Shell 程序设计 —2

曹东刚

caodg@sei.pku.edu.cn

Linux 程序设计环境

http://c.pku.edu.cn/

内容提要

条件判断

```
~$ test condition
~$ [ condition ]
```

条件判断

判断字符串是否相等, 检查文件状态, 数字测试等.

- 测试的结果为 true (返回 0) 或者 false (返回 1),
- 用 [␣condition␣] 的时候要注意, condition 左右两侧必须要有空格
- 引用变量的时候最好加双引号, 例 "$arg"
- test, true, false 既是 shell 函数, 同时也是 shell 命令.

示例

```
~$ test 1
~$ test 0
~$ test true
~$ test false
~$ [ 1 ]
```

文件属性判断

表达式	含义
`-d file`	file 存在且是目录
`-e file`	file 存在
`-f file`	file 存在且是普通文件
`-r file`	file 存在且可读
`-w file`	file 存在且可写
-x file	file 存在且可执行
`-s file`	file 存在且长度非零
`-h file`	file 存在且为符号链接
`file1 -nt file2`	file1 比 file2 新
`file1 -ot file2`	file1 比 file2 旧

示例

例: 测试当前目录的文件 check.sh 是否可执行

```
~$ [ -x check.sh ]
~$ echo $?
```

例: 测试是否存在目录文件 mydir

```
~$ [ -d mydir ]
~$ echo $?
```

字符串比较

表达式	含义
`-n str`	str 长度非 0
`-z str`	str 长度为 0
`str1 = str2`	str1 和 str2 相同[1]
`str1 != str2`	str1 和 str2 不等

[1] bash 中也可以用 == 判断字符串相等

示例

例: 字符串 str1="aa bb", 判断是否等于 "aabb"

```
~$ [ "$str1" = "aabb" ]
```

比较

```
~$ [ $str1 = "aabb" ]
```

例: 判断字符串 str1 和 str2 是否相等

```
~$ [ "$str1" = "$str2" ]
```

数字串比较

表达式	含义
`int1 -eq int2`	int1 和 int2 相等
`int1 -ne int2`	int1 和 int2 不等
`int1 -gt int2`	int1 大于 int2
`int1 -ge int2`	int1 大于等于 int2
`int1 -lt int2`	int1 小于 int2
`int1 -le int2`	int1 小于等于 int2

示例

例: 数字串 int1="1234", 判断是否等于 "01234"

```
~$ [ "$int1" -eq "01234" ]
```

比较

```
~$ [ "01234" = "1234" ]
```

例: 判断字符串 str 长度是否大于 3

```
~$ [ `expr length "$str"` -gt 3 ]
~$ [ $(expr length "$str") -gt 3 ]
```

复合表达式

`\( expression \)`	一组表达式, expression 为真则表达式为真
`! expression`	expression 为假则表达式为真
`expr1 -a expr2`	expr1 和 expr2 同时为真则表达式为真
`expr1 -o expr2`	expr1 和 expr2 有一个为真则表达式为真

```
~$ [ -z "$DTHOME" -a -d /usr/dt ]
~$ [ -z "$DTHOME" ] && [ -d /usr/dt ]
```

复合表达式

`\( expression \)`	一组表达式, expression 为真则表达式为真
`! expression`	expression 为假则表达式为真
`expr1 -a expr2`	expr1 和 expr2 同时为真则表达式为真
`expr1 -o expr2`	expr1 和 expr2 有一个为真则表达式为真

```
~$ [ -z "$DTHOME" -a -d /usr/dt ]
~$ [ -z "$DTHOME" ] && [ -d /usr/dt ]
```

内容提要

命令执行顺序

Unix 提供了多种控制命令执行顺序的机制. 例如, 确保文件成功复制后, 才删除原来的文件.

