Exame de Sistemas de Operação — 13 de Junho de 2006

Parte Prática **A**

1. Comunicação entre Processos

Pretende-se simular a comunicação entre processos interligados por pipes. O programa a desenvolver simulará a comunicação entre quatro processos colocados numa *pipeline* de comunicação, como se ilustra na figura seguinte:

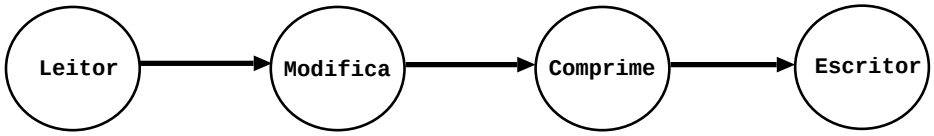


Figura 1: Fluxo de informação num encadeamento de 4 processos

Os processos envolvidos no encadeamento têm as seguintes funções:

- O processo `Leitor`, lê linhas de um ficheiro `original.txt` e envia cada linha ao processo `Modifica`;
- Para cada linha recebida pelo processo `Modifica`, este substitui as letras 'a' e 'o' pelo caracter espaço, enviando depois a linha transformada para o processo `Comprime`;
- Para cada linha recebida pelo processo `Comprime`, este remove todos os caracteres espaço e envia a linha comprimida resultante para o processo `Escritor`;
- O processo `Escritor` escreve para um ficheiro `final.txt` as linhas que recebe, colocando o separador de mudança de linha após cada linha que escreve.

Algumas funções úteis (informação obtida das man-pages)

```
int pipe(int filedes[2]);
  pipe() creates a pair of file descriptors, pointing to a pipe inode, and
  places them in the array pointed to by filedes. filedes[0] is for
  reading, filedes[1] is for writing. On success, zero is returned.  On
  error, -1 is returned, and errno is set appropriately.

int open(const char *pathname, int flags);
  flags is one of O_RDONLY, O_WRONLY or O_RDWR which request opening the
  file read-only, write-only or read/write, respectively. flags may also be
  bitwise-or'd with one or more of the following: O_CREAT If the file does
  not exist it will be created.  O_TRUNC If the file already exists it will
  be truncated.  O_APPEND The file is opened in append mode.

int close(int filedesc);
  closes a file descriptor, so that it no longer refers to any file and may
  be reused.
```

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
  dup and  dup2 create a copy of the file descriptor  oldfd.  dup uses the
  lowest-numbered unused descriptor for the new descriptor.  dup2 makes
  newfd be the copy of oldfd, closing newfd first if necessary.


ssize_t read(int fd, void *buf, size_t count);
  read() attempts to read up to count bytes from file descriptor fd into
  the buffer starting at buf. On success, the number of bytes read is
  returned (zero indicates end of file), and the file position is advanced
  by this number. It is not an error if this number is smaller than the
  number of bytes requested; this may happen for example because fewer
  bytes are actually available right now ...


ssize_t write(int fd, const void *buf, size_t count);
  write() writes up to count bytes to the file referenced by the file
  descriptor fd from the buffer starting at buf. On success, the number of
  bytes written are returned (zero indicates nothing was written).  On
  error, -1 is returned, and errno is set appropriately.


pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
  The wait function suspends execution of the current process until a child
  has exited. If a child has already exited by the time of the call, the
  function returns immediately. The waitpid function suspends execution of
  the current process until a child as specified by the pid argument has
  exited.  The value of options is an OR of zero or more of the following
  constants: WNOHANG which means to return immediately if no child has
  exited.  WUNTRACED which means to also return for children which are
  stopped.


void perror(const char *s);
  The routine perror() produces a message on the standard error output,
  describing the last error encountered during a call to a system or
  library function. The argument string s is printed first, then a colon
  and a blank, then the message and a new-line.


int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
  The functions in the printf family produce output according to a format
  as described below. sprintf, snprintf, vsprintf and vsnprintf write to
  the character string str.  Upon successful return, these functions return
  the number of characters printed (not including the trailing '\0' used to
  end output to strings). The functions snprintf and vsnprintf do not write
  more than size bytes (including the trailing '\0'). If the output was
  truncated due to this limit then the return value is the  number of
  characters (not including the trailing '\0') which would have been
  written to the final string if enough space had been available. Thus, a
  return  value of size or more means that the output was truncated. (See
  also below under NOTES.) If an output error is encountered, a negative
  value is returned.
```

2. Memória Virtual

Considere as instrucções de *input* e as operações de *output* de acesso à memória definidas no trabalho prático.

