Sistemas de Operação (2018/2019)
Teste de Auto-Avaliação Prática
(duração 1 hora)

Resolva as 3 questões que se seguem da forma mais independente possível, evitando consultar a Internet ou os materiais das aulas.

Q1. Considere o seguinte programa `prog.c` que abre um ficheiro de texto, cujo nome é dado na linha de comando, e transfere para o `stdout` o conteúdo do mesmo convertido para minúsculas ou maiúsculas.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

typedef enum amode {UPPER=0, LOWER=1} mode;

#define CONVERT(X,Y)  ((X)==UPPER)? toupper(Y): tolower(Y)

int main(int argc, char* argv[]) {
  mode m;

  /* check if 2 args available */
  if (argc != 3) {
    printf("usage: chcase -u|-l file\n");
    return EXIT_FAILURE;
  }

  /* check if option is valid */
  if(strcmp(argv[1], "-u") == 0 )
    m = UPPER;
  else
  if(strcmp(argv[1], "-l") == 0 )
    m = LOWER;
  else {
    printf("usage: chcase -u|-l file\n");
    return EXIT_FAILURE;
  }
```

```
  /* check if argv[2] can be opened and is readable */
  FILE* fp = fopen(argv[2], "r");
  if(fp == NULL) {
    printf("%s: cannot open %s\n", argv[0], argv[2]);
    return EXIT_FAILURE;
  }

  char c;
  /* get one char at a time and print upper or lower case */
  while(fread(&c, 1, 1, fp) != 0) {
    fputc(CONVERT(m,c), stdout);
  }

  /* close file */
  fclose(fp);
  return EXIT_SUCCESS;
}
```

Modifique o programa, eliminando **todo** o código desnecessário, de forma a que imprima sempre e apenas o número de caracteres e o número de linhas do ficheiro, como no exemplo seguinte (os valores apresentados são fictícios):

```
{glaurung: ~luis}$ gcc prog.c -o prog
{glaurung: ~luis}$ ./prog prog.c
{glaurung: ~luis}$ prog.c: 200 chars, 23 lines
```

Q2. O programa que se segue cria um processo filho que executa o comando simples (i.e., sem argumentos, e.g., `ls`, mas não `ls -l`) dado em `argv[1]`.

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

int main(int argc, char* argv[]) {
  pid_t pid;

  /* fork a child process */
  if ((pid = fork()) < 0 ) {
    printf("%s: cannot fork()\n", argv[0]);
    return EXIT_FAILURE;
  }
  else if (pid == 0) {
```

```
    /* child process */
    if (execlp(argv[1],argv[1],NULL) < 0) {
      printf("bummer, did not exec %s\n", argv[1]);
      return EXIT_FAILURE;
    }
  }
  else {
    /* parent process */
    if (waitpid(pid, NULL, 0) < 0) {
      printf("%s: cannot wait for child\n", argv[0]);
      return EXIT_FAILURE;
    }
    printf("child exited\n");
  }
  return EXIT_SUCCESS;
}
```

Compile e experimente o programa, por exemplo:

```
{glaurung: ~luis}$ gcc prog.c -o prog
{glaurung: ~luis}$ ./prog ls
...   // output de ls
{glaurung: ~luis}$ ./prog pwd
...   // output de pwd
```

Transforme-o numa shell de comandos básica, permitindo que leia sucessivos comandos simples e os execute. A shell deve terminar com o comando especial `quit`. Por exemplo:

```
{glaurung: ~luis}$ gcc prog.c -o prog
{glaurung: ~luis}$ ./prog
$ ls
... // output de ls
$ pwd
... // output de pwd
$ quit
{glaurung: ~luis}$
```

Q3. O programa seguinte cria um processo filho e transfere para ele uma mensagem de texto através de uma "pipe" partilhada.

```
#include <sys/wait.h>
#include <errno.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define READ 0
#define WRITE 1
#define LINESIZE  256

int main(int argc, char* argv[]) {
  int   n, r, fd[2];
  pid_t pid;
  char  line[LINESIZE];

  if (pipe(fd) < 0) {
    perror("pipe error");
    exit(EXIT_FAILURE);
  }

  if ((pid = fork()) < 0) {
    perror("fork error");
    exit(EXIT_FAILURE);
  }
  else if (pid > 0) {
    /* parent */
    close(fd[READ]);
    snprintf(line, LINESIZE, "Hello! I'm your parent pid %d!\n", getpid());
    if ((r = write(fd[WRITE], line, strlen(line))) < 0) {
      fprintf(stderr, "Unable to write to pipe: %s\n", strerror(errno));
    }
    close(fd[WRITE]);
    /* wait for child and exit */
    if ( waitpid(pid, NULL, 0) < 0) {
      fprintf(stderr, "Cannot wait for child: %s\n", strerror(errno));
    }
    exit(EXIT_SUCCESS);
  }
  else {
    /* child */
    close(fd[WRITE]);
    if ((n = read(fd[READ], line, LINESIZE)) < 0 ) {
      fprintf(stderr, "Unable to read from pipe: %s\n", strerror(errno));
    }
    close(fd[READ]);
```

```
    /* write message from parent */
    write(STDOUT_FILENO, line, n);
    /* exit gracefully */
    exit(EXIT_SUCCESS);
  }
}
```

Altere o programa de modo a inverter os papéis: o filho abre um ficheiro de texto e transfere-o na totalidade para o pai através da "pipe". O pai deve escrever o ficheiro recebido para o `stdout`. Sugestão: pode aproveitar o código de abertura e leitura de ficheiro da pergunta **Q1**.