Improvements on EGOS-2000

Leo Liang <wl353@cornell.edu>
Supervising Faculty Member: Robbert van Renesse <rvr@cs.cornell.edu>

Summary

The project aimed to improve EGOS-2000 by incorporating some of the C standard libraries into the OS, specifically `math.h`, `stdlib.h`, `stdio.h`, `ctype.h`, and `string.h`. This allowed the operating system to support these standard library functions and made it easier for developers to write applications for EGOS-2000.

Current Issue

One of the limitations of EGOS-2000 is that the majority of the C standard libraries are missing in the project. This can make it difficult for developers to write applications for the operating system, as they may not have access to the standard functions that they are used to using. For example, if a developer wants to use the standard math library to perform trigonometric calculations, they may not be able to do so in EGOS-2000 without implementing the library themselves. This can be time-consuming and may result in code that is less efficient than if the standard library had been available. Additionally, the lack of standard libraries may limit the types of applications that can be developed for EGOS-2000, which could impact its adoption and popularity.

One seemingly obvious solution to this issue is to use the implementation provided by the RISC-V compiler directly in user applications. However, after experimenting with this idea, it is shown that this would not work well with EGOS-2000 due to two reasons. Firstly, the limitations of EGOS-2000 user application file size present a significant challenge to using the compiler-provided C standard libraries in user applications for EGOS-2000. Here is an example application that opens a file and then closes it:

```c
#define LIBC_STDIO // This is necessary to use the compiler-provided stdio.h.
#include "app.h"
#include <stdio.h>

int main(int argc, char** argv) {
	FILE *file = fopen("README", "r");
	fclose(file);
	int int_buffer;
	sscanf(argv[0], "%d", &int_buffer);
	printf("Integer I read: %d\r\n", int_buffer);
	return 0;
}
```

However, when compiling this application, one would get this error message: (tested on macOS)

```
/Users/test/riscv64-unknown-elf-gcc-8.3.0-2020.04.1-x86_64-apple-darwin/bin/../l
ib/gcc/riscv64-unknown-elf/8.3.0/../../../../riscv64-unknown-elf/bin/ld:
build/release/myapp.elf section `.text' will not fit in region `ram'
/Users/test/riscv64-unknown-elf-gcc-8.3.0-2020.04.1-x86_64-apple-darwin/bin/../l
ib/gcc/riscv64-unknown-elf/8.3.0/../../../../riscv64-unknown-elf/bin/ld: region
```

```
`ram' overflowed by 2232 bytes
```

The compiler complains that the application size is too big, and that is the result of all the helper functions in the compiler stdio implementation that need to be included. One way to resolve this is to increase the ram region in `app.lds`, but this would require significant changes to the OS as memory mapping would be modified from increased application size.

To address the size limit issue, the standard libraries must be part of the operating system itself as much as possible rather than being compiled into the executables of user applications. This approach would allow the standard libraries to be used without significantly increasing the size of individual executable files beyond the limit.

Another limitation is the missing compatibility layer between the C standard libraries and the operating system. Some of the functions provided in the C standard libraries are closely tied to the operating system, such as those in `stdio.h`. If the compiler-provided implementation of these functions, like "`fopen`" and "`fread`," is directly used in user applications, it will result in compile-time errors. Using the same example above, the compiler shows a second error message in addition to ram overflowing:

```
/Users/test/riscv64-unknown-elf-gcc-8.3.0-2020.04.1-x86_64-apple-darwin/bin/../l
ib/gcc/riscv64-unknown-elf/8.3.0/../../../../riscv64-unknown-elf/bin/ld:
/Users/test/riscv64-unknown-elf-gcc-8.3.0-2020.04.1-x86_64-apple-darwin/bin/../l
ib/gcc/riscv64-unknown-elf/8.3.0/../../../../riscv64-unknown-elf/lib/rv32i/ilp32
/libc.a(lib_a-openr.o): in function `.L0 ':
openr.c:(.text+0x28): undefined reference to `_open'
```

Here, the compiler shows that it is missing the implementation of "`_open`". This is OS-dependent code that the compiler C standard library needs to know how to open a file. To address this, a compatibility layer needs to be implemented that provides a bridge between the C standard libraries and the operating system. This layer would allow developers to use the standard libraries in their applications without running into compatibility issues with the operating system.

