Implementação dum Interpretador de Comandos: "SoShell"

4.1 Obter os ficheiros Necessários.

a) Via GIT: git clone https://gitlab.com/crkxcariy/soubi.git
b) VIA URL (browser) https://gitlab.com/crkxcariy/soubi/-/tree/master/soshell
c) Para **fazer download** da pasta necessária poderá usar o seguinte (copy/paste os comandos em baixo)

```
wget https://gitlab.com/crkxcariy/soubi/-/archive/master/soubi-master.zip?path=soshell
unzip 'soubi-master.zip?path=soshell'
mv soubi-master-soshell/soshell   .
rm 'soubi-master.zip?path=soshell' ; rmdir soubi-master-soshell
cd soshell ; pwd ; ls
```

- **Compilar usando "make" e executar com >soshell ou >./soshell (conforme variavel PATH)**

Deverá alterar os ficheiros para o projeto compilar sem avisos! Pode ter que adicionar por exemplo mais "includes" no ficheiro shell.h

Explicação e Código: Capítulo 8 da Sebenta das Aulas Praticas:
https://www.di.ubi.pt/~operativos/praticos/html/8-processos.html (pdf) https://www.di.ubi.pt/~operativos/praticos/pdf/8-processos.pdf

- **Faça diagramas (i) das dependencias dos ficheiros e (ii) Fluxograma do programa e (iii) as variaveis buf e args**

2 Aumente a funcionalidade do seu Shell

(i) implemente a possibilidade de mudar dinamicamente (durante runtime) o prompt usando o comando prompt> PS1=String : onde String será o novo prompt

(ii) implemente um comando embutido chamado "quemsoueu" que mostre detalhes sobre a identificação do usuário. Baste chamar a função system("id") na função "builtin".

(iii) implemente o comando embutido "socp" que permite o shell copiar um ficheiro através do comando prompt> socpy destino fonte

4.2 Implementação da função embutida *obterinfo*. O esboço da solução é dada em baixo

```
SOSHELL: prompt> ls
            shell.h main.c ….
SOSHELL: prompt> obterinfo
       SoShell 2023 versão 1.0
SOSHELL: prompt>
```

```
if( 0==strcmp(args[0], "obterinfo") ){
    printf("SoShell 2022 versaõ 1.0\n");
    return 1 ;  //comando embutido
}
```

4.3 Implementação da função da mudança do *prompt* (Exercício do capitulo 8) - Deverá ser possível de forma análoga à mudança de *prompt* no *Bash Shell*, através do comando **`PS1=string`**

```
SOSHELL: prompt> PS1=ruishell:>
ruishell:>
```

```
if( strlen(args[0])>4  &&   0==strncmp(args[0], "PS1=",4) ){
    strcpy(prompt,args[0]+4);
    return 1 ;  //comando embutido
}
```

4.4 Implementação da função *builtin* quemsoueu **(Exercício do capitulo 8)**

```
SOSHELL: prompt> quemsoueu
uid=1000(crocker) gid=1000(crocker) groups=1000(crocker),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev)
```

```
if( 0==strcmp(args[0], "quemsoueu") ){
      system("id");
     return 1 ;  //comando embutido
}
```

4.5 Implementação da função *builtin* para mudança do diretório (**cd**). O objetivo é implementar a funcionalidade de mudança de diretório, de forma a que a sintaxe fique semelhante a:

```
SOSHELL: prompt> cd [directory]
```

Utilizará a chamada ao sistema **chdir**. Note que escrevendo apenas **cd** ou **cd ~** implica mudar para o directório raiz do utilizador. O directório raiz do utilizador pode ser achado usando a função **getenv()** como por exemplo **char *userRaiz = getenv("HOME")**. Considere o esboço da solução em baixo:

```
if (0 == strcmp(args[0], "cd")) {
  int err;
  if  NULL == args[1]  OU   args[1] é igual ao string "~"
      err=chdir( getenv("HOME") ) ;
  else
      err = chdir( args[1] )
  if (err<0) perror ( args[1]);
  return 1 ;  //comando embutido
}
```

