 sysuzyc / ES2016_14353404

Watch0

Star1

Fork0

<> Code

Issues 0

Pull requests 0

Projects 0

Pulse


Graphs

Branch: master

ES2016_14353404 / assignment / dol.md

Find file

Copy path

 1111EWEEWEWEWEWE lab2

eef9475 25 days ago

1 contributor

Executable File104 lines (54 sloc)4.78 KB

Raw

Blame

History







DOL实例分析

姓名：张亚琛 学号：14353404

example1

这次的实验是修改代码，使其输出为3次方数。我们可以从以下几个方面着手，完成这次的实验。

首先，我们需要做的是找到example1.xml文件，分析一下这个文件到底是干什么的：

```
<!-- processes -->
<process name="generator">
  <port type="output" name="1"/>
  <source type="c" location="generator.c"/>
</process>

<process name="consumer">
  <port type="input" name="1"/>
  <source type="c" location="consumer.c"/>
</process>

<process name="square">
  <port type="input" name="1"/>
  <port type="output" name="2"/>
  <source type="c" location="square.c"/>
</process>
```

我们看到，这里面一共有三个进程，，所以，我们分开来看下到底他们都干了什么：

```
int generator_fire(DOLProcess *p) {
    if (p->local->index < p->local->len) {
        float x = (float)p->local->index;
        DOL_write((void*)PORT_OUT, &(x), sizeof(float), p);
        p->local->index++;
    }

    if (p->local->index >= p->local->len) {
        DOL_detach(p);
        return -1;
    }

    return 0;
}
```

其中，我们先看下generator函数中的操作，我们看到，他的意思是进行了一次遍历，如果，当前位置小于生产长度，那么我们就输出下标到输出端口，否则，就销毁程序。类似于我们的for循环进行遍历。

```

int square_fire(DOLProcess *p) {
    float i;

    if (p->local->index < p->local->len) {
        DOL_read((void*)PORT_IN, &i, sizeof(float), p);
        i = i*i*i;
        DOL_write((void*)PORT_OUT, &i, sizeof(float), p);
        p->local->index++;
    }

    if (p->local->index >= p->local->len) {
        DOL_detach(p);
        return -1;
    }

    return 0;
}

```

我们看到的是，这个部分，是对于数据的操作，一个square就是多了对传入的数据i进行了运算，这里是进行了三次方的运算，所以，一次square就是一个3次方的操作。

```

int consumer_fire(DOLProcess *p) {
    float c;

    if (p->local->index < p->local->len) {
        DOL_read((void*)PORT_IN, &c, sizeof(float), p);
        printf("%s: %f\n", p->local->name, c);
        p->local->index++;
    }

    if (p->local->index >= p->local->len) {
        DOL_detach(p);
        return -1;
    }

    return 0;
}

```

这个部分，是我们进行了数据的输出。

所以，总体来说的话，其实generator是相当于传入数据，square是进行数据的运算，consumer是进行数据的输出。

在了解了上面的信息之后，我们就可以进行结果的分析了。

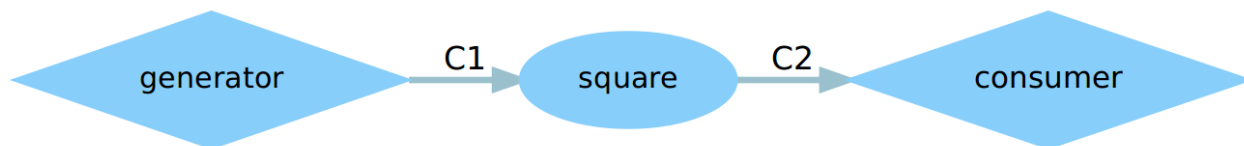
```

sysuzyc@ubuntu: ~/dol/build/bin/main
[echo] Run HdS application.
[concat] consumer: 0.000000
[concat] consumer: 1.000000
[concat] consumer: 8.000000
[concat] consumer: 27.000000
[concat] consumer: 64.000000
[concat] consumer: 125.000000
[concat] consumer: 216.000000
[concat] consumer: 343.000000
[concat] consumer: 512.000000
[concat] consumer: 729.000000
[concat] consumer: 1000.000000
[concat] consumer: 1331.000000
[concat] consumer: 1728.000000
[concat] consumer: 2197.000000
[concat] consumer: 2744.000000
[concat] consumer: 3375.000000
[concat] consumer: 4096.000000
[concat] consumer: 4913.000000
[concat] consumer: 5832.000000
[concat] consumer: 6859.000000

BUILD SUCCESSFUL
Total time: 8 seconds

```

我们看到输出的结果为上图所示，就是每个consumer是其数据的三次方，所以，我们可以猜测出，example1的dot结构为generator，square和consumer：



所以，和我们上面的猜测是一致的。

我们还可以看下的是example1.xml中的其他信息：

```

<!-- sw_channels -->
<sw_channel type="fifo" size="10" name="C1">
  <port type="input" name="0"/>
  <port type="output" name="1"/>
</sw_channel>

<sw_channel type="fifo" size="10" name="C2">
  <port type="input" name="0"/>
  <port type="output" name="1"/>
</sw_channel>

