Departamento de Engenharia Informática e de Sistemas

Instituto Superior de Engenharia de Coimbra

Instituto Politécnico de Coimbra

Licenciatura em Engenharia Informática

Ramo de Desenvolvimento Aplicações

Unidade Curricular de Sistemas Operativos

Ano Lectivo de 2020/2021

Trabalho Prático - Programação em C para UNIX

CHAMPION

Relatório Técnico

José Paulo Ferreira Martins nº2016013630

Pedro Manuel Contente Batalha nº2017014903

Coimbra, 8 de Fevereiro de 2021

José Martins & Pedro Batalha

CHAMPION

Aprovação à Unidade Curricular de Sistemas Operativos

Coimbra, 8 de Fevereiro de 2021

Índice

Resumo

Introdução

O nosso trabalho permite realizar um pequeno torneio com variados jogos disponíveis.
Para participar basta abrir a aplicação do cliente e introduzir o nome, quando houver pelo menos 2 jogadores será iniciado um contador(o tempo é definido pelo árbitro quando abre o servidor ou é lhe atribuído um valor por defeito se não lhe for dado nenhum), no fim desse contador acabar será iniciado o torneio que terá um tempo limitado(também definido pelo árbitro quando abre o servidor e também lhe é atribuído um valor por defeito na falta do mesmo).
Quando o torneio acaba o utilizador vai receber a sua pontuação e quem é o vencedor do torneio e a sua pontuação.
O árbitro pode abrir o servidor e definir o tempo de espera para começar o torneio e a duração do torneio, durante o torneio o árbitro tem alguns comandos disponíveis que pode utilizar(explicado em mais detalhe mais à frente), se o árbitro fechar o servidor todos os jogos acabam e as aplicações dos clientes são fechadas.

Chaves

Ao longo do relatório estes termos são utilizados para referenciar as seguintes coisas:

- Árbitro - pessoa que vai interagir com a aplicação do servidor
- Servidor - aplicação resultante da compilação do arbitro.c
- Utilizador - pessoa que vai interagir com a aplicação do cliente
- Cliente - aplicação resultante da compilação do cliente.c

Cliente

Como funciona

Quando o utilizador abre a aplicação do cliente será-lhe pedido para introduzir o nome que vai ter durante o torneio e depois fica à espera que o torneio começa.
No fim do utilizador introduzir o nome o cliente vai aceder ao namedpipe do servidor e vai enviar uma struct cli com o nome do utilizador ao servidor que a vai receber e depois vai ver se o nome é válido ou se é preciso fazer alterações, por fim fica a espera que o torneio comece.
Quando o torneio começa o cliente vai receber do servidor a informação do jogo pelo seu próprio namedpipe que vai receber todas as informações vindas do jogo e do árbitro e vai utilizar outro namedpipe para enviar informação para o árbitro que vai enviar a informação para o jogo ou se detectar que é um comando vai interpretá lo.

Comandos

- “#mygame” - permite ao utilizador rever as regras do jogo que está a jogar durante o torneio
- “#quit” - permite ao utilizador desistir do torneio

Select

```
FD_ZERO(&fds);
FD_SET(0, &fds);    //TECLADO
FD_SET(c_fifo_fd, &fds);    //FIFO

res_fds = select(c_fifo_fd+1, &fds, NULL, NULL, NULL);

if(res_fds > 0 && FD_ISSET(0, &fds)){ //vai ler do teclado
else if(res_fds > 0 && FD_ISSET(c_fifo_fd, &fds)){ //vai ler do fifo
```

O cliente tem um select para interpretar a localização da entrada de dados e saber o que fazer com eles, assim ele sabe que se os dados vieram do namedpipe das perguntas vai ser informação do servidor como as perguntas do jogo/mensagens do árbitro , se for do teclado vai ser a resposta para o jogo/comandos para o servidor.

Sinais

- SIGUSR1 - quando o cliente recebe este sinal significa que ele foi expulso do torneio e que deve encerrar as ligações com o servidor(fechar e eliminar os namedpipe) e encerrar a sua execução, para isso vai utilizar a função “Expulso()”

```
void Expulsao(){
        char temp[20],temp2[20];

            sprintf(temp, FIFO_CLI, getpid());
            sprintf(temp2, FIFO_JOGO, getpid());
            printf("VOCE FOI EXPLUSO\n");
            unlink(temp);
            unlink(temp2);

    exit(0);
}
```

- SIGUSR2 - quando o cliente recebe este sinal significa que o servidor vai encerrar e que deve encerrar as ligações com o servidor(fechar e eliminar os namedpipe) e encerrar a sua execução, para isso vai utilizar a função “terminarExecucao()”

```
void terminarExecucao(){
    char temp[20],temp2[20];

        sprintf(temp, FIFO_CLI, getpid());
        sprintf(temp2, FIFO_JOGO, getpid());
        int aux = open(temp2,O_WRONLY);
        write(aux,"a\n",2);
        printf("Terminou Execuçao\n");
        unlink(temp);
        unlink(temp2);

    exit(0);
}
```

