Quando os argumentos são introduzidos corretamente, o programa passa para a parte seguinte, onde são consecutivamente criadas *threads* até o tempo de execução do cliente terminar. A criação das mesmas é realizada com um intervalo aleatório de alguns milissegundos, que é conseguido através da chamada à função *nanosleep()*. Cada uma das threads executa então a mesma função, utilizada para enviar pedidos e receber as respostas do servidor.

3.Comunicação servidor – cliente

Cada uma das *threads* está então responsável por fazer os pedidos ao servidor. Sendo assim, em cada uma das *threads* é aberto o canal público de comunicação (fifo público), por onde é enviado o pedido ao servidor. Caso o servidor feche o canal público, as *threads* ficam em espera ativa até alcançado o tempo de *timeout* predefinido. Se alcançado o *timeout* de espera, todas as *threads* são encerradas e não são criadas mais nenhumas.
Para receber a resposta do servidor, a *thread* cria, antes de enviar o pedido ao servidor, um fifo privado com o nome "pid.tid", sendo "pid" o id do processo e "tid" o id do thread. Os dados do pedido e da resposta do servidor são enviados numa *struct*, utilizando as funções *write()* e *read()*.

4.Evitamento de *deadlocks*/*busy-waiting*

Para evitar qualquer tipo de *deadlocks* foram utilizados os *mutexes*. Após a criação das *threads*, sempre que uma *thread* execute a função *send_request_and_wait_response()* o *lock* do *mutex* é ativado, impedindo assim que qualquer outra *thread* a execute ao mesmo tempo, impossibilitando assim ocorrência de *racing conditions*. O *mutex* é desbloqueado pela *thread* que o bloqueou, apenas quando é recebida a resposta enviada pelo servidor pelo fifo privado, e só aí é que outra *thread* pode executar essa mesma função.

5.Saída do Programa

1. Caso todos os pedidos sejam feitos e sejam obtidas todas as respostas, as *threads* terminam através da função *pthread_exit()* e a estrutura do *mutex* é destruída.
2. Quando o tempo de espera pela resposta do servidor, imposto pelo cliente, é ultrapassado, a *thread* que está à espera de resposta desiste e é eliminada.
3. Quando ocorre algum erro ao longo da execução do programa, é retornado o valor de ERROR (-1), logo após da exibição do motivo do erro, na saída padrão de erros (*stderr*).
4. Se o servidor não criar o fifo público, ou o tiver fechado, durante o tempo de execução do cliente, este desiste e consequentemente todas as *threads* são eliminadas, fazendo uma chamada à função *close_client_inner_thread()* e o programa encerra.

6.Registos

As operações do cliente (IWANT, GOTRS, CLOSD, GAVUP) são registadas na *stdout* através da nossa função *reg()*, que utiliza a função *standard fprintf()*. O formato utilizado é o explicitado no enunciado: “inst ; i ; t ; pid ; tid ; res ; oper”.

Os erros de execução são registados para o *stderr*, sendo as mensagens de, exclusivamente, *debug* controladas com uma variável global booleana. Na versão final, estas estão desligadas, dado que sairiam pela saída padrão (*stdout*) e não são estritamente necessárias.

Auto-avaliação

Aluno	Nota (0-20)	Esforço (%)
Sérgio da Gama	20	33.3
Rui Moreira	20	33.3
José Silva	20	33.3