Atomik Kernel API

Standard library functions

As Atomik is the lowest level multi-platform software in the operating system, it has to define its own types and libraries from scratch. Although many functions and types can be found in the user-level standard C library, most of them are particularized to the microkernel context.

The following document describes the standard library functions that can be used by microkernel code.

Structures

```
#include <misc/vkprintf.h>

struct vkprintf_stream
{
  memsize_t counter;
  void *opaque;

  int (*putchar) (struct vkprintf_stream *, char);
  int (*puts) (struct vkprintf_stream *, const char *);
};
```

Describes the output stream operations used by `vkputchar`, `vkputs` and `vkprintf` for abstract formatted output. `counter` contains the number of bytes correctly sent to the stream and `opaque` is a pointer with private data to be used by the implementation.

The output stream operations are implemented by the functions pointed by `putchar` and `puts`. The only mandatory function to be defined is putchar, which receives the stream (and thus, its private data) and the character to be sent and returns 0 if the operation was performed correctly or non-zero on error.

The pointer `puts` is optional and, if defined, it will be used to output strings instead of repeatedly calling `putchar`. The function shall return the number of bytes correctly sent to the stream. If the implementation doesn't provide a puts operation, it must be set to `NULL`.

Functions

```
#include <string.h>

size_t strlen (const char *s);
```

Returns the number of bytes starting from `s` before the first null character (`'\0'`) is found. Equivalent to the `strlen` function found in the standard userspace C library.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

int strcmp (const char *a, const char *b);
```

Compares two strings pointed by *a* and *b* until the first null character is found, returning 0 if both strings are equal, 1 if *b* has a character whose ASCII code is bigger than the character in the same position in *a* and -1 otherwise.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

int strncmp (const char *a, const char *b, size_t n);
```

Same as `strncmp`, but limits the search of the first null byte to `n` characters max.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

char* strcpy (char *dest, const char *orig);
```

Copies the string pointed by `orig` to the location pointed by `dest` until the first null byte in `orig` is found, which is copied aswell. **This function is inherently dangerous as it cannot guess the size of the destination buffer `dest` and can lead to buffer overflows**.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

char* strncpy (char *dest, const char *orig, size_t n);
```

Same as `strcpy`, but limits the copy up to `n` bytes. Note that if the null byte is not found within the first `n` bytes, the resulting string in `dest` won't be properly finished. The programmer must take this situation into account when treating the stored string as an ASCIIZ string.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

char* strchr (const char *dest, int c);
```

Returns the pointer of the first occurrence of character `c` in the string pointed by `dest` until the first null byte is found.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

int memcmp (const char *a, const char *b, size_t n);
```

Compares the first n bytes of memory regions pointed by *a* and b, returning 0 if both regions are equal, 1 if a byte found in b is bigger in absolute value than the byte in the same position in *a*, and -1 otherwise.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

void* memcpy (void *dest, const void *orig, size_t n);
```

Copies the first n bytes pointed by `orig` to the memory location pointed by `dest`, and returns the pointer `dest`. This function doesn't support overlapping.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

void* memset (void *dest, int c, size_t n);
```

Fills the n first bytes pointed by `dest` with the byte `c`, and returns the pointer `dest`. Useful when initializing buffers.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <string.h>

void* memcpy (void *dest, const void *orig, size_t n);
```

Copies the first `n` bytes pointed by `orig` to the memory location pointed by `dest`, and returns the pointer `dest`.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <misc/vkprintf.h>

void vkputchar (struct vkprintf_stream *stream, char c);
```

Sends the character `c` to the stream described by `stream`. This function will use `stream->putchar` to send the character and increment the byte counter if operation was performed correctly.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <misc/vkprintf.h>

void vkputs (struct vkprintf_stream *stream, const char *s);
```

Sends the string `s` to the stream described by `stream`. This function will try to use `stream->puts` if defined or `stream->putchar` repeatedly otherwise. The byte counter is incremented accordingly.

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes

```
#include <misc/vkprintf.h>

void vkprintf (struct vkprintf_stream *stream, const char *fmt,
...);
```

Formatted output to the stream described by `stream`. This function expect a `printf`-like format string in `fmt`, taking arguments from the variable argument list as usual. The supported formats by vkprintf are:

Format	Argument type	Description	Examples
%d	**int**	Decimal representation of argument	-263 0 22
%h	**uint32_t**	Decimal representation of argument as a memory size with a unit suffix. The value is divided to be fit the biggest unit represented (for instance, 65536 will be represented as 64K)	0b 450M 23K 2G
%H	**uint32_t**	Same as **%h**, but units are shown as a string instead of a single-character suffix.	0 bytes 450 MiB 23 KiB 2 GiB
%b	**unsigned int**	Hexadecimal representation of the least significant 8 bits of the argument, lower case.	00 7f 22 ac
%B	**unsigned int**	Hexadecimal representation of the least significant 8 bits of the argument, upper case.	00 7F 22 AC
%x	**unsigned int**	Hexadecimal representation of the argument, lower case.	0 f0802a7f 22 ffff9eac
%X	**unsigned int**	Hexadecimal representation of the argument, upper case.	0 F0802A7F 22 FFFF9EAC
%w	**unsigned int**	Hexadecimal representation of the least significant 32 bits of the argument, lower case with a separator.	0000:0000 f080:2a7f 0000:0022 ffff:9eac
%W	**unsigned int**	Hexadecimal representation of the least	0000:0000

		significant 32 bits of the argument, lower case with a separator.	F080:2A7F 0000:0022 FFFF:9EAC
%o	unsigned int	Octal representation of the argument.	0 712 644 213
%y	unsigned int	Hexadecimal representation of the least significant 32 bits of the argument, lower case.	00000000 f0802a7f 00000022 ffff9eac
%Y	unsigned int	Hexadecimal representation of the least significant 32 bits of the argument, lower case.	00000000 F0802A7F 00000022 FFFF9EAC
%c	unsigned int	ASCII representation of the byte given as argument	c A 7 _
%C	uint32_t	Representation of CPU flags. Architecture dependant.	C--ZST--ONR----------I---------
%p	void *	Hexadecimal representation of a pointer	(null) 0x8048000 0xd0000000 0x7af
%s	char *	Representation of the string pointed by the argument	(any string is possible)
%%	(none)	Character '%', used to avoid conflicts with other format strings. Doesn't pop any argument from the argument list.	%

Reentrante : yes
Thread-safe : no
Interrupt-safe : yes