Seminar 1: Shell Commands with Read and Write
Sistemes Operatius

1 Introduction

The objective of this seminar is to code small C programs that can be seen as simple shell commands. We are going to practice with the system calls "read", "write", "open". The final objective is to understand how a Shell Interpreter works.

2 System calls: open, read and write

The system call open returns a file descriptor of the opened file (an integer). Has as parameters: the name of the file, the flags (specifying creation, reading or writing needs), permissions (referring to owner, group and system access permissions). For permissions, we are always going to use 644 meaning: the owner can:read/write, the group:read, others:read).

```
int open(char* name, O_CREAT | O_RDWR, 0644);
```

Files are sequences of bytes stored in disk. Files are always binary (meaning that they store bytes of any kind). But it can happen that the set of bytes that the file contains all belong to the character set plus spaces and new line, in that case the file is a text file and can be directly printed into the screen.

Here you have the signatures of the read and write system calls:

```
int read(int fd, void *buf, int nbyte);
int write(int fd, void *buf, int nbyte);
```

The first parameter is an integer that can be 0 for the standard input (assigned to the keyboard by default), 1 for the standard output (assigned to the screen by default), a file descriptor of an opened file (using the system call open).

2.1 Standard C lib functions: printf, sprintf, sscanf, scanf

You can use the standard C lib functions for printing with format (printf), formatting a text buffer (sprintf), getting data from standard input (scanf), and parsing ints, floats and strings and save them into its correspondent data types (sscanf).

```
int int_var;    float float_var;   char str[10], char_var,buff[80];          // Declarations

printf("%d %f %s %c", int_var, float_var, str, char_var);
sprintf(buff,"%d %f %s %c", int_var, float_var,str, char_var);
scanf("%d", &int_var);                                    // getting an int from keyboard
sscanf(buff,"%d %f %s %c", &int_var, &float_var, str, &char_var);
```

2.2 Exercises: Open/Read/Write

1. Write a call to the sys-call "read" and the variables you need to read one byte from the keyboard and store it in a variable of type char.
2. Discuss how you can declare 100 integers and how you can know the size in bytes of the declared data.
3. Using the sys-call "read" 100 integers from a binary file "nums.dat" and store them in memory.
4. In the previous program check if everything went correctly and all the information could be read.
5. Do a sys-call "read" that reads a string from the command line and checks if the length of the string is greater than 0.
6. Write of a call to "write" that writes a character 'a' to the screen.

7. Reads bytes one by one from the standard input and write them into the standard output.

8. Imagine the previous program has been compiled into an executable named "mycp": how can you call it from the command line to copy two files "in.txt" to "out.txt"?

9. Read an integer from keyboard using scanf function and print the result of multiplying it by 2.

10. Read an integer of maximum 8 digits from keyboard (using sys-call read) and print the result of multiplying it by 2. Its not possible to read binary data from keyboard (standard input), so first put it in a char buffer and then use sscanf for conversion.

11. Read a binary file named nums.dat containing 10 integers and print each integer multiplying it by 2.

12. Write a command shell program that counts the number of characters 'a' in the standard input and writes the result in the standard output.

13. Discuss and explain what the following code does. Pay attention to the parameters: a file descriptor (or standard input 0), a pointer to an array of characters, the max length to be read, a pointer to the character that was encountered last (acts as an output parameter).

```
int read_split( int fin, char* buff, int maxlen, char* ch_end ) {
  int i = 0, oneread = 1;
  char c = '.';
  while(c != ' ' && c != '\n' && oneread == 1 && i < maxlen) {
    oneread = read(fin, &c, 1);
    if(c != ' ' && c != '\n' && oneread == 1) {
      buff[i] = c;
      i++;
    }
  }
  *ch_end = c;
  if(i < maxlen) buff[i] = '\0';
  return i;
}
```

14. Read a text file (call it nums.txt) using read_split (defined in ex.13) which has 10 integers written in it, separated by spaces and print each integer multiplying it by 2.

2.3 Delivery Exercises

1. Program a cat command shell that prints to the screen the content of a text file (given as first argument) using read_split (defined in ex.13) and in addition indicates the total number of words and lines.

2. Use read_split (ex.13) to read a file with a word and an integer in each line (having a maximum of 1000 of these pairs). Put all words in an array: char* words[] and print them. Use malloc (allocate memory in bytes) and memcpy (copies n bytes of memory from src to dst, both void pointers):

```
void *malloc(size_t size)
void *memcpy(void *dest, const void * src, int n);
```

Example words.txt:

```
hello 1333
to 123
airplaine 10
snowflake 4
sleep 100
```

Desired result:

```
hello to airplaine snowflke sleep
```