```

其中，这一部分是说明了上面的两条路线上c1和c2的一个基本情况，都是fifo，然后数据为10，所以，一共有20个数据。

而在connection中：

```

<connection name="g-c">
  <origin name="generator">
    <port name="1"/>
  </origin>
  <target name="C1">
    <port name="0"/>
  </target>
</connection>

```

这一部分是针对于generator和C1的连接的说

```

<connection name="c-c">
  <origin name="C2">
    <port name="1"/>
  </origin>
  <target name="consumer">
    <port name="1"/>
  </target>
</connection>

```

这一部分是C2和consumer的连接说明

```

<connection name="s-c">
  <origin name="square">
    <port name="2"/>
  </origin>
  <target name="C2">
    <port name="0"/>
  </target>
</connection>

```

这一部分是square和C2的连接说明

```
<connection name="c-s">
  <origin name="C1">
    <port name="1"/>
  </origin>
  <target name="square">
    <port name="1"/>
  </target>
</connection>
```

这一部分是square和C1的连接说明。

所以，我们可以看到，完全符合dot图中的描述。

example2

example2和example1其实比较接近，不过example2主要集中体现在xml文件的修改上，使其从3个square变成2个square。

前面的分析中，我们已经知道了generator，square以及consumer的函数的作用了，那么我们就主要从xml中来分析这次的实验：

```
<variable value="2" name="N"/>
```

首先，我们先是在这个地方进行了修改，因为我们需要的是两个square，所以，这里定义为2，就可以了。

```
<iterator variable="i" range="N">
  <process name="square">
    <append function="i"/>
    <port type="input" name="0"/>
    <port type="output" name="1"/>
    <source type="c" location="square.c"/>
  </process>
</iterator>
```

这个地方是定义了0为input，1为output，所以，在后面的分析中，我们就可以更好的分析了。

我们可以看下connection中的代码：

```
<iterator variable="i" range="N">
  <connection name="to_square">
    <append function="i"/>
    <origin name="C2">
      <append function="i"/>
      <port name="1"/>
    </origin>
    <target name="square">
      <append function="i"/>
      <port name="0"/>
    </target>
  </connection>
</iterator>
```

这里是说明了从square到square的连接，他们的下表是一样的，也就是说C2_1和square_1是连接在一起，一次类推，直到迭代完毕，其中，C2是output，而square是input。

```
<connection name="from_square">
  <append function="i"/>
  <origin name="square">
    <append function="i"/>
    <port name="1"/>
  </origin>
  <target name="C2">
    <append function="i + 1"/>
    <port name="0"/>
  </target>
</connection>
</iterator>
```

所以，我们看到这里是对应的square和C2的连接，其中C2的下表会大1，也就是说square_1和C2_2是连接在一起的，迭代到square的结束，其中square是output，C2是input。

剩下的就是我们的generator和consumer的连接问题了：

```
<connection name="g_">
  <origin name="generator">
    <port name="10"/>
  </origin>
  <target name="C2">
    <append function="0"/>
    <port name="0"/>
  </target>
</connection>

<connection name="c">
  <origin name="C2">
    <append function="N"/>
    <port name="1"/>
  </origin>
  <target name="consumer">
    <port name="100"/>
  </target>
</connection>
```

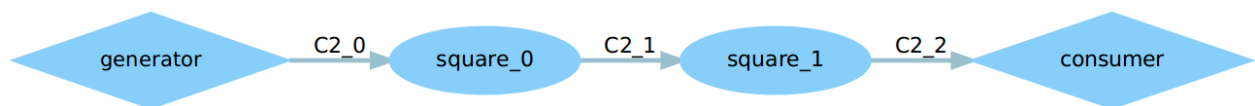
这样，可以看到的是generator和C2连接的，consumer是和C2_N连接的。

所以，按照我们的分析，结果应该为两个2次方迭代，也就是4次方的数据，所以，我们可以看到为：

```
sysuzyc@ubuntu: ~/dol/build/bin/main
[echo] Run HdS application.
[concat] consumer: 0.000000
[concat] consumer: 1.000000
[concat] consumer: 16.000000
[concat] consumer: 81.000000
[concat] consumer: 256.000000
[concat] consumer: 625.000000
[concat] consumer: 1296.000000
[concat] consumer: 2401.000000
[concat] consumer: 4096.000000
[concat] consumer: 6561.000000
[concat] consumer: 10000.000000
[concat] consumer: 14641.000000
[concat] consumer: 20736.000000
[concat] consumer: 28561.000000
[concat] consumer: 38416.000000
[concat] consumer: 50625.000000
[concat] consumer: 65536.000000
[concat] consumer: 83521.000000
[concat] consumer: 104976.000000
[concat] consumer: 130321.000000

BUILD SUCCESSFUL
Total time: 7 seconds
```

所以，我们的结果是正确的，而我们现在看下dol图：



所以，我们的结果是正确的。

实验感想

这次的实验，其实还是比较简单的，就只是需要更改一些数据而已，为什么会出现这种结果，dot图又是如何出现的，连接为什么是那个样子的，这些都是我们需要了解的，不然只是改几个数据就做完了实验的话，我们也就失去做实验的意义。所以，我们还是需要完成实验的基础上理解清楚实验的代码，知道到底怎么做之后，我们才可以很好的完成这次的实验，达到做实验的目的。