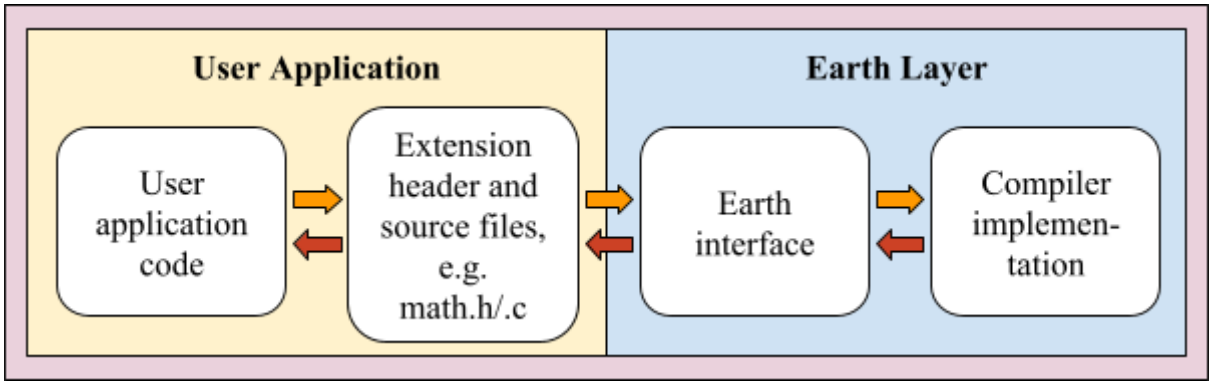
Solution Proposed

The proposed solution consists of two parts: OS-independent functions and OS-dependent ones. OS-independent functions are standard C library functions that are not closely tied to the operating system and do not require a compatibility layer to work in user applications. Examples of such functions include the ones in `stdlib.h` and `math.h`. While these functions do not concern the compatibility layer issue mentioned previously, using them in user applications would still cause the size of the resulting executable files to exceed the size limit of the operating system, as the compiler implementation can be quite heavy.

To address this issue, the proposed solution is to include the implementations of these OS-independent functions in the earth layer. The earth layer is the layer EGOS-2000 that sits between the hardware and the grass layer. It provides a much more generous size limit that can effectively hold all the implementation. By including the implementation of these functions in the earth layer, all user applications can reference the same code, eliminating redundancy and promoting code reuse. It

ensures that all applications can use the standard libraries without increasing their size beyond the limit.

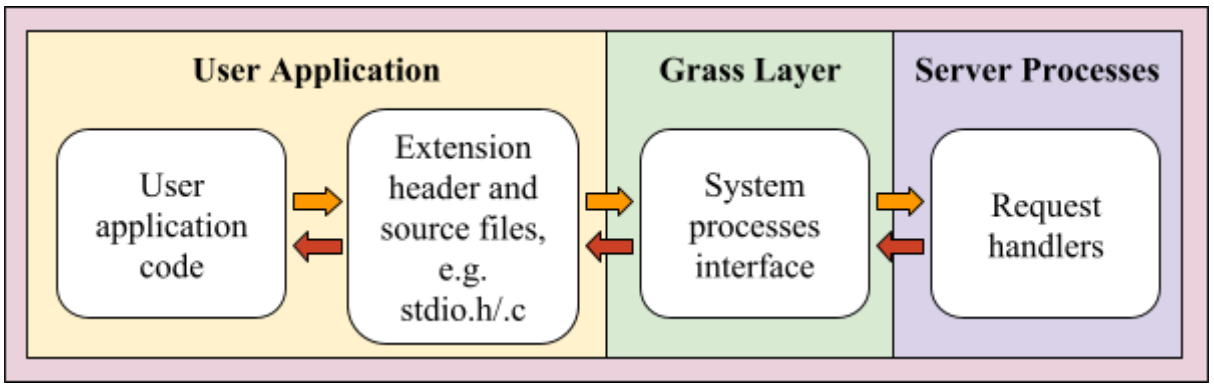
Custom header and source files are included to sit between the user application and the earth implementations. The chart below shows the flow of how these function calls work:



Note that the header and source files here are not the ones from the compiler. Instead, they are custom code that provides the familiar interface of the C standard library but actually serve as links to the earth interface. At its foundation, the implementations of these functions are provided by the compiler. Because compiler implementations are well-tested, this minimizes the probability of bugs. It also prevents unnecessary reimplementation of C standard libraries.

OS-dependent functions, on the other hand, rely on code that is specific to EGOS-2000. An example of such a function is "`fopen`" in `stdio.h`. In order to implement this in EGOS-2000, one needs to make a directory request to the grass layer and be re-routed to the sys_dir server process. For this reason. these functions cannot be included in the earth layer like the OS-independent ones.