```
    Instrucções de input           Operações de output
--------------------------     --------------------------
R end                          load  M(mold) P(pag)
W end val                      store M(mold) P(pag)
                               read  M(mold) D(des) V(val)
                               write M(mold) D(des) V(val)
```

Considere ainda um sistema de gestão de memória virtual com paginação de um nível, e que dispõe de 5120 *bytes* de memória virtual e 3 molduras de página com 1024 *bytes* cada. O sistema funciona com uma interrupção de relógio a cada 2 instruções (a primeira instrução é a número 1) e utiliza o algoritmo NFU com contador de 8 bits para substituição de páginas. Antes de executar a décima instrução, o sistema apresenta a tabela de páginas com a seguinte configuração:

```
------------------------------
Page | Mold Load Last Cont R M
------------------------------
   0 |    -    -    -    - - -
   1 |    0    1    -  112 0 1
   2 |    1    2    -  112 0 0
   3 |    -    -    -    - - -
   4 |    2    7    -  128 1 1
------------------------------
```

a) Indique quais seriam as operações de *output* de acesso à memória realizadas pelo sistema para executar o conjunto de instruções de *input* que se segue:

```
W 1000 10
W 2000 20
R 1000
W 3000 30
```

b) Considere a estrutura de dados a seguir apresentada como forma de manter a informação sobre as entradas na tabela de páginas.

```
typedef enum {OFF=0, ON=1} bit;

typedef struct table_entry {
  unsigned int Mold;
  unsigned int Load;
  unsigned int Last;
  unsigned char Cont;
  bit R;
  bit M;
} Table_Entry;
```

Escreva a função `'int nfu(Table_Entry *tab_pages, int num_pages)'` que implementa o algoritmo de substituicão de páginas NFU com contador de 8 bits. A função recebe como argumentos o apontador para a tabela de páginas (`tab_pages`) e o número de páginas (`num_pages`) e deverá devolver o número da página a ser substituída.

3. Sistema de Ficheiros

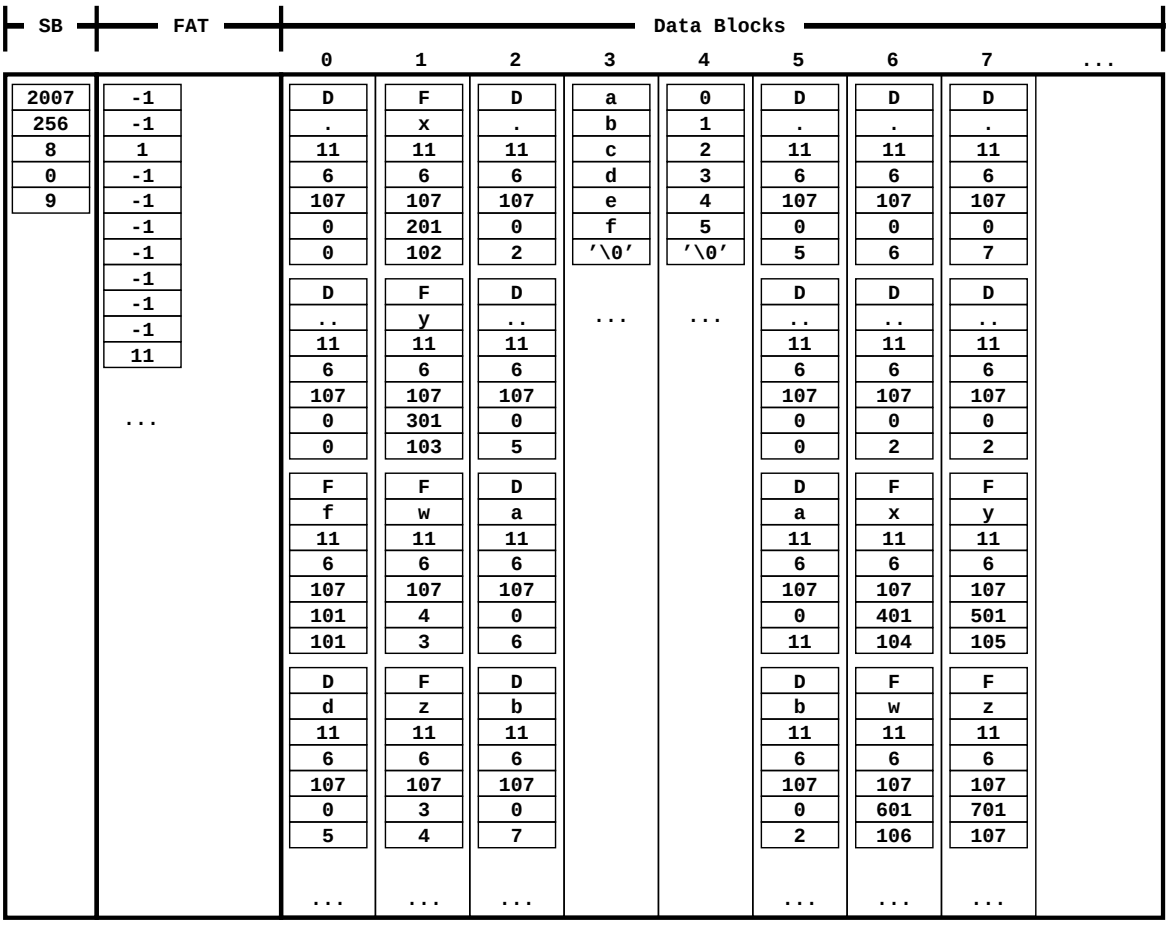
A figura abaixo representa parte de um sistema de ficheiros organizado segundo as estruturas de dados utilizadas no trabalho prático (considere inteiros de 4 *bytes*).

