Connection information and credentials (SSID: *popcorn-tutorial*, password: *popcorn1234*)				
	Machine name	**IP address**	**User name**	**Password**
x86-64 laptop	laptop0	192.168.1.10	user0	user0
aarch64 board	potato0	192.168.1.20	user0	user0

Popcorn Unikernel/Heterogeneous EXecution Offloading (HEXO) Tutorial

---

1 Setup Description

The setup you will use is composed of a x86-64 machine (laptop0) connected to an ARM64 embedded board (potato0) via LAN. An account was created for you on both machines. The machines names, IP addresses, and the accounts names and passwords are indicated in the table at the top of this page. Note that the account name on the x86-64 machine is not necessarily the same as the one on the ARM64 board.

In this tutorial, we will create a unikernel, checkpoint it in the middle of its execution on the x86-64 machine, and resume it on the ARM64 board. To do so you only need a laptop with a SSH client. First, connect to the WiFi network with the following SSID: **popcorn-tutorial**. The password is: **popcorn1234**. Next, use SSH to login to your account on the x86-64 machine from your laptop:

```
you@your-laptop$ ssh user0@192.168.1.10
```

Use the password noted in the table at the top of this page. All the commands presented in the rest of this tutorial will be done from that account on the x86-64 machine.

2 Trying Out an Example: NPB Integer Sort (IS)

There is a set of example programs already instrumented for migration in the following directory: `~/popcorn-compiler/apps`. Let's go into NPB IS directory:

```
user0@laptop0$ cd ~/popcorn-compiler/apps/npb-is
```

The entire build and test process can be controlled with the Makefile. To build the unikernel image, simply use make:

```
user0@laptop0:~/popcorn-compiler/apps/npb-is$ make
# ...
[ld-aligned] prog_x86-64_aligned
[ld-aligned] prog_aarch64_aligned
```

Start the application on x86-64 and checkpoint its state after 5 seconds (you may want to tune this value according to the speed of the machine used so that IS is interrupted approximately in the middle of its execution):

```
user0@laptop0:~/popcorn-compiler/apps/npb-is$ MIGTEST=5 make
# IS starts to run ...
# ...
4
5
6
Uhyve: exiting
```

Transfer the checkpoint to the ARM64 board:

```
user0@laptop0:~/popcorn-compiler/apps/npb-is$ make transfer-full-checkpoint-to-arm
    rsync --no-whole-file *.bin stack.bin.* tls.bin.* user0@potato0:/home/user0
```

Resume on the ARM64 board:

```
user0@laptop0:~/popcorn-compiler/apps/npb-is$ RESUME=1 make test-arm
# IS resumes ...
7
8
9
10
# IS ends
```

3 Writing Your Own Program

Below is presented the source code of a simple program incrementing and printing a stack variable every second. A migration point can be added with a call to the `popcorn_check_migrate` function. When this function is executed, if the checkpointing signal was received by the hypervisor, the VM is checkpointed, and execution is stopped.

```c
#include <stdio.h>
#include <hermit/migration.h>   // This needs to be included in all programs

extern void sys_msleep(int ms);
int main(void) {
  int i=0;

  for(i=0; i<10; i++) {
    popcorn_check_migrate(); // checkpoint if checkpointing signal received
    printf("Iteration %d/10\n", i);
    sys_msleep(1000);
  }
}
```

To compile and test your program, use the template Makefile we provide. It is located in `~/popcorn-compiler/util/hermit/Makefile` and is described below.

Template Makefile. A template Makefile is provided for easy building and testing of unikernel images. It is pre-configured with the IP address of the x86-machine and board as well as the correct account names (see the `ARM_TARGET_IP` and `ARM_TARGET_USER` variables in the Makefile. Interesting `make` targets include:

- `make` builds the unikernel images;
- `make test-x86` and `make test-arm` start the execution on the corresponding machine. Use the `MIGTEST=X` environment variable to trigger checkpointing `X` seconds after launch by sending the checkpointing signal to the hypervisor. The checkpoint will be dumped on the machine executing the unikernel. Use the `RESUME=1` environment variable to resume from a (previously transferred) checkpoint);
- `make transfer-full-checkpoint-to-arm` and `make transfer-full-checkpoint-from-arm` transfer a (previously dumped) checkpoint from the x86-machine to the board and vice-versa.

Post-Copy State Transfer. In addition to full checkpoint mode, we also provide post-copy (on demand) state transfer. Below is a usage example. Start on x86-64 as follows:

```
user0@laptop0:~/test$ MIGTEST=5 FULL_CHKPT_SAVE=0 make test-x86
```

Once the hypervisor switched to page server mode, keep it running and open a new terminal on the x86-64 machine. Then transfer a minimal checkpoint to the board and resume as follows:

```
user0@laptop0:~/test$ make transfer-checkpoint-to-arm # faster than transfer full checkpoint
user0@laptop0:~/test$ RESUME=1 FULL_CHKPT_RESTORE=0 SERVER=192.168.1.10 make test-arm
```

More information on our website: `http://popcornlinux.org/index.php/hexo`