Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Sistemas Operativos, Inverno de 2016/17

Segunda Série de Exercícios

I Parte

Biblioteca Uthread

Para cada questão onde não for exigido explicitamente, apresente pelo menos um programa de teste que suporte a correção da solução proposta.

1. Modifique a biblioteca `Uthread` para suportar as seguintes funcionalidades:

   a) Acrescente um campo ao descritor das *uthreads* para indicar o seu estado corrente. Os estados podem ser: `Running`, `Ready` e `Blocked`. Adicione à API a função `INT UtThreadState(HANDLE thread)` que retorna o estado da *thread* passada por parâmetro. Faça as alterações necessárias para manter o estado actualizado.

   b) Realize a função `BOOL UtAlive(HANDLE thread)` que retorna *true* se o *`handle`* passado como argumento corresponder ao de uma *thread* em actividade. Entende-se por *thread* em actividade qualquer *thread* que tenha sido criada e ainda não tenha terminado (não tenha invocado a função `UtExit`), independentemente do seu estado. Sugestão: mantenha uma lista de todas as *threads* em actividade.

   c) Acrescente a função `VOID UtSwitchTo(HANDLE threadToRun)`, que provoca uma comutação imediata de contexto para a *thread* `threadToRun`, se esta se encontrar no estado *ready*. Se não for esse o caso a função não tem nenhum efeito.

   d) Acrescente a função `BOOL UtMultJoin(HANDLE handle[], int size)` que espera pela terminação de todas as *threads* passados no *array* `handle`. No caso de algum *handle não* corresponder a *thread* ***alive*** (de acordo com a função `UtAlive` proposta na alínea b)), ou corresponder à *thread* invocante, a função retorna de imediato com o valor `FALSE`. Caso contrário, espera (usando uma e uma só transição de *running->blocked*) que todas as *threads* terminem, retornando nesse caso o valor `TRUE`.

II Parte

Modelo Computacional do Windows

2. Escreva programas para determinar o tempo de comutação de *threads* no sistema operativo Windows. Teste o tempo de comutação entre *threads* do mesmo processo e entre *threads* de processos distintos. Para a medição de tempos, utilize a função da Windows API `GetTickCount`.

3. O programa em anexo (projeto `JPG_SearchProgram`) obtém o conjunto de todos os ficheiros com imagens JPG presentes num repositório que incluam uma determinada *Tag* de metadados *Exif* e apresenta os nomes dos ficheiros e respectivos caminhos na consola. O repositório e *Tag* são especificados por argumentos da aplicação identificando a diretoria raiz de pesquisa e o valor em base decimal do identificador da *Tag Exif*. A pesquisa é realizada a partir da diretoria raiz e pastas internas. O programa utiliza a DLL `JPGExifUtils` para descodificar as *Tags Exif*, é fornecida em anexo em formato binário e apresenta a seguinte interface pública:

```
typedef BOOL (*PROCESS_EXIF_TAG)(LPCVOID ctx, DWORD tagNumber, LPCVOID value)
VOID JPG_ProcessExifTags(PTCHAR fileImage, PROCESS_EXIF_TAG processor, LPCVOID ctx);
```

   A função `JPG_ProcessExifTags` chama a função `processor` para cada *Tag Exif* standard, privada ou GPS encontrada na imagem JPG `fileImage`. A função `JPG_ProcessExifTags` retorna logo que uma chamada à função `processor` retorne `FALSE` ou quando forem processadas todas as *Tags Exif* presentes em `fileImage`. A função de *callback* `processor` recebe o mesmo contexto `ctx` recebido pela função `JPG_ProcessExifTags`, o identificador da *Tag* e o valor correspondente. O programa utiliza uma única *thread* para realizar o processamento de todas as imagens presentes no repositório.

   Escreva uma versão do programa em anexo que apresenta na consola os ficheiros JPG com fotografias que tenham sido capturadas dentro de um intervalo de datas. Esta versão deverá explorar a multiplicidade de processadores do sistema onde é executado e a unidade de trabalho de cada *thread* é o ficheiro. Valoriza-se uma solução que considere os seguintes aspectos:

   - Criação da DLL `JPGExifUtils` cumprindo a especificação definida em cima adaptando o código da série 1;
   - Utilização de um *pool* de *threads* em vez de uma solução que crie uma *thread* por cada ficheiro. Nesse sentido, sugere-se a utilização de um dos *thread pool* do Windows através da função `QueueUserWorkItem` (consulte o MSDN para mais informação: https://msdn.microsoft.com/en-us/library/windows/desktop/ms684957(v=vs.85).aspx);
   - Participação da *thread* principal no processamento de ficheiros sempre que o número de *threads* da *pool* em actividade atingir um valor previamente estabelecido;
   - Colecção do resultado em memória e respectiva apresentação apenas depois de processados todos os ficheiros JPG do repositório.

4. O serviço em anexo realiza a mesma pesquisa com o mesmo resultado do programa anterior. Neste caso, o processamento de pesquisas é realizada por um único processo (servidor) e distinto dos processos que colocam pesquisas (clientes). Os processos pertencem todos ao mesmo sistema e comunicam entre si através de memória partilhada. O serviço é criado com nome permitindo várias instâncias do serviço com nomes diferentes no mesmo sistema. A solução em anexo inclui três projectos: uma DLL e duas aplicações.

   `JPG_SearchService` – DLL que implementa o serviço e que inclui as componentes servidora e cliente;

   `JPG_SearchServiceServer` – aplicação servidora que processa pesquisas. A pesquisa é realizada por uma única *thread* e apresenta o resultado na consola. O nome do serviço é especificado por argumento da aplicação e esta aplicação deve ser instanciada antes de qualquer cliente;

   `JPG_SearchServiceClient` – aplicação cliente que coloca uma pesquisa e que termina após processamento. O nome do serviço a usar, repositório e filtro são especificados por argumento da aplicação.

   Altere a solução em anexo por forma a que o resultado da pesquisa seja apresentado pela aplicação cliente. A aplicação servidora comunica o resultado da pesquisa para a aplicação cliente através de memória partilhada usando a dimensão de memória estritamente necessária para o efeito. Valoriza-se uma solução que explore a multiplicidade de processadores do sistema onde o serviço é executado.

João Patriarca e Jorge Martins

ISEL, 7 de Maio de 2017

Data de entrega: 4 de Junho de 2017