- ; 顺序执行
- && 只有 && 左侧执行成功才执行右边命令
- || 只有 || 左侧执行失败才执行右边命令
- & 后台执行

示例

先关闭 eth0, 再启动 eth0

```
~$ ifconfig eth0 down ; ifconfig eth0 up
```

先复制目录, 成功后再删除原来目录

```
~$ cp -r apps ./bak && rm -rf apps
```

复制文件, 如果出错则打印 Error

```
~$  cp a.txt b.txt || echo Error
```

后台执行 fetchmail 程序

```
~$ fetchmail &
```

示例

先关闭 eth0, 再启动 eth0

```
~$ ifconfig eth0 down ; ifconfig eth0 up
```

先复制目录, 成功后再删除原来目录

```
~$ cp -r apps ./bak && rm -rf apps
```

复制文件, 如果出错则打印 Error

```
~$  cp a.txt b.txt || echo Error
```

后台执行 fetchmail 程序

```
~$ fetchmail &
```

示例

先关闭 eth0, 再启动 eth0

```
~$ ifconfig eth0 down ; ifconfig eth0 up
```

先复制目录, 成功后再删除原来目录

```
~$ cp -r apps ./bak && rm -rf apps
```

复制文件, 如果出错则打印 Error

```
~$  cp a.txt b.txt || echo Error
```

后台执行 fetchmail 程序

```
~$ fetchmail &
```

示例

先关闭 eth0, 再启动 eth0

```
~$ ifconfig eth0 down ; ifconfig eth0 up
```

先复制目录, 成功后再删除原来目录

```
~$ cp -r apps ./bak && rm -rf apps
```

复制文件, 如果出错则打印 Error

```
~$  cp a.txt b.txt || echo Error
```

后台执行 fetchmail 程序

```
~$ fetchmail &
```

if 语句：多行模式

```
if condition1
then
    list1
elif condition2
then
    list2
else
    list3
fi
```

if 语句: 单行模式

```
if condition1; then
list1;
elif condition2; then
list2;
else
list3;
fi;
```

注意事项

常见错误:

- 单行模式时, then 前面漏掉分号";"
- 使用 elif 语句时忘记 then
- 用 else if 或者 elsif 而不是 elif
- if 语句的末尾用了 if 而不是 fi

示例 1

```
1 if uuencode simsun.ttc simsun.ttc > simsun.uu ; then
2     echo "Encoded simsun.ttc to simsun.uu"
3 elif rm simsun.uu ; then
4     echo "Encoding failed, temporary files removed."
5 else
6     echo "An error occured."
7 fi
8 uuencode simsun.ttc simsun.ttc | mail y@org.cn
9 uudecode message.eml
```

示例 2

```
1 if [ ! -d $HOME/bin ] ; then
2     mkdir $HOME/bin
3 fi
```

比较:

```
1 [ ! -d $HOME/bin ] && mkdir $HOME/bin
```

示例 2

```
1 if [ ! -d $HOME/bin ] ; then
2     mkdir $HOME/bin
3 fi
```

比较:

```
1 [ ! -d $HOME/bin ] && mkdir $HOME/bin
```

case

case 语句的基本语法为

```
case word in
    pattern1)
        list1
        ;;
    pattern2)
        list2
        ;;
esac
```

规则

- 字符串 word 与每一个模式进行比较，直到找到一个匹配，然后执行其后的命令清单
- 若找不到匹配，case 语句不执行任何动作并退出
- 匹配数量无上限，但至少应有一个
- 模式可使用与路径名相同的特殊字符，以及“或”字符“|”

示例 1

```
1  case "$TERM" in
2      network | dialup | unknown | vt[0-9][0-9][0-9])
3          TERM=vt100 ;;
4      *term)
5          TERM=xterm ;;
6      *)
7          echo "Error, quit"
8          exit 1
9          ;;
10 esac
```

示例 2

提示用户输入 yes, y, n, No

```
1  echo "Do you wish to continue ? [Yes/No]
2  read ANS
3  case "$ANS" in
4      y|Y|[yY][eE][sS])
5          process ;;
6      n|N|[nN][oO])
7          abort ;;
8      *)
9          echo "Error input"
10         ;;
11 esac
```

for

for 语句的基本语法为

```
for name in wordlist ; do
    list
done
```

- name 是变量名
- wordlist 是被空格分开的单词序列
- 每次循环时, name 被设为单词中的下一个单词

示例

```
1  for i in 0 1 2 3 4 5 6
2  do
3      echo $i
4  done
5
6  for i in `seq 1 2 100`
7  do
8      echo $i
9  done
```

