Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Sistemas Operativos, Inverno de 2018/19

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) Realize a função `DWORD UtJoin(HANDLE thread)` que não retorna enquanto não terminar a *thread* correspondente ao *handle* passado como argumento ou a operação *Join* não tiver sido cancelada. A função retorna 1 com a terminação da *thread* `thread`, 0 se o *handle* passado não corresponder a uma *thread* em actividade, ou -1 se a operação tiver sido cancelada. Considere uma *thread* em actividade, qualquer *thread* que ainda não tenha evocado a função `UtExit.`

   d) Realize a função `BOOL UtJoinCancel(HANDLE thread)` que cancela a operação *Join* executada pela *thread* correspondente ao *handle* passado como argumento. O sucesso da operação depende do *handle* passado corresponder a uma *thread* bloqueada na operação *Join*.

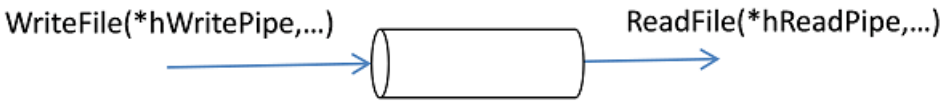
   e) Acrescente a função `VOID UtSwitchTo(HANDLE threadToRun)`, que provoca uma comutação imediata de contexto para a *thread* `threadToRun`, se esta se encontrar no estado *ready*. Se não for esse o caso a função não tem nenhum efeito.

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. A função da Windows API:

```
BOOL WINAPI CreatePipe(PHANDLE hReadPipe, PHANDLE hWritePipe,
                       LPSECURITY_ATTRIBUTES lpPipeAttributes, DWORD nSize);
```

Cria um canal unidirecional de comunicação entre processos. Na execução com sucesso da função, os *handles* apontados por `hWritePipe` e `hReadPipe` permitem, respetivamente, a escrita e a leitura no *pipe*, utilizando as operações usadas na escrita e leitura de ficheiros (`WriteFile` e `ReadFile`), tal como mostra a fig. seguinte:



O *pipe* funciona como sincronizador: estando cheio, a escrita bloqueia até haver espaço suficiente e, estando vazio, a leitura bloqueia até existirem dados no *pipe*.

Implemente a aplicação `Talk`, que cria um processo filho numa consola distinta e, tirando partido de dois *pipes*, permite a comunicação bidirecional entre os processos pai e filho, com o seguinte comportamento: o que for lido do *standard input* (*stdin*) do pai é mostrado no *standard output* (*stdout*) do filho e o que for lido do *stdin* do filho é mostrado no *stdout* do pai.

Notas de implementação:

- Para obter os *handles* correspondentes ao *stdin* e *stdout* de um processo utilize a função `GetStdHandle;`
- Os *pipes* são criados no processo pai e os *handles* necessários ao filho são passados por herança;
- Na sua implementação deverá suportar a terminação controlada dos processos intervenientes.

4. O programa em anexo organiza o armazenamento de fotografias no formato JPG baseado na data em que foram tiradas. O programa recebe por argumento as directorias origem e destino e um indicador para eliminar, ou não, as fotografias transferidas da directoria origem. O programa cria directorias com o formato `YYYY_MM_DD` na directoria destino, copia as fotografias para as respectivas directorias e apaga-as da directoria origem. O programa utiliza a DLL `JPGExifUtilsLibrary` para descodificar *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 na directoria origem.

Escreva uma versão do mesmo programa explorando a multiplicidade de processadores do sistema onde é executado. A *thread* principal é responsável pela distribuição de trabalho e o processamento de um ficheiro é delegado numa *thread* de trabalho. Valoriza-se uma solução que considere os seguintes aspectos:

- 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);
- Controlo sobre o nível máximo de concorrência bloqueando a *thread* distribuidora de trabalho até que o nível máximo de concorrência deixe de ser atingido.

João Patriarca
ISEL, 5 de Novembro de 2018
Data de entrega: 3 de Dezembro de 2018