Porting compression services to Capsicum

Daniel Peyrolon
dpl@FreeBSD.org
Mentor: Brooks Davis
brooks@FreeBSD.org

FreeBSD

FreeBSD Developer Summit
Hilton Conference Centre
St. Julian's, Malta
September 26 – 28, 2013

The GSoC proposal

Porting the following software to Capsicum:

- bzip2(1)
- xz(1)
- zlib(3)
- libavcodec

Porting applications

The way I used Capsicum.

- Used the fork() style.
- I didn't use Casper.
- Child is in capability mode.
- Changed all the functions using paths to fds. (stat - fstat)
- I wanted to keep the portability of the application.

Porting bzip2(1)

It was not that hard, but I had this issues:

- Lack of experience.
- Tried to send fds to the child through UNIX sockets.
- That proved to be a bad choice.

Porting bzip2(1)

This was how I did it.

- Put the file descriptors bzip2 was working with on global scope.
- I needed a fd of the directory where the current file was located (unlink).
- Just a matter of writing the code to limit all the fds, and enter capability mode.
- The parent just waits until the sandboxed child finishes.

Porting bzip2(1)

How I kept portability:

```
#   ifdef __FreeBSD__
#      include <osreldate.h>
#      if __FreeBSD_version >= 900041
#         define CAPSICUM
#         include <libgen.h>
#         include <sys/capability.h>
#         include <sys/wait.h>
#         include <sys/un.h>
#      endif /* __FreeBSD_version >= 900041 */
#   endif /* __FreeBSD__ */
```

Porting bzip2(1)

Typical fork() usage:

```
#if defined(CAPSICUM)
        if ( (forkpid = fork()) == -1 ){
            error();
        } else if ( forkpid != 0) {
            /* Let the children compress */
            wait(NULL);
        } else if (forkpid == 0){
            capsicum_enter();
#endif /* CAPSICUM */
            stuff();
#if defined(CAPSICUM)
            exit(0);
        }
#endif /* CAPSICUM */
```

Porting xz(1)

Well, that was harder...

- The fds are stored in a struct file_pair.
- The function where the work is done uses the path of the files.
- Opens the files, and works on them.

Porting xz(1)

File handling code:

```
void (*run)(const char *filename) = opt_mode == MODE_LIST
    ? &list_file : &coder_run;

for (size_t i = 0; i < args.arg_count && !user_abort; ++i) {
    if (strcmp("-", args.arg_names[i]) == 0)
        //Processing from stdin to stdout.

    run(args.arg_names[i]);
}
```

Porting xz(1)

New way of opening the files:

- I kept an array of malloc'ed file_pair *.
- All the files are opened before doing the actual work.
- run() uses now file_pair * instead of paths.

A Frame with Table

Benchmarking results for bzip2 and xz.

Filesize	bzip2	cbzip2	xz	cxz
1kb	0.01	0.01	0.11	0.11
10kb	0.01	0.01	0.11	0.11
100kb	0.03001	0.03002	0.13672	0.13071
1mb	0.27016	0.27029	0.41307	0.41234
1kb	0	0	0.01	0.01
10kb	0	0	0.01	0.01
100kb	0.01	0.01	0.01	0.01
1mb	0.10051	0.10213	0.01033	0.0104

- 1000 tests.
- We couldn't quantify the overhead on my machine.

FreeBSD

Porting a library

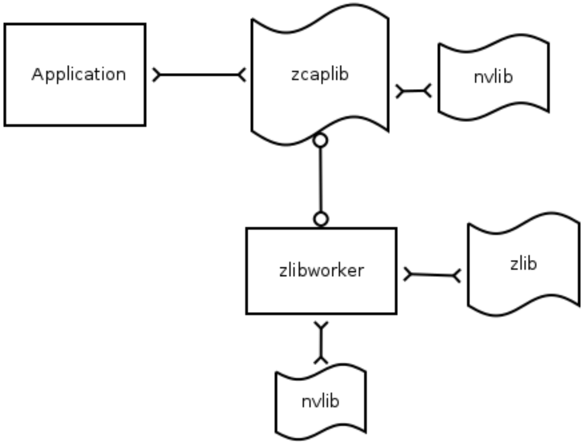
Everything changes.

- You can't wait to a child process (that's what we thought).
- Use pdfork(2).
- This child process will have to get the data from somewhere. (unless you're inheriting everything).

zcaplib

- It can be linked instead of zlib.
- zcaplib is just a giant wrapper trying to work.
- Uses libnv.
- It executes zlibworker. Which is listening for commands.
- At most I'm sending and receiving 5kb of data.
- I stored the sandboxes on a SLIST (queue(3)).

zcaplib's design



zcaplib

Typical function in zcaplib.

```
extern const char * zcapcmd_gzerror();
const char * ZEXPORT gzerror(file, errnum)
    gzFile file;
    int *errnum;
{
    return zcapcmd_gzerror(file, errnum);
}
```

zcaplib

Typical nvlist usage in zlibworker.

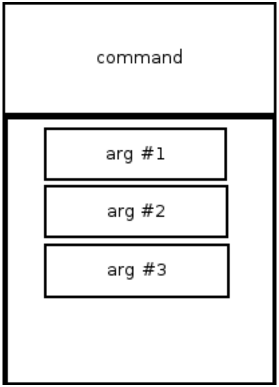
```
initNvl();

nvlist_add_number(nvl,"command",ZCAPCMD_GZERROR;
nvlist_add_binary(args,"file",file,gzsize);
nvlist_add_nvlist(nvl,"args",args);

result=sendCommand(nvl,file);

ptr=nvlist_get_string(result,"result");
*errnum=nvlist_get_number(result,"zerrno");
```

zcaplib's design



Using one or many sandboxes

- One sandbox - We do care about sending the structs from application to sandbox.
- Many sandboxes - We only send it once, and the application should forget about it.

Conclusions

- Porting an application to Capsicum is easy.
- Porting a library to Capsicum is hard.
- It's possible to write a tool that automates most of the work.
- Overload Casper with features?

Thank you all for your attention!

Questions?