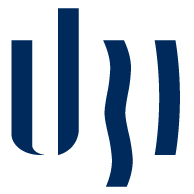
**Ficha Prática 7:** Sistemas Operativos 2023



***1. Capitulo 9 – THREADs http://www.di.ubi.pt/~operativos/praticos/html/9-threads.html***

***2. Avisos***

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

**no aviso.c**

void aviso (char \*mesg, int tempo)

{

while (tempo > 0) {

sleep (1);

tempo--;

}

fprintf (stderr, "Aviso : %s\n", mesg );

}

Implemente esta função **no ficheiro aviso.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 (strcmp (args[0], "avisoTeste")==0 ) {

aviso( args[1], atoi ( args[2]) ) ;

return 1;

}

## 3. Convertir para execução numa thread separada usando a técnica duma função intermediaria … as vezes chamado “wrapper”

1. Criar um comando para lançar threads de *Avisos*. Nota: são threads detached 🡪 não é necessário fazer join

**No main.c (builtin)**

if (strcmp (args[0], "aviso") == 0){

pthread\_t th;

pthread\_create(&th, NULL, avisowrapper, (void \*)args);

return 1;

}

**No aviso.c**

void \* aviso**wrapper**(void \*args) {

char \*\* argsin = (char \*\*)args;

aviso( argsin[1], atoi ( argsin[2]) ) ;

return NULL;

}

Teste 1

Soshell> aviso ola 3

Teste 2

Soshell > aviso ola 10

Soshell > aviso adeus 2

Verifique que esta solução esteja **incorreta**!

1. 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)**

if (strcmp (args[0], "aviso") == 0){

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 aviso.c**

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**

Implemente a função *socpth* 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: socpth fonte destino***

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);**

**ou**

**void socp(int in, int out);**

**Dica No ficheiro shell.h e conforme a sua implementação…**

* typedef struct { char fonte[100]; char destino[100]; } copiar\_t;
* typedef struct { char fonte[100]; char destino[100]; int buffsize } copiar\_t;
* typedef struct { int in , int out} copiar\_t;

**Testar a sua solução** copiando um ficheiro grande. Por exemplo para criar um ficvheiro de 100 MB com numeros aleatorios usar o comando seguinte

ubuntu > 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

ubuntu >ls -lh big

-rw-r--r-- 1 user user 100M Apr 20 15:29 bigfile

ubuntu >

**Um exercício suplementar interessante. Arranjar** um sistema em qual a thread que faça a copia envie um notificação para o programa principal do shell para dizer que a copia foi efectuada com sucesso ou não.