Servidor

Como funciona

Quando o árbitro abre o servidor ele pode definir o tempo de espera até ao torneio no fim de estarem dois jogadores e o tempo do torneio dando esses valores pela linha de argumento quando lança o servidor. Ao abrir o servidor ele vai ver se já existe algum aberto(ele verifica se já existe algum namedpipe do servidor criado) e se existir ele nao abre, depois ele fica à espera que os clientes se conectem, ele vai recebê los pelo seu namedpipe e depois responde pelo namedpipe do cliente, quando ele deteta que já tem pelo menos dois clientes ele vai começar uma thread com o temporizador que irá dar início ao torneio.
Quando o torneio começar o árbitro vai criar uma thread para cada utilizador, vai ser com essas threads que ele vai gerir a informação que ele recebeu dos jogadores e onde vai executar os jogos, o árbitro executa o jogo e depois manda pelo namedpipe das perguntas do utilizador e vai ler a resposta pelo namedpipe das respostas do utilizador e vai interpretar para ver se é uma resposta ou um comando para ele executar.

A árbitro também pode executar alguns comando na thread principal que vão afetar o torneio

Comandos

- “start” - o start permite começar o jogo sem ser necessário esperar pelo tempo de início de partida.
- “players” - mostra todos os utilizadores ligados ao servidor
- “games” - mostra todos os jogos disponíveis para o torneio
- “k” - este comando em conjunto com o nome de um utilizador vai fazer com que ele seja expulso e vai terminar o jogo dele se algum torneio estiver a decorrer
  O árbitro vai terminar a thread que está a correr o jogo dele e depois manda um sinal ao cliente dele para terminar
- “s” - este comando em conjunto com o nome de um utilizador vai fazer com que ele seja suspenso e se ele estiver a participar em algum torneio parará de jogar o seu jogo até o árbitro dar ordem contrária
  O utilizador tem um boolean associado a ele para a thread saber se tem de interpretar o que recebe dele ou se ele está suspenso e ignora os comando e manda uma mensagens a dizer que ele está suspenso
- “r” - este comando em conjunto com o nome de um utilizador vai fazer com que ele pare de estar suspenso e se tiver algum torneio a participar em algum torneio poderá retomar o jogo em que estava
- “end” - serve para terminar o torneio mais cedo
- “exit” - serve para terminar o servidor e todos os clientes a ele ligado incluindo os jogos se houver algum a decorrer

Select

```
FD_ZERO(&fds);
FD_SET(0, &fds);    //TECLADO
FD_SET(s_fifo_fd, &fds);    //FIFO
res_fds = select(s_fifo_fd+1, &fds, NULL, NULL, NULL);

if(res_fds > 0 && FD_ISSET(0, &fds)){ //vai ler do teclado
else if(res_fds>0 && FD_ISSET(s_fifo_fd, &fds)){ //vai ler do fifo
```

O servidor tem um select para interpretar a localização da entrada de dados e saber o que fazer com eles, assim ele sabe que se os dados vieram do namedpipe do servidor vai ser um utilizador a dar entrada no torneio, se for do teclado vai ser um comando do árbitro.

Threads

O servidor começa só com uma thread e depois vai lançar uma para cada utilizador quando o torneio começa, essas threads só são terminadas quando o torneio acaba, o jogador desiste ou é expulso, também são utilizadas threads para fazer ambos os temporizadores tanto o de espera para o iniciar do torneio como o de contar o tempo até o torneio acabar que vão receber uma estrutura Timer com as informações que precisam.

Mutex

```
pthread_mutex_t mutexTimeToStart;
pthread_mutex_t mutexStartGame;
```

O servidor utiliza mutex para bloquear os contadores de espera do torneio começar e o do tempo do torneio.
As mutex tem como propósito evitar erros de leitura, isto é, enquanto uma thread se encontra a escrever numa variável é importante que neste momento não exista outra thread a ler esses mesmos dados.

A mutexTimeToStart serve para o servidor poder ler o tempo que falta para o início da partida, esse tempo é uma variável que é alterada noutra thread.

A mutexStartGame tem como utilidade evitar os mesmos problemas em cima mencionados, no entanto esta mutex encontra a sua utilidade a servir de protecção das flags das threads que realizam a contagem do tempo.