```
#define TYPE_DIR  'D'
#define TYPE_FILE 'F'
#define TYPE_FREE '-'

typedef struct superblock_entry {
  int check_number;
  int block_size;
  int fat_type;
  int root_block;
  int free_block;
} superblock;
```

```
#define MAX_NAME_LENGHT 20

typedef struct directory_entry {
  char type;
  char name[MAX_NAME_LENGHT];
  unsigned char day;
  unsigned char month;
  unsigned char year;
  int size;
  int first_block;
} dir_entry;
```



Considerando que o bloco de dados 2 é o primeiro bloco do directório corrente e que este possui 16 entradas, incluindo as entradas '.' e '..', indique justificando:

a) o número de blocos ocupado pela FAT e o número de blocos de dados em utilização.

b) o resultado da execução do comando *'pwd'*.

c) o resultado da execução do comando *'cat w'*.

e) o resultado da execução do comando *'cp y a'* (assuma que a entrada *y* não existe em *a*).

d) o resultado da execução do comando *'mkdir d'* (assuma que a entrada *d* não existe no directório corrente).

Exame de Sistemas de Operação — 13 de Junho de 2006

Parte Prática **B**

1. Comunicação entre Processos

Pretende-se simular a comunicação entre processos interligados por pipes. O programa a desenvolver simulará a comunicação entre quatro processos colocados numa *pipeline* de comunicação, como se ilustra na figura seguinte:

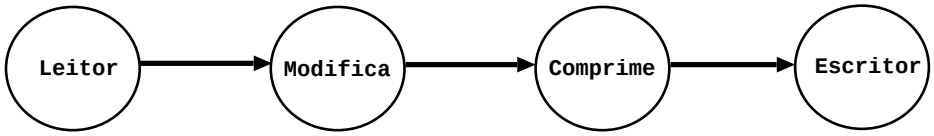


Figura 2: Fluxo de informação num encadeamento de 4 processos

Os processos envolvidos no encadeamento têm as seguintes funções:

- O processo `Leitor`, lê linhas de um ficheiro `original.txt` e envia cada linha ao processo `Modifica`;
- Para cada linha recebida pelo processo `Modifica`, este substitui as letras 'a' e 'o' pelo caracter espaço, enviando depois a linha transformada para o processo `Comprime`;
- Para cada linha recebida pelo processo `Comprime`, este remove todos os caracteres espaço e envia a linha comprimida resultante para o processo `Escritor`;
- O processo `Escritor` escreve para um ficheiro `final.txt` as linhas que recebe, colocando o separador de mudança de linha após cada linha que escreve.

Algumas funções úteis (informação obtida das man-pages)

```
int pipe(int filedes[2]);
  pipe() creates a pair of file descriptors, pointing to a pipe inode, and
  places them in the array pointed to by filedes. filedes[0] is for
  reading, filedes[1] is for writing. On success, zero is returned.  On
  error, -1 is returned, and errno is set appropriately.

int open(const char *pathname, int flags);
  flags is one of O_RDONLY, O_WRONLY or O_RDWR which request opening the
  file read-only, write-only or read/write, respectively. flags may also be
  bitwise-or'd with one or more of the following: O_CREAT If the file does
  not exist it will be created.  O_TRUNC If the file already exists it will
  be truncated.  O_APPEND The file is opened in append mode.

