



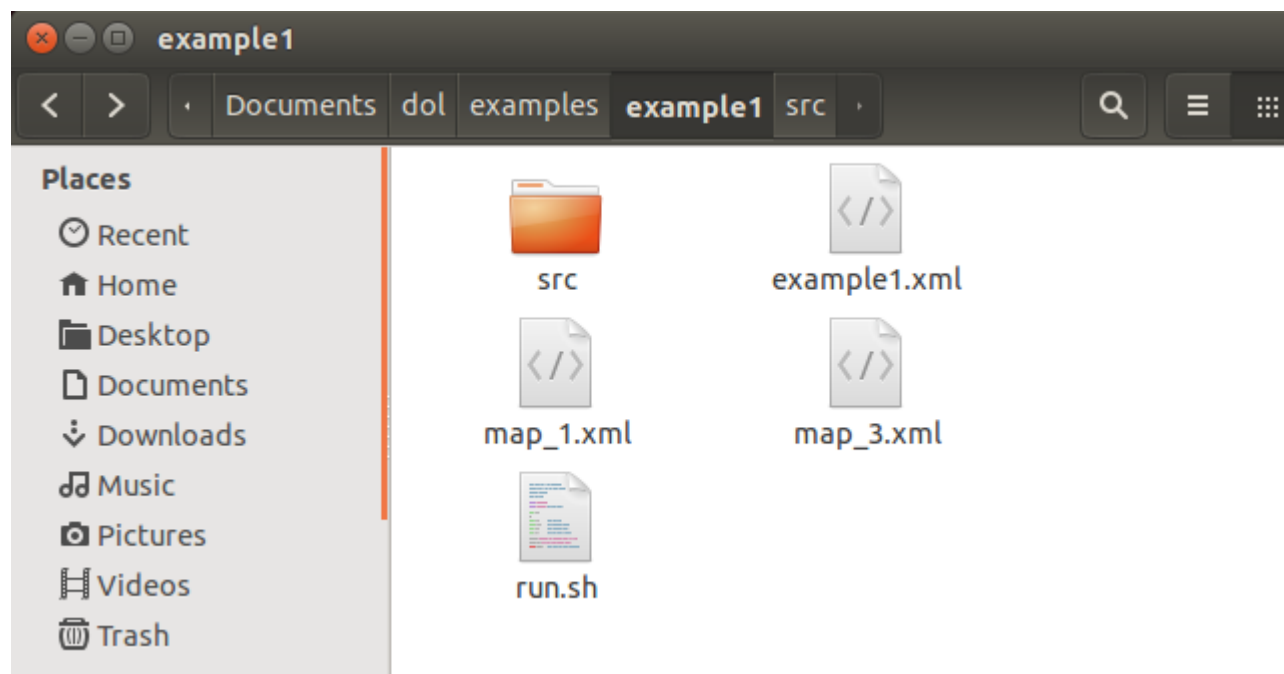
Lab2: DOL实例分析&编程



example中各文件的含义：

src文件夹：各进程（生产者，消费者，处理模块等）的功能定义

example1.xml：系统架构即模块连接方式定义



example1目录文件



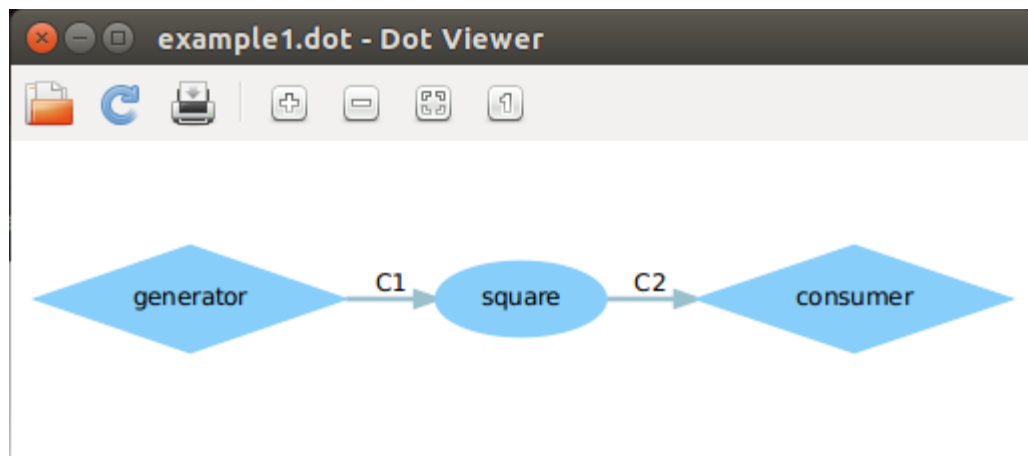
4分钟内看完代码

- /src 文件夹内包含2种文件：*.c, 与对应的.h，就是实现的模块，就是*.dot的框框的功能描述。（每个模块要实现2个接口，xxx_init和xxx_fire两个函数，分别是初始化这个模块是干了什么，以及这个模块开干的时候做什么）
- ./example*.xml 里面定义了模块与模块之间是怎么连接的，就是有哪些框，哪些线，比如A框跟B框用一根线连起来，他们就在一起了。嗯，他们。
- 这个xml是这样的：process就是那些框, sw_channel那些线, connection就是把线的那头连到框的那头。

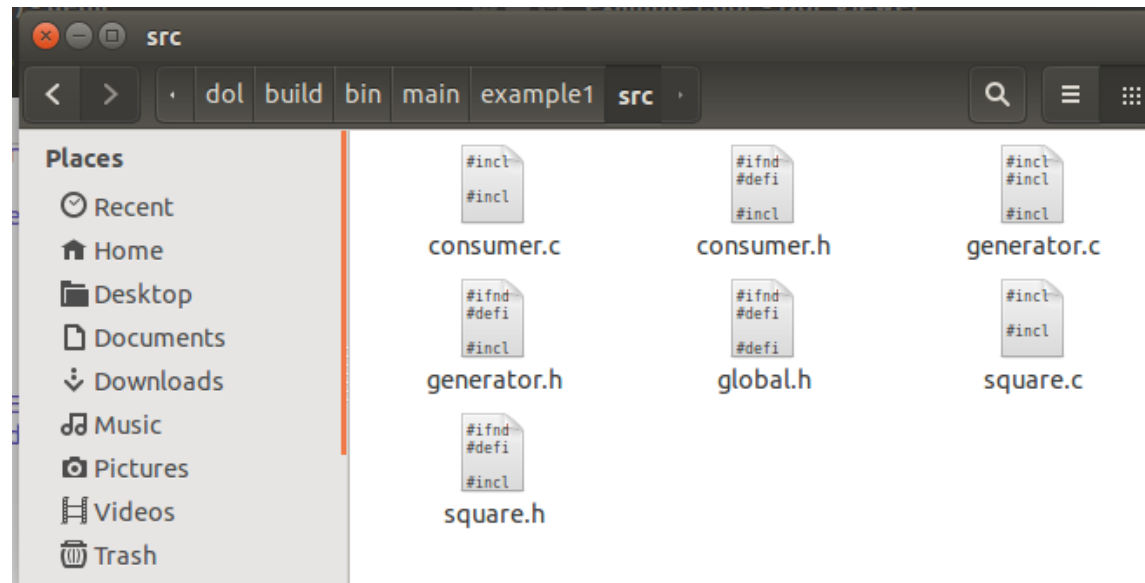


example1 代码分析

- 运行example1之后的dot图，其中包含生产者、平方模块、消费者【3个框，男朋友、女朋友和女朋友的男闺蜜】、通道C1与C2【两条线】
- 可以看到src里面，对应gennerator这个模块的代码就是gennerator.h, generator.c啦，至于里面有什么奸情。



example1 dot图



example1 src目录



generator.c 代码

```
void generator_init(DOLProcess *p) {
    p->local->index = 0;
    p->local->len = LENGTH;
}

int generator_fire(DOLProcess *p) {
    if (p->local->index < p->local->len) {
        float x = (float)p->local->index;
        //将x写到generator的端口 "PORT_OUT" 上
        DOL_write((void*)PORT_OUT, &(x), sizeof(float), p);
        p->local->index++;
    }
    if (p->local->index >= p->local->len) {
        DOL_detach(p); //销毁
        return -1;
    }
    return 0;
}
```

