lcfos Documentation

Release 0.1

Windel Bouwman

September 18, 2016

Contents

CHAPTER 1

About

1.1 Project goals

- To write a microkernel based OS
- Write the kernel in the c3 language
- Create a c3 compiler in python

1.2 Directory structure

'kernel' contains the microkernel. 'python' contains the python utilities.

1.3 How to run this?

Install required software:

- python3.3
- (optional) pyqt5, pyqt4 or pyside

Checkout the code:

```
hg clone https://bitbucket.org/windel/lcfos
cd lcfos
```

Run some unit tests:

```
cd test
python3 run_tests.py
```

1.4 Weblinks

Docs are located here: http://lcfos.readthedocs.org/en/latest/

Sources are located here: https://bitbucket.org/windel/lcfos

here: http://hg.assembla.com/lcfOS/

and here: https://pikacode.com/windel/lcfos/

The project is contains tests which are run continuously at drone.io.

https://drone.io/bitbucket.org/windel/lcfos

Repository metrics:

http://www.ohloh.net/p/lcfos

Live demo is at redhat openshift:

http://lcfos-windel.rhcloud.com/

CHAPTER 2

OS

2.1 Implementation

2.1.1 Arm

Vexpress-a9

For the first implementation the qemu arm system vexpress-a9 machine was targeted.

To launch this machine with a kernel use:

```
qemu-system-arm -M vexpress-a9 -m 128M -kernel kernel/kernel_arm.bin \
  -serial stdio
```

The memory layout of this image is as follows:

- 0x00000000
- 0x10000000 : hardware.
- 0x10009000 : pl011 –> the uart peripheral
- 0x60000000 : bootloader of qemu itself.
- 0x60010000 : main memory, where kernel is loaded by the bootloader.

2.2 Design

2.2.1 Processes / threads

Processes are completely seperated and fully pre-emptive. This means a process can be unscheduled at any moment.

Threads are co-operative. This means they yield control voluntary. This means that mutexes and locks are not required. This is done with the built-in language feature called tasks.

If some heavy duty task must be performed, either way spawn a new process, or yield frequently from this hard labour.

2.2.2 tasks

Consider the following:

```
function int insanemath(int a)
{
    while (a > 0)
    {
       a = a -1;
       resume agent1;
    }
    return a - 1;
}

task agent1()
{
  start agent2;
}

task agent2()
{
   insanemath(55);
   insanemath(44);
}

task main()
{
  start agent1;
  join agent1;
}
```

Say to tasks are running in concurrent / parallel.

Stack layout for tasks. || || / +————+ | return address | locals | +—— | return address | locals | +—

Assembly code for the functions above:

```
.code
insanemath:
L1:
load r0, sp - 4
cmp r0, 0
jl L2
dec r0
store r0, sp - 4
jmp L1
L2:
ret

agent1:
hlt?

agent2:
hlt?

main:
jmp agent1

.data
agent1_task:
dd 0
agent2_task:
dd 0
```

CHAPTER 3

About

3.1 Project goals

- To write a microkernel based OS
- Write the kernel in the c3 language
- Create a c3 compiler in python

3.2 Directory structure

'kernel' contains the microkernel. 'python' contains the python utilities.

3.3 How to run this?

Install required software:

- python3.3
- (optional) pyqt5, pyqt4 or pyside

Checkout the code:

```
hg clone https://bitbucket.org/windel/lcfos
cd lcfos
```

Run some unit tests:

```
cd test
python3 run_tests.py
```

3.4 Weblinks

Docs are located here: http://lcfos.readthedocs.org/en/latest/

Sources are located here: https://bitbucket.org/windel/lcfos

here: http://hg.assembla.com/lcfOS/

and here: https://pikacode.com/windel/lcfos/

The project is contains tests which are run continuously at drone.io.

https://drone.io/bitbucket.org/windel/lcfos

Repository metrics:

http://www.ohloh.net/p/lcfos

Live demo is at redhat openshift:

http://lcfos-windel.rhcloud.com/

Unit test results:

testB (testdiagrameditor.DiagramEditorTestCase)	SKIP
testScenario1 (testdiagrameditor.DiagramEditorTestCase)	SKIP
testemulation (unittest.loader.ModuleImportFailure)	FAIL