int close(int filedesc);
  closes a file descriptor, so that it no longer refers to any file and may
  be reused.
```

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
  dup and  dup2 create a copy of the file descriptor  oldfd.  dup uses the
  lowest-numbered unused descriptor for the new descriptor.  dup2 makes
  newfd be the copy of oldfd, closing newfd first if necessary.


ssize_t read(int fd, void *buf, size_t count);
  read() attempts to read up to count bytes from file descriptor fd into
  the buffer starting at buf. On success, the number of bytes read is
  returned (zero indicates end of file), and the file position is advanced
  by this number. It is not an error if this number is smaller than the
  number of bytes requested; this may happen for example because fewer
  bytes are actually available right now ...


ssize_t write(int fd, const void *buf, size_t count);
  write() writes up to count bytes to the file referenced by the file
  descriptor fd from the buffer starting at buf. On success, the number of
  bytes written are returned (zero indicates nothing was written).  On
  error, -1 is returned, and errno is set appropriately.


pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
  The wait function suspends execution of the current process until a child
  has exited. If a child has already exited by the time of the call, the
  function returns immediately. The waitpid function suspends execution of
  the current process until a child as specified by the pid argument has
  exited.  The value of options is an OR of zero or more of the following
  constants: WNOHANG which means to return immediately if no child has
  exited.  WUNTRACED which means to also return for children which are
  stopped.


void perror(const char *s);
  The routine perror() produces a message on the standard error output,
  describing the last error encountered during a call to a system or
  library function. The argument string s is printed first, then a colon
  and a blank, then the message and a new-line.


int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
  The functions in the printf family produce output according to a format
  as described below. sprintf, snprintf, vsprintf and vsnprintf write to
  the character string str.  Upon successful return, these functions return
  the number of characters printed (not including the trailing '\0' used to
  end output to strings). The functions snprintf and vsnprintf do not write
  more than size bytes (including the trailing '\0'). If the output was
  truncated due to this limit then the return value is the  number of
  characters (not including the trailing '\0') which would have been
  written to the final string if enough space had been available. Thus, a
  return  value of size or more means that the output was truncated. (See
  also below under NOTES.) If an output error is encountered, a negative
  value is returned.
```

2. Memória Virtual

Considere as instrucções de *input* e as operações de *output* de acesso à memória definidas no trabalho prático.

```
    Instrucções de input           Operações de output
--------------------------     --------------------------
R end                          load  M(mold) P(pag)
W end val                      store M(mold) P(pag)
                               read  M(mold) D(des) V(val)
                               write M(mold) D(des) V(val)
```

Considere ainda um sistema de gestão de memória virtual com paginação de um nível, e que dispõe de 5120 *bytes* de memória virtual e 3 molduras de página com 1024 *bytes* cada. O sistema funciona com uma interrupção de relógio a cada 2 instruções (a primeira instrução é a número 1) e utiliza o algoritmo NFU com contador de 8 bits para substituição de páginas. Antes de executar a décima instrução, o sistema apresenta a tabela de páginas com a seguinte configuração:

```
------------------------------
Page | Mold Load Last Cont R M
------------------------------
   0 |    -    -    -    - - -
   1 |    0    1    -   80 1 1
   2 |    1    2    -   80 0 0
   3 |    -    -    -    - - -
   4 |    2    3    -  160 0 1
------------------------------
```

a) Indique quais seriam as operações de *output* de acesso à memória realizadas pelo sistema para executar o conjunto de instruções de *input* que se segue:

```
W 1000 10
W 2000 20
R 1000
W 3000 30
```

b) Considere a estrutura de dados a seguir apresentada como forma de manter a informação sobre as entradas na tabela de páginas.

```
typedef enum {OFF=0, ON=1} bit;

typedef struct table_entry {
  unsigned int Mold;
  unsigned int Load;
  unsigned int Last;
  unsigned char Cont;
  bit R;
  bit M;
} TabEnt;
```

Escreva a função `'int nfu_8bits_counter(int n_pags, TabEnt *t_pags)'` que implementa o algoritmo de substituicão de páginas NFU com contador de 8 bits. A função recebe como argumentos o número de páginas (`n_pags`) e o apontador para a tabela de páginas (`t_pags`) e deverá devolver o número da página a ser substituída.