Suplementar

```
|| OU   args[1] é igual ao string "$HOME"
Verificar / Implementar  cd ..
Implementar cd - : Feature para regressar a pasta anterior
Ver https://man7.org/linux/man-pages/man1/cd.1p.html
(Pode ser implementado com uma pilha de pastas i.e podemos regressar
para as pastas onde estavamos por um ordem especifico)
```

4.6 Exercício. Implementação do comando embutido socp para copiar um ficheiro

Utilizar I/O de baixo nível e aproveita a solução das semanas passada

Criar um ficheiro socp.c com as duas funções seguintes necessárias.

void socp(char *fonte, char *destino) ;

void ioCopy(int IN,int OUT);

Na função "BuiltIn" detetar o comando adicionado o seguinte.

```
if( 0==strcmp(args[0], "socp") ){
      socp( args[1], args[2] );
     return 1 ;  //comando embutido
}
```

Nota: Pode depois adicionar a opção do ter o blocksize como argumento.

Not a: Vai ter de modificar o seu Makefile

```
OBS=main.o execute.o parse.o socp.o
socp.o : shell.h socp.c
        $(CC) $(FLAGS) socp.c
```

E faça as alterações no ficheiro shell.h necessárias adicionado os protótipos das funções novas

4.7 Tratamento dum pedido de execução dum programa em segundo plano - "background" &

exemplo ***SoShell*** *foreground (fg)* ***soprompt > gedit*** OU ***./1.sh ola*** ***(fazer antes chmod 755 1.sh)***
--O shell deve estar bloqueado. Experimente com gedit ou 1.sh

exemplo ***SoShell*** *background (bg)* ***soprompt > gedit &*** OU ***./1.sh ola &***
--Deve ser possivel ainda usar o shell. Fazer ps para os processos em execução e kill -9 pid para os terminar

Implementação

Mudança do *parser* para devolver o número de argumentos da linha de comandos e mudança da função *execute* para aceitar este valor. A função ***main()*** terá que apenas comunicar este valor entre o parser e a função *execute*, assim:

```
int numargs = parse( buf, args );     /* particiona a string em argumentos */
…..
execute( numargs, args  );                    /* executa o comando */
```

- Mudar os protótipos no Shell.h !!

- **p**arse.c : **Contagem de Numero de Argumentos**

```
int cnt=0;              /*novo: initialize numero de argumentos */
while ..
{
  *args++ = buf;      /* salvaguarda argumento   */
   cnt++     ;        /*novo: incremente do numero de argumentos */
}
return cnt;
```

- **Na ficheiro execute.c acrescentar a função seguinte para detetar a presença do & e caso que seja detetado decrementar o numero de argumentos e anular a apontador para "&"**

```
int ultimo ( int *numargs, char **args )
{
   if ( args[*numargs-1][0]=='&') {
      *numargs=*numargs-1;
      args[*numargs]=NULL ;
      return BG;
    }
   return FG;        /*return FG ou BG definidos no shell.h */
}
```

Na função ***execute()*** deverá utilizar este função para detectar se o último argumento é o "&". No caso de ser detectado este terá de ser anulado da lista de argumentos a passar para o novo comando. A função auxiliar ***ultimo()*** fará este trabalho.

As outras mudanças na função ***execute()*** são bastantes triviais esperamos com wait()) ou não !!!.

Esboço da solução :

```
int code = ultimo( &numargs, args);

..pid = fork(..)

..if (0==pid) …execvp(..)

if ( FG==code ) while( wait(..) );
```

Necessitar de tee Conhecimentos sobre :
- Funcionamento dum CLI (Shell)
- Criação de novos processos com fork() e Programas com exec()
- Execução da funcionalidade embutida versus execução dum novo processo.
- Utilizar um Makefile para compilar
- Diferença entre execução em foreground (primeiro plano) vs background (segundo plano)