SISTEMAS OPERATIVOS 2020/2021

Trabalho Prático - Programação em C para UNIX

Makefile

Para a implementação do makefile escrevemos as seguintes rules:

Rule	Dependencies	O que faz
all	arbitro cliente jogos	Compila todos os programas associados ao trabalho
debug	arbitro-debug cliente-debug	Compila todos os programas (menos os jogos) associados ao trabalho para debugging
cliente	cliente.c	Compila o programa cliente
cliente-debug	cliente.c	Compila o programa cliente para debugging
arbitro	arbitro.c arbitro.h	Compila o programa arbitro
arbitro-debug	arbitro.c arbitro.h	Compila o programa arbitro para debugging
jogos	$(GAMES)	Compila todos os jogos presentes no subdiretório Games
%	%.c	Compila o ficheiro %.c
clean	clean-obj clean-exe	Limpa todos os ficheiros gerados pelo makefile
clean-obj	N/A	Limpa todos os ficheiros .o gerados pelo makefile
clean-exe	N/A	Limpa todos os ficheiros executáveis gerados pelo makefile (ficheiros sem extensão)

Para além das rules, para evitar repetir código e permitir o bom funcionamento da rule **jogos**, definimos as seguintes variáveis:

Variável	Valor	Objetivo
CC	gcc -W	Evitar a constante repetição do **gcc** e da flag **warning**
GAMES	$(patsubst %.c, %, $(wildcard ./Games/g_*.c))	Conter o caminho para todos os jogos (só guarda ficheiros que comecem por g_)

Fora a variável criada para guardar o caminho para todos os jogos, criámos também uma rule especial para poder compilar individualmente os jogos. O codigo da rule é o seguinte:

```
%: %.c
    @ $(CC) -c $< -o $@.o
    @ $(CC) $@.o -o $@
    @ echo "Jogo $(patsubst Games/g_%,%, $@) compilado."
```

O **%** indica que qualquer nome pode chamar aquela rule e depois para indicar a dependência temos o **%.c** na mesma linha. Depois nos comandos da rule temos também o uso de variáveis automáticas.

Variável Automática	Significado	Uso
$<	Primeira dependência	Chamar o nome do ficheiro com a extensão **.c** para mostrar que é necessária para a correta execução da rule
$@	Nome da rule	Como esta tem o nome do ficheiro "pai" sem extensão, usamos para chamar os ficheiros criados durante a execução

Árbitro

Para o árbitro fizemos um código inicial bastante simples.

1. Começamos por verificar se as variáveis de ambiente estão corretamente configuradas e, caso isso não se verifique, usamos um valor por defeito ao invés das mesmas.
    - No caso de GAMEDIR usamos o diretório de execução como default;
    - No caso de MAXPLAYERS, como foi referido no enunciado que este "*à partida (...) nunca será superior a 30*", este foi o valor escolhido por defeito.
2. Em seguida vamos ler, através do getopt, as opções para o programa e respetivos argumentos passados pela consola. Se não for incluído um argumento para cada opção (ou esse argumento não for inteiro), o programa avisa o administrador da execução errada do programa. No caso de não ser passada as opções por argumento, associámos um valor default às mesmas.
    - -d: duração do campeonato, em minutos (default a 5);
    - -t: tempo de espera até o campeonato começar automaticamente, em segundos (default a 60).
3. Por fim, temporariamente, imprimimos os restantes argumentos fornecidos ao programa e deixamos uma mensagem de finalização.

No ficheiro header do árbitro criámos uma struct para guardar os dados de um jogador chamada **jogador**. Para além disto, definimos também um novo tipo de variável chamada **bool** para guardar valores *true* ou *false* (1 ou 0, respetivamente).

Tipo	Nome	Objetivo
string	nome	Guardar o nome do jogador
string	jogo	Guardar o nome do jogo que esta a ser jogado
int	PID	Guardar o process ID do cliente
int	score	Guardar a pontuação deste jogador
bool	hasComms	Valor lógico para saber o estado da relação jogador-jogo (0 para suspensa, 1 para ativa)
jogador*	prox	Ligar ao próximo jogador da lista ligada

Optamos por usar uma lista ligada ao invés de um array de structs por nos parecer mais facil de manipular.

Unscrambler

O jogo proposto por nós é um jogo bastante didático onde o objetivo é reorganizar as letras apresentadas de modo a formar a palavra original.

1. Começa por iniciar o gerador de números aleatórios e, com o número obtido, ir buscar uma palavra aleatória ao array de palavras possiveis;
2. Depois envia-a para a função **randomize**, onde vai reordenar a ordem original das letras.
3. No final entra num loop infinito que só acaba quando o jogador acertar a palavra pretendida.

Futuramente serão implementados mais jogos.

Trabalho realizado por:

- TheForgotten
- JOSEALM3IDA

08 de novembro de 2020