3. Sistema de Ficheiros

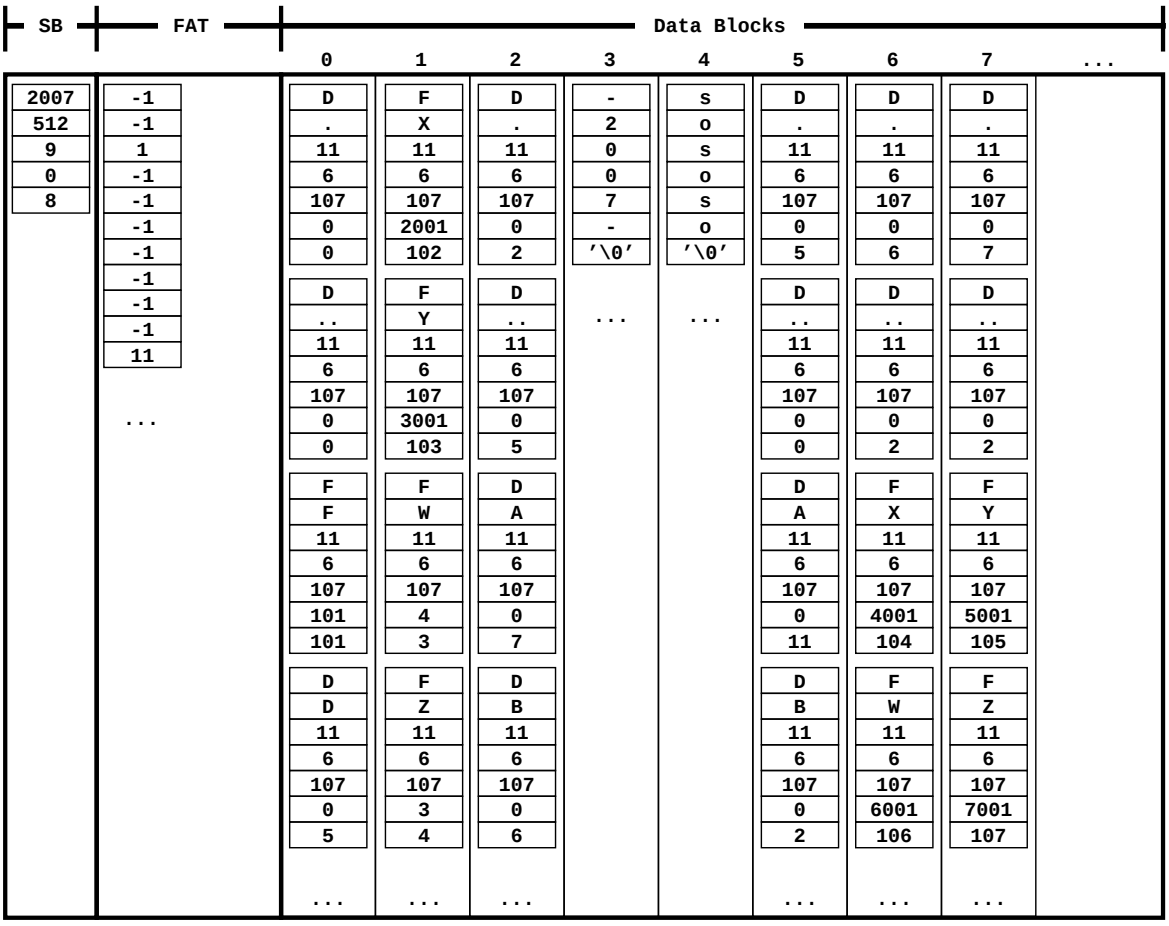
A figura abaixo representa parte de um sistema de ficheiros organizado segundo as estruturas de dados utilizadas no trabalho prático (considere inteiros de 4 *bytes*).

```
#define TYPE_DIR  'D'
#define TYPE_FILE 'F'
#define TYPE_FREE '-'

typedef struct superblock_entry {
  int check_number;
  int block_size;
  int fat_type;
  int root_block;
  int free_block;
} superblock;
```

```
#define MAX_NAME_LENGHT 20

typedef struct directory_entry {
  char type;
  char name[MAX_NAME_LENGHT];
  unsigned char day;
  unsigned char month;
  unsigned char year;
  int size;
  int first_block;
} dir_entry;
```



Considerando que o bloco de dados 2 é o primeiro bloco do directório corrente e que este possui 32 entradas, incluindo as entradas '.' e '..', indique justificando:

a) o número de blocos ocupado pela FAT e o número de blocos de dados em utilização.

b) o resultado da execução do comando *'pwd'*.

c) o resultado da execução do comando *'cat Z'*.

e) o resultado da execução do comando *'cp X A'* (assuma que a entrada *X* não está em *A*).

d) o resultado da execução do comando *'mkdir D'* (assuma que a entrada *D* não está no directório corrente).