In this document, we'll port an existing app to MutekH. We'll choose a standard publicly available application: `bc`.

bc is a command-line arbitrary precision calculator well known in the unix world. It uses standard APIs, makes reasonable use of I/O, and has a quite verifiable result.

Getting the source tree

Let's begin getting the source, and unpack it:

```
$ wget http://ftp.gnu.org/pub/gnu/bc/bc-1.06.tar.gz
$ tar xzvf bc-1.06.tar.gz
```

Creating our source tree

Now create another directory, and extract the meaningful sources

```
$ mkdir mutekh_port
$ cd mutekh_port
$ mkdir bc lib h sys
$ cp ../bc-1.06/bc/*.[ch] bc/
$ cp ../bc-1.06/lib/*.[ch] lib/
$ cp ../bc-1.06/h/*.h h/
```

Let's create the Makefiles:

- `bc/Makefile`

  ```
  objs = bc.o execute.o global.o load.o main.o scan.o storage.o util.o

  DIR_CFLAGS=-I$(srcdir)/../lib -I$(srcdir)/../h -I$(srcdir)/.. -U__MUTEK__
  scan.o_CFLAGS=-DEINTR=-1024
  ```

- `lib/Makefile`

  ```
  objs = getopt.o getopt1.o number.o glue.o

  DIR_CFLAGS=-I$(srcdir)/../lib -I$(srcdir)/../h -I$(srcdir)/.. -U__MUTEK__ -DHAVE_CONFIG_H
  ```

- `Makefile`

  ```
  subdirs = bc lib
  ```

Providing glue code

Now let's create some glue code:

- As this is userland-specific, MutekH does not define
  - `getenv()`
  - `signal()`
  - `sys/types.h`
- As it is unix and user-specific, MutekH starts with `app_start()`, not `main()`. We need a wrapper that calls `main()`.

  We will have to provide them.

Add a `lib/glue.c` file:

```
#include <hexo/types.h>

char *getenv(const char *env)
{
    return NULL;
}

int main(int, char**);

/**
  Wrap between app_start() and main() entry-points.
  Provide command-line arguments, and initialize the
  terminal in order to enable local echo.
 */
void app_start()
{
    printk("\x1b[12l\x1b[20h");
    char *argv[] = {"bc", "-l", NULL};
    main(sizeof(argv)/sizeof(char*)-1, argv);
}
```

Add a `signal.h` file:

```
#define SIGINT 0

static inline int signal(int sig, void *handler)
{
        return -1;
}
```

Create a `sys` directory, and a `sys/types.h` file containing:

```
#define isatty(x) 1
#define fileno(x) 0
#define fopen(x,y) NULL

#include <hexo/types.h>
```

Creating the config.h

As many autoconf-based programs, bc relies on a `config.h` file generated by the build system.

For MutekH, we will create this config.h from the config.h.in file.

Let's create a `config.h` file

```
#define HAVE_VPRINTF
#define STDC_HEADERS
#define DC_VERSION "1.06"
#define DC_COPYRIGHT "GNU"
#define BC_COPYRIGHT "GNU"
#define HAVE_LIMITS_H
#define HAVE_STDARG_H
#define HAVE_STDLIB_H
#define HAVE_STRING_H
#define HAVE_UNISTD_H
#define PACKAGE "bc for MutekH"
#define VERSION "1.06"
```

The MutekH configuration file

Finally, let's create a configuration file. For instance, with the tutorial platforms, we may use the following config:

```
# Application license
CONFIG_LICENSE_APP_GPL

# Mutek features
CONFIG_PTHREAD
CONFIG_MUTEK_CONSOLE

CONFIG_ARCH_SMP undefined

CONFIG_LIBC_STREAM
CONFIG_LIBC_STREAM_STD

CONFIG_HEXO_INTTYPES_DEPRECATED undefined

# New source code module to be compiled
%append MODULES bc:$(CONFIGPATH)

%include $(SRC_DIR)/examples/common/platforms.conf
%include $(SRC_DIR)/examples/common/build_options.conf
```

That's all, we finished porting our app !

Compiling and running

As an unix process

As seen in quick start, we may compile the kernel as an unix process:

```
user@host ? mutekh $ make CONF=/path/to/my/ported/app/config BUILD=emu-darwin-x86
?
BUILD DIR    .../mutekh/obj-kernel-emu
KERNEL       kernel-emu.out
```

And directly run it !

```
user@host ? mutekh $ ./kernel-emu.out
MutekH is alive.
CPU 0 is up and running.
bc for MutekH 1.06
GNU
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
123
123
123*123
15129
```

On SoCLib

As seen in soclib tutorial, we may compile the application

```
user@host ? mutekh $ make CONF=/path/to/my/ported/app/config BUILD=pf-tutorial:soclib-mips32el
...
    LD o    .../mutekh/kernel-soclib-mips32el.o
```

```
          LD out  .../mutekh/kernel-soclib-mips32el.out
```

Then run a simulator hosting the kernel

```
      user@host ? caba-vgmn-mutekh_kernel_tutorial $ ./simulation.x mips32el:1 /path/to/mutekh/kernel-
```

(When typing lines in the BC interpreter in a soclib terminal, you'll probably need to hit `Control-J` to insert a new line rather than `enter`.)