Ficha Prática P9

Sistemas Operativos: Threads

1. Consultar os apontamentos sobre threads

   https://www.di.ubi.pt/~operativos/praticos/pdf/9-threads.pdf

2. Implementação no seu interpretador de comandos : Avisos

Criar uma função de '*aviso'* – uma função que depois de esperar um certo número de segundos imprime uma mensagem no ecrã.

no ficheiro threads.c

```
void aviso (char *mesg, int tempo)
{
   while (tempo > 0) {
      sleep (1);
      tempo--;
  }
  fprintf (stderr, "Aviso : %s\n", mesg );
}
```

Implemente esta função **no ficheiro threads.c** e testar no seu Shell (no builtin) criando um comando ***avisoTeste***

Nota que o Shell vai bloquear até que a função *aviso* tenha executada até o fim.

No main.c (builtin)

```
if ( 0 == strcmp ( args[0], "avisoTeste") ) {
      aviso( args[1], atoi ( args[2] ) ) ;
      return 1;
}
```

Como a função atoi não verifica erros deve tambem verificar o numero de argumentos e que args[2] não sej null.

3. Converter uma função existente para execução numa thread separada usando a técnica duma função intermediaria de embrulho ou "wrapper"

(a) Criar um comando para lançar threads de *Avisos*. Nota: são threads ***detached*** → não é necessário fazer join

No main.c (builtin)

```
if ( 0 == strcmp (args[0], "avisomau") ){
      pthread_t th;
      pthread_create(&th, NULL, avisowrapper MAU, (void *)args);
      return 1;
}
```

No threads.c

```
void * avisowrapperMAU(void *args) {
      char ** pargs = (char **) args;
      aviso( pargs[1], atoi ( pargs[2]) ) ;
      return NULL;
}
```

Teste 1
Soshell> avisomau ola 3

Teste 2
Soshell > avisomau ola 10
Soshell > avisomau adeus 2

Tem que escrever os dois comandos ..rapidos e .. Verifique que esta solução esteja **incorreta**!

(b) Solução Boa: Utilize **uma estrutura segura** para passar os argumentos

No ficheiro shell.h : `typedef struct { char msg[100] ; int tempo;} aviso_t;`

No main.c (builtin) //embrulhar

```
if ( 0 == strcmp (args[0], "aviso") ){
      pthread_t th;
      aviso_t * ptr = (aviso_t *)malloc( sizeof(aviso_t) );
      strcpy(ptr->msg, args[1])
      ptr->tempo=atoi(args[2]);
      pthread_create(&th, NULL, avisowrapper, (void *)ptr);
      return 1;
}
```

No threads.c //desembrulhar

```
void * avisowrapper(void *args) {
       aviso_t * ptr = (aviso_t *)args;
       aviso( ptr->msg, ptr->tempo ) ;
       free(ptr);
       return NULL;
}
```

*** Implementar e Verificar que esta solução esteja **correta**!

4 Implementação da função "builtin" socpth

A) Implemente a função *socpthread* que efetuará a copia do ficheiro "fonte" para "destino" usando I/O de baixo-nível com tamanho de bloco variável. A cópia será efetuada **numa nova thread (detached)**

Shell Sintaxe: socpthread fonte destino [blksize]

Vai utilizar a seguinte função para efetuar a copia que implica a abertura previa dos ficheiros que deverá ter previamente implementada nas aulas anteriores

Dependendo do que tem implementado …

```
void socp(char *fonte, char *destino);
ou
void socp(char *fonte, char *destino, int buffsize);    //melhor
```

Dica: No ficheiro shell.h e conforme a sua implementação especifca algo assim

- `typedef struct { char fonte[100]; char destino[100]; } copiar_t;`
- `typedef struct { char fonte[100]; char destino[100]; int buffsize } copiar_t;`

A função do "wrapper" deve ser escrita no ficheiro threads.c

B) Testar a sua solução copiando um ficheiro grande. Desta maneira, usando um tamanho de bloco pequeno pode monitorizar a thread a copiar este ficheito no seu shel.

Por exemplo para criar um ficheiro de 100 MB com números aleatórios usar o comando seguinte

```
soshell > dd if=/dev/urandom of=bigfile bs=10M count=10
10+0 records in
10+0 records out
104857600 bytes (105 MB, 100 MiB) copied, 0.085198 s, 1.2 GB/s
soshell > ls -lh bigfile
-rw-r--r-- 1 user user 100M Apr 20 15:29 bigfile
```

soshell > *socpthread* `bigfile thecopy` 16 //blocksize de 16 bytes

```
soshell > ls -lh thecopy
-rw-r--r-- 1 user user 33M Apr 20 16:29 thecopy
soshell > ls -lh thecopy
-rw-r--r-- 1 user user 82M Apr 20 16:52 thecopy
```

etc.

C) Relatório: No final da Copia a função deverá escrever para um vetor de strings uma mensagem "Date/Time fileName" para relatar que a copia foi efetuada com sucesso ou não e quando.
Também vai precisar duma função embutida no shell para listar estas informações – como no exemplo em baixo (implementar a função no ficheiro threads.c)

```
char strings[MAX][130]; int indices..
time_t tempoAtual;
time(&tempoAtual);
char *diaHora = ctime(&tempoAtual);
if ('\n' == t[strlen(t)-1]) t[strlen(t)-1] = '\0'; //remover \n inserido pelo ctime
sprintf(strings[k++%MAX], "%s %s", diaHora, fileName);
```

soshell> **lnfoCopias**
Sun Apr 20 21:16:24 2024 bigFile
Sun Apr 20 21:16:42 2024 db2.pecas