RATIONALE FOR THE STR5 FUNCTIONS

Introduction

¤ Few functions of the *C standard library* cumulate as many failures as the functions dedicated to the copy or concatenation of character strings (**strcpy**/**strcat**, **strncpy**/**strncat**):

+ *Careless design* (ex: useless return value, insufficient number of parameters)

+ *Inconsistent behaviours* (ex: **strncpy** and **strncat** have opposite behaviours)

+ *Ambiguity* (ex: the third parameter of **strncpy**/**strncat** interpreted by the programmer as a length or as a size)

+ *Week robustness* (ex: no error checking with **strcpy**/**strcat**, incomplete error checking with **strncpy**/**strncat**)

¤ *C Standard Library* is not solely used by experts but also by students, teachers and simple programmers who need reliable functions

¤ Sometimes, even experienced developers make mistakes when they use these functions

¤ A paradoxical situation:

+ Problems caused by string manipulations are well-known for decades: *bad-formed strings* (i.e. *not null terminated strings*), *buffer overflows* and *undetected truncated strings*

+ For decades, no modification was made on the *C standard library* to improve string copying/concatenating.

Technical and psychological sources of problems

char * **strncpy(** char * *dst*, const char * *src*, size_t *n* **);**
char * **strncat(** char * *dst*, const char * *src*, size_t *n* **);**

Technical reasons

¤ Insufficient number of parameters

+ The three parameters transmitted to **strncpy**/**strncat** do not make it possible to delegate error checking to them

¤ Difficult to use

+ Creating robust code using these functions quickly becomes difficult because the taking into account of the various possible error cases is left to the programmer and error checking is error prone

+ Because error checking is tricky, sloppy programmers prefer to ignore it

Psychological reasons

¤ A stubborn refusal to consider string copy/concatenation to be complex operations that need as much care as system call or IO function

+ After the execution of the **fopen** function, the *return value/error condition* (**errno**) couple provides meaningful information to the developer. Consequently, writings about file operation problems are not abundant

+ At the opposite, the careless design of the string functions has led to a rich literature

Properties of the Str5 functions

int **strtcpy(** char * *dst*, size_t *dstsize*, const char * *src* **);**
int **str5cpy(** char * *dst*, size_t *dstsize*, const char * *src*, size_t *nb*, size_t *mode* **);**

[cf. *str5cpy.txt*]

int **strtcat(** char * *dst*, size_t *dstsize*, const char * *src* **);**
int **str5cat(** char * *dst*, size_t *dstsize*, const char * *src*, size_t *nb*, size_t *mode* **);**

[cf. *str5cat.txt*]

¤ They provide a major added value

+ They copy/concatenate a *full string* or a *substring*

+ All error checking is delegated to the functions (enough information is given to the functions to make their job)

+ They never create *bad-formed strings* (programmers do not have to manage the terminating null byte)

¤ They respect sound design principles

+ ***Clarity***: the number of details to take into account to understand and use a component must be as low as possible

- Before execution, developers have to write the parameters values in the natural order (destinations parameters, source parameters and mode), specifying what they want

- After execution, developers only need to consult the *return value*. After success, the content of the destination array is always a *well-formed string* (*null terminated string*)
- *Return value* and *result* are clearly distinguished

+ ***Separation of concerns***: there is no responsibility sharing between two components
No external component to the function participates in error checking

+ ***Consistency***: in similar situations, behaviours are similar
Whatever *str5* functions, the same error condition produces the same *return value*

+ ***Neutrality***: after an error, input data are not modified
After an error, the functions don't modify the content of the destination array

+ ***Robustness***: The function performs correctly under normal and unexpected conditions
Even with insane input data, the functions are able to give a meaningful return value

¤ They are easy to use

+ Detection of an error condition: `if ( str5_function(…) < 0 )`

+ Creation of specific tools:

```
#define strntcpy(dst,dstsize,src) str5cpy(dst,dstsize,src,dstsize,NOTRUNC)
if ( (r=strntcpy(dst,dstsize,src)) < 0 )   /* a not allowed truncation ? */
```

+ Detection of a valid string copy/concatenation:

```
if ( str5_function(dst,…) >= 0 )
{ /* dst is a well-formed string */ }
```

¤ They are highly portable

+ They are conforming to the C89 and upper Standards

Some flaws of other proposed solutions

¤ **Strlcpy/strlcat**

size_t **strlcpy(** char * *dst*, const char * *src*, size_t *dstsize*);
size_t **strlcat(** char * *dst*, const char * *src*, size_t *dstsize*);

+ Weak *robustness*:
- When the destination pointer or source pointer is a NULL pointer, the function crashes
- If **strlcat** scans *dstsize* characters without finding a null character, the destination string will not be null terminated

+ *Inconsistent* behaviour: the functions return the total length of the string they tried to create, but for **strlcpy** it's the length of *src* and for **strlcat** it's the initial length of *dst* plus the length of *src*

+ It doesn't respect the principle of *separation of concerns*: error checking is shared by the function and the developer (truncation detection)

¤ **Snprintf**

It's a multi-purpose function:

int **snprintf**(char * *dst*, size_t *n*, const char * *format*, ... **);**

To copy the string *src* in the destination array pointed to by *dst*:

snprintf(*dst*, *dstsize*, *"%s"*, *src* **);**

+ Weak *robustness*: when the destination pointer is a NULL pointer, the function crashes

+ *Inconsistent* behaviour: when the destination pointer is a NULL pointer, the function crashes; when the source pointer is a NULL pointer, the function modifies the destination array with an artificial string: **'(null)'**

+ It does not respect the principle of *separation of concerns*: error checking is shared by the function and the developer (truncation detection)

¤ **Strncpy_s/strncat_s**

errno_t **strncpy_s(** char * *ds*t, size_t *dstsize*, const char * *src*, size_t *nb* **);**
errno_t **strncat_s(** char * *ds*t, size_t *dstsize*, const char * *src*, size_t *nb* **);**

+ The principle of *separation of concerns* is broken: two components share error processing (the function itself and the runtime-constraint handler)

+ They do not respect the principle of *neutrality*: after error, the content of the destination array may be modified

+ Weak *usability* and *portability*: only a few platforms support these functions (C11 Standard conformance, integration into a heavyweight library)

+ Weak *readability*: using a runtime-constraint handler breaks code readability. For example, at the end of section K.3.7.1.4 of the *C11 Annex K* document there is this piece of code:

```
        char src2[7] = {'g', 'o', 'o', 'd', 'b', 'y', 'e'};
        char dst2[5] ;
        int r2 ;
    r2 = strncpy_s(dst2,5,src2,7);
```

and the associated comment says that a nonzero value is assigned to *r2* and the sequence **\0** to *dst2*.
In fact, it is wrong if a runtime-constraint handler is called and does not return.