- 定义进程：每个模块都要写上xxx_fire（可能被执行无数次），至于init是可选择写或者不写的，xxx_init（只会被执行一次）。
- generator_init 是初始化函数。这里代码的意思是将当前位置置为0，设置生产者长度。这里的local指针指向的是.h文件的_local_states结构。
- generator_fire 是信号产生函数。这里的代码是：如果当前位置小于生产长度，则将x（这里是当前下标）写入到输出端，否则销毁进程。所以说就是，让这个程序被发射、开火、执行length次之后停下来。



consumer.c 代码

```
void consumer_init(DOLProcess *p) {
    sprintf(p->local->name, "consumer"); //就是p->local->name=="consumer"
    p->local->index = 0;
    p->local->len = LENGTH;
}

int consumer_fire(DOLProcess *p) {
    float c;
    if (p->local->index < p->local->len) {
        DOL_read((void*)PORT_IN, &c, sizeof(float), p); //读consumer的端口 "PORT_IN"
        printf("%s: %f\n", p->local->name, c); //将结构输出到命令行
        p->local->index++;
    }
    if (p->local->index >= p->local->len) {
        DOL_detach(p);
        return -1;
    }
    return 0;
}
```

- 定义消费者进程
- consumer_init初始化函数，含义同generator_init。
- consumer_fire信号消费函数，若当前位置小于设定长度，则读出输入端信号，并且打印；否则销毁进程（停下来）。



square.c 部分代码

```
int square_fire(DOLProcess *p) {  
    float i;  
    if (p->local->index < p->local->len) {  
        // 读square的端口 “PORT_IN”, 将值读到  
        DOL_read((void*)PORT_IN, &i, sizeof(float), p);  
        i = i*i; //做了个平方  
        // 写square的端口 “PORT_OUT”, 把i 写到那个端口  
        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;  
}
```

- 定义平方进程
- square_fire信号处理函数，读入输入端信号i，将其平方后写出到输出端，也是重复length次之后就停止了。



./examples/example1/example1.xml文件

```
<!-- 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>
```

```
1 #ifndef G_H
2 #define G_H
3
4 #include <dl.h>
5 #include "global.h"
6
7 #define PORT_IN "gin"
8 #define PORT_OUT0 "gout0"
9 #define PORT_OUT1 "gout1"
10
```

- 进程定义

<process name="未知数1">

<port type="未知数2" name="未知数3"/>

有几个端口就要有几行这个

<port type="未知数2" name="未知数3"/>

这里就是说有2个端口

<source type="c" location="未知数1.c"/>

</process>

- 未知数1==实现的模块的名字，比如写了xxx.c这里就是xxx了
- 未知数2==output或者input
- 未知数3==端口的名字，在*.h的文件里面，见左图



```
<!-- 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>
```

- 通道定义，一条线就是一条通道

```
<sw_channel type="fifo" size="未知数1" name="未知数2">
```

```
  <port type="input" name="in"/>
```

```
  <port type="output" name="out"/>
```

两个端口，一个叫“in”，一个叫“out”

```
</sw_channel>
```

- 未知数1是指缓冲区的大小，左边是10
- 未知数2是这条线的名字，比如左边是C2



```
<!-- connections -->
-<connection name="g-c">
  -<origin name="generator">
    <port name="1"/>
  </origin>
  -<target name="C1">
    <port name="0"/>
  </target>
</connection>
-<connection name="c-c">
  -<origin name="C2">
    <port name="1"/>
  </origin>
  -<target name="consumer">
    <port name="1"/>
  </target>
</connection>
+<connection name="s-c"></connection>
+<connection name="c-s"></connection>
```

举个例子，上面这条connection叫“g-c”；它从“generator”这个模块的“1”端口连接到“c1”这个模块（线）的“0”端口。此时还需要对应另外的一条connection，它要从“c1”这个模块（线）的其他端口，连接到其他的模块上。

- 定义各个模块之间的连接,一条线会对应两个connection，就是A框的右手牵着这条线的左边，这条线的右边牵着B框。**！！每条线要有2个connection！！**

```
<connection name="未知数1">
  <origin name="未知数2">!!从这!!
    <port name="未知数3"/>
  </origin>
  <target name="未知数4">!!连到这!!
    <port name="未知数5"/>
  </target>
</connection>
```

