```c
/*****************************************************************
  DISCIPLINA: 203523 - Sistemas Operacionais

  Shell para execução de comandos simples;
     - Esse código roda em Linux;

*****************************************************************/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main( void )
{
 char lc[ 81 ];
 char *argv[ 20 ];
 int pid, i, status;

 while( 1 ) {
    printf( "Prompt > " );
    gets( lc );  /*recebe comando de entrada*/
    if( ! strcmp( lc, "" ) ) /*se vazio, cancela*/
        continue;
    argv[ 0 ] = strtok( lc, " " );
    if( ! strcmp( argv[ 0 ], "exit" ) )
        exit( 0 );
    i = 1;
    while( i < 20 && (argv[ i ] = strtok( NULL, " " )) )// char * strtok ( char * str, const
char * delimiters );
                            /*maximo de 20 char
                            SINTAXW: char * strtok( char * enderecoStrOrigem, char *
enderecoStrDelimitador);
                            A função strtok devolve um ponteiro para a proxima palavra apontada
por
                            por enderecoStrOrigem. Os caracteres que formam a string apontada por
                            enderecoStrDelimitador são os delimitadores que terminam a palavra. Um
                            ponteiro nulo é devolvido quando não há mais palavras na
string.
                            */
         ++i;
    if( (pid = fork()) == -1 ) {/*valor negativo = erro*/
         printf( "Erro no fork\n" );
         exit( 1 );
    }
    if( pid == 0 ) /*se for processo filho*/

                    /*
                        A função forke() duplica o processo atual
                        dentro do sistema operacional. O processo que chama
                        o fork() é chamado de processo pai, e o processo
                        criado pelo fork() é chamado processo filho. Todas
                        as áreas do processo são duplicadas dentro do sistema operacional,
                        (código, dados, pilha, memória dinâmica.
                    */
    if( execvp( argv[ 0 ], argv ) ) {/*Executa o vetor de parametros passado*/

            /*
                As funções execv e execvp fornecem
                um vetor de ponteiros para strings
                não-nulas que representam a lista de
                argumentos para o programa executado.
            */
        printf( "Comando nao reconhecido\n" );
        exit( 1 );
    }
     wait( &status );
 }
}
```