Comunicações

Cliente-Jogo

A comunicação entre o cliente e o jogo é feita de forma indireta sendo as mensagens primeiro interpretadas pelo servidor. Ao iniciar o servidor vai interpreta as mensagem diretamente na thread principal sendo que a primeira que ele recebe de cada cliente é uma struct do tipo cli com o nome inserido pelo utilizador e que vai ser verificado pelo servidor para ver se é válido e depois envia de volta para o cliente que fica a espera do torneio começar.
Quando o torneio começa o servidor vai criar uma thread para cada utilizador que vai ficar a tomar conta das ligações entre o cliente e o seu jogo.Vai utilizar dois namedpipes um para enviar informação para o cliente, que corresponde ao pipe do cliente, e outro para receber informação(pipe do jogo), quando o servidor recebe informação do cliente ele vai ver se a string começa por “#” para saber se é um comando para ele interpretar ou uma resposta para o jogo.

Árbitro-Jogo

O jogo é executado dentro de um fork feita na thread no servidor que lida com a comunicação entre o cliente e o jogo, para a thread comunicar com o jogo é utilizado um unnamed pipes do lado do filho, a entrada(STDIN) e saída(STDOUT) de dados são redirecionados para os unnamed pipes para no lado do pai vai ler e escrever nesses unnamed pipes o que recebe do cliente e manda para o cliente o que recebe do jogo.

Estruturas

cli,pcli

```
typedef struct {
    char nome[NOME_MAX];
    char nomeJogo[50];
    int pontos;
    pid_t pid;
    int aceite;
    bool suspenso;
    bool ingame;
}cli, pCli;
```

Esta estrutura serve para enviar a informação do utilizador entre o servidor e o cliente. Guarda o nome, o nome do jogo que o utilizador vai jogar no torneio, o pid do cliente, o "aceite" vai guardar se ele foi aceite e vai mudando conforme o estado dele, vai guardar se ele está suspenso e se ele está em jogo.

Timer

```
typedef struct {
    cli *clientes;
    int time;
    int tempoDeJogo;
    bool flagContinue;
    bool *startGames;
    pthread_t *threads_cli;
    pthread_t *thread_tempo_jogo;
    pthread_mutex_t *mutexTimeToStart;
    pthread_mutex_t *mutexStartGame;
    pthread_mutex_t *mutexAddingPlayer;
} Timer;
```

Esta estrutura é utilizada para enviar informação para as threads dos contadores no servidor, tem um ponteiro para os clientes para poder contar o número de utilizadores para saber se pode começar a contagem para o torneio começar(valor do time).

resposta_c

```
typedef struct {
    pid_t pid;
    char mensagem[MSG_MAX];
    bool onServer;
} resposta_c;
```

Esta estrutura é utilizada pelo cliente para enviar a resposta do utilizador para o jogo, o onServer serve para sinalizar se a estrutura está no cliente ou no servidor.

pergunta_s

```
typedef struct {
    char mensagem[MSG_MAX];
    int estado;
} pergunta_s;
```

Esta estrutura é utilizada pelo servidor para enviar a pergunta do jogo para o cliente, o estado serve para atualizar o estado do cliente.

Jogos

O servidor vai buscar o nome do diretório onde estão guardados os jogos a uma variável ambiente criada previamente ao lançamento do servidor, se está não existir ele vai utilizar um nome por defeito já definido.

G_001

Neste jogo é pedido ao utilizador para introduzir as capitais dos países demonstrados, ganha um ponto por cada resposta certa e perde um por cada errada.

G_002

Neste jogo é pedido ao utilizador para acertar num número aleatório entre 0 e N em que N começa igual a 2 e vai duplicando cada vez que o utilizador acerta e ganha um ponto, quando perde um ponto por errar a resposta o N volta a ser igual a 2.

G_003

Neste jogo é pedido ao utilizador para fazer uma pequena conta (o número é entre 10 e 10+15N)que vai aumentando a dificuldade conforme o utilizador vai acertando, quando ele erra perde um ponto e a dificuldade volta ao iniciar(o valor fica a 10).

G_004

Neste jogo é pedido ao utilizador para introduzir as traduções em portugues das palavras demonstradas em inglês, ganha um ponto por cada resposta certa e perde um por cada errada.

Conclusão

A realização deste trabalho prático permitiu que consolidasse-mos diversas competências de programação em C no Unix.
Depois de concluir o trabalho, vemos que a sua dificuldade se encontra nas pequenas coisas, como funcionamento dos pipes, e threads, é necessário ter a ideia bem estruturada para que todos estes mecanismos funcionem de forma correta.
Apesar de algumas dificuldades que fomos encontrando ao longo do trabalho prático, penso que conseguimos ultrapassar, implementando tudo o que era pedido da maneira que achamos mais correta, baseado nas aulas e pesquisas que fizemos.
Com a conclusão do trabalho podemos dizer que este nos ajudou a consolidar conceitos de namedpides, unamedpipes, threads entre outros.