- 未知数1是这段感情的名字，随便填。
- 未知数2\未知数4是模块或者通道的名字
- 未知数3\未知数5要对应process或者channel的端口名



```
[concat] consumer: 0.000000
[concat] consumer: 1.000000
[concat] consumer: 4.000000
[concat] consumer: 9.000000
[concat] consumer: 16.000000
[concat] consumer: 25.000000
[concat] consumer: 36.000000
[concat] consumer: 49.000000
[concat] consumer: 64.000000
[concat] consumer: 81.000000
[concat] consumer: 100.000000
[concat] consumer: 121.000000
[concat] consumer: 144.000000
[concat] consumer: 169.000000
[concat] consumer: 196.000000
[concat] consumer: 225.000000
[concat] consumer: 256.000000
[concat] consumer: 289.000000
[concat] consumer: 324.000000
[concat] consumer: 361.000000

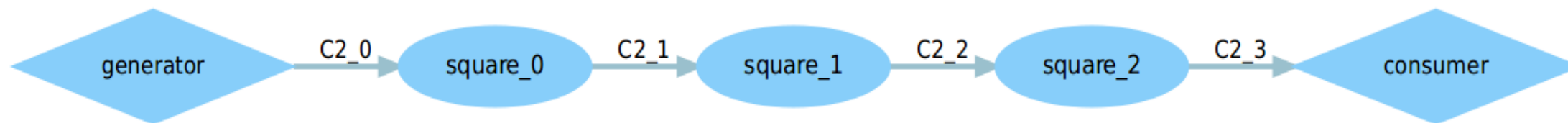
BUILD SUCCESSFUL
Total time: 20 seconds
```

- example1 运行结果如左图所示
- generator 产生0-19的整数（length为20，初始值为0）
- square 对输入进行平方操作
- consumer 输出结果



example2 代码分析

- 各进程功能定义与example1相同，不同之处在于example2架构中包含3个square进程，故结果为 i^8



example2 dot图



- 通过迭代，定义了3个square模块

```
<variable value="3" name="N"/>
<!-- instantiate resources -->
-<process name="generator">
  <port type="output" name="10"/>
  <source type="c" location="generator.c"/>
</process>
-<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>
-<process name="consumer">
  <port type="input" name="100"/>
  <source type="c" location="consumer.c"/>
</process>
-<iterator variable="i" range="N + 1">
  -<sw_channel type="fifo" size="10" name="C2">
    <append function="i"/>
    <port type="input" name="0"/>
    <port type="output" name="1"/>
  </sw_channel>
</iterator>
```

- 迭代生成连接connection

```
<!-- instantiate connection -->
-<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>
  -<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>
```



```
[concat] consumer: 0.000000  
[concat] consumer: 1.000000  
[concat] consumer: 256.000000  
[concat] consumer: 6561.000000  
[concat] consumer: 65536.000000  
[concat] consumer: 390625.000000  
[concat] consumer: 1679616.000000  
[concat] consumer: 5764801.000000  
[concat] consumer: 16777216.000000  
[concat] consumer: 43046720.000000  
[concat] consumer: 100000000.000000  
[concat] consumer: 214358880.000000  
[concat] consumer: 429981696.000000  
[concat] consumer: 815730752.000000  
[concat] consumer: 1475789056.000000  
[concat] consumer: 2562890752.000000  
[concat] consumer: 4294967296.000000  
[concat] consumer: 6975757312.000000  
[concat] consumer: 11019960320.000000  
[concat] consumer: 16983563264.000000
```

BUILD SUCCESSFUL

- example2 运行结果如左图所示
- generator 生成0~19的整数
- square 经过三次平方操作
- consumer 输出结果



实验报告提交及要求:

- 任务:
 1. 修改example2, 让3个square模块变成2个, tips:修改xml的iterator
 2. 修改example1, 使其输出3次方数, tips:修改square.c
- 提示: 修改代码之后要重新编译、运行 (`sudo ant -f runexample.xml -Dnumber=XXX`) XXX是运行example的号码, 比如上面是2和1
- `git add ./assignment/dol.md`:
 - 步骤: 新建文件夹assignment 在该文件夹下添加 dol.md
 - 1. 改完的*.dot截图
 - 2. 具体如何修改的解释 (不要写太长=。=)
 - 3. 实验感想
- 截止日期: TBD