To address this issue, the OS-dependent functions are reimplemented in the header and source files. The custom implementation makes requests to the grass layer so that the handlers in server processes can fulfill those requests and return the result. The chart below shows the flow of how it would work:



Because the custom implementations are stored in the header and source files that would be compiled into the user application, this would mean that there will be a size increase on the executable. However, the reimplementation is much more lightweight than the compiler ones, thus the size impact would be minimal while retaining behaviors close to the compiler implementation.

List of implemented library functions

Header file	Function	OS dependent		Header file	Function	OS dependent
ctype.h	isalnum	NO		stdio.h	printf	YES
	isalpha	NO			vprintf	YES
	iscntrl	NO			sprintf	YES
	isdigit	NO			vsprintf	YES
	isgraph	NO			fprintf	YES
	islower	NO			vfprintf	YES
	isprint	NO			scanf	YES
	ispunct	NO			vscanf	YES
	isspace	NO			sscanf	YES
	isupper	NO			vsscanf	YES
	isxdigit	NO			fscanf	YES
	tolower	NO			vscanf	YES
	toupper	NO			fopen	YES
stdlib.h	abs	NO			freopen	YES
	div	NO			fread	YES
	labs	NO			fwrite	YES
	ldiv	NO			fclose	YES
	rand	NO			fflush	YES
	srand	NO			fgetpos	YES
	atof	NO			fsetpos	YES
	atoi	NO			ftell	YES
	atol	NO			fseek	YES
	strtod	NO			rewind	YES
	strtol	NO			getchar	YES
	strtoll	NO			gets	YES
	strtoul	NO			putchar	YES
	bsearch	NO			puts	YES

	qsort	NO			clearerr	YES
	mblen	NO			feof	YES
	mbtowc	NO			ferror	YES
	wctomb	NO			perror	YES
	mbstowc	NO		string.h	memcmp	NO
	wcstombs	NO			strcmp	NO
	calloc	NO			strcoll	NO
	malloc	NO			strncmp	NO
	realloc	NO			strxfrm	NO
math.h	fabs	NO			strcat	NO
	ceil	NO			strncat	NO
	floor	NO			memcpy	NO
	fmod	NO			memmove	NO
	exp	NO			strcpy	NO
	frexp	NO			strncpy	NO
	ldexp	NO			memchr	NO
	log	NO			strchr	NO
	log10	NO			strcspn	NO
	modf	NO			strpbrk	NO
	pow	NO			strrchr	NO
	sqrt	NO			strspn	NO
	acos	NO			strstr	NO
	asin	NO			strtok	NO
	atan	NO			memset	NO
	atan2	NO			strerror	NO
	cos	NO			strlen	NO
	cosh	NO				
	sin	NO				
	sinh	NO				
	tanh	NO				

Additional changes

- The size of the earth layer interface is slightly increased to fit the additional functions.
- Another structure called "`extension`" is added alongside "`earth`" and
- cd and sys_dir have been modified to allow better directory lookup. For example, the following commands, which were previously not permitted, are now possible:

```
> cd ../..
> cat home/yunhao/README
```

- "`ed`" from EGOS has been ported to EGOS-2000 for file editing.

Limitations

The proposed solution has a series of limitations:

- The implementation of the C standard library functions in EGOS-2000 is not comprehensive, meaning that not all functions are included. This poses a challenge for developers who require the use of functions that are not implemented, as code that uses those functions will not work as expected. In such cases, developers must either find an alternative solution or manually update the code in multiple places to allow the use of another C standard library function that was not previously included. This can be a time-consuming and error-prone process, especially for complex codebases.
- To incorporate a C standard library function that was not previously included, updating the code in multiple places is necessary. This could take significant effort if one wants to allow the use of a significant amount of library functions.
- Renaming, moving, copying, or creating files are not implemented in this project as the OS does not currently support it natively.
- OS-dependent code is reimplemented in the header and source files to work in user applications. However, these implementations have not gone through the same rigorous testing as the compiler implementation and may contain bugs or exhibit behavior that is not consistent with C standards. Developers must be aware of this when using OS-dependent code and ensure that their applications are tested thoroughly to identify and fix any issues.
- Modifying the earth layer in EGOS-2000 means that the ARTY board needs to be reflashed. This process can be time-consuming and inconvenient, and switching between this extended EGOS-2000 and the standard EGOS-2000 is somewhat cumbersome.
- Memory leak exists that would cause commands that happen after one to fail.