示例

```
1 counter=0
2 for f in *
3 do
4     counter=`expr $counter + 1`
5 done
6 echo "There are $counter files"
```

while

while 语句的基本语法为

```
while command ; do
    list
done
```

- command 是要执行的单条命令, 通常是一个 test 表达式
- list 称为 while 循环体
- 若 command 的退出状态为 0, 则执行 list; 否则退出

示例 1

```
1 x=0
2 while [ $x -lt 10 ]
3 do
4     echo $x
5     x=`echo "$x + 1" | bc`
6 done
```

示例 2: 验证用户的输入

```
1  RESPONSE=
2  while [ -z "$RESPONSE" ]
3  do
4      echo "Enter a directory name "
5      read RESPONSE
6      if [ ! -d "$RESPONSE" ] ; then
7          echo "ERROR: please input a directory name"
8          RESPONSE=
9      fi
10 done
```

select

select 循环提供了一种从用户可选项中创建编号菜单的便捷形式[2]. 基本语法为

```
select name in wordlist ; do
    list
done
```

- wordlist 是由空格分开的单词序列
- 用户输入值保存在变量 $REPLY 中
- 若没有使用循环控制机制跳出 select 循环, 则重复选择过程

[2]select 由 ksh 引入, 在 bash 上兼容, 但 bourne sh 不支持

示例

```
select COMPONENT in comp1 comp2 comp3 all none
do
    case $COMPONENT in
    comp1|comp2|comp3) process $COMPONENT ;;
    all) process comp1
         process comp2
         process comp3
         ;;
    none) break ;;
    *) echo "ERROR: Invalid selection, $REPLY." ;;
    esac
done
```

无限循环和 break

无限循环永远不中止. 例:

```
1 while :
2 do
3     read CMD
4     case "$CMD" in
5     [qQ]|[qQ][uU][iI][tT]) break ;;
6     *) process $CMD ;;
7     esac
8 done
```

continue 命令

continue 命令中止当前迭代, 但不退出整个循环. 例:

```
1 for FILE in FILES
2 do
3     if [ ! -f "$FILE" ] ; then
4         echo "ERROR: $FILE is not a file."
5         continue
6     fi
7     # process the file
8 done
```

综合示例 —打印 usage 语句

```
1 USAGE="Usage: `basename $0` [-c|-t] [file|directory]"
2 case "$1" in
3     -t) TARGS="-tvf $2" ;;
4     -c) TARGS="-cvf $2.tar $2" ;;
5     *)  echo "$USAGE"
6         exit 0
7         ;;
8 esac
9 tar $TARGS
```

综合示例 —检查参数数目

```
1  USAGE="Usage: `basename $0` [-c|-t] [file|directory]"
2  if [ $# -lt 2 ] ; then
3      echo "$USAGE"
4      exit 1
5  fi
6
7  case "$1" in
8      -t) TARGS="-tvf $2" ;;
9      -c) TARGS="-cvf $2.tar $2" ;;
10     *)  echo "$USAGE"
11         exit 0
12         ;;
13 esac
14 tar $TARGS
```

综合示例 —检查参数语义

```
1  USAGE="Usage: `basename $0` [-c|-t] [file|directory]"
2  if [ $# -lt 2 ] ; then
3      echo "$USAGE"
4      exit 1
5  fi
6  case "$1" in
7      -t) TARGS="-tvf"
8          for i in "$@" ; do
9              if [ -f "$i" ] ; then tar $TARGS "$i" ; fi ;
10         done
11         ;;
12     -c) TARGS="-cvf $2.tar $2"
13         tar $TARGS
14         ;;
15     *)  echo "$USAGE"
16         exit 0 ;;
17 esac
```

综合示例 —进一步提高

```
case "$1" in
    -t) shift ; TARGS="-tvf"
        for i in "$@" ; do
            if [ -f "$i" ] ; then
                FILES=`tar $TARGS "$i" 2>/dev/null`
                if [ $? -eq 0 ] ; then
                    echo ; echo "$i" ; echo "$FILES"
                else
                    echo "ERROR: $i not a tar file."
                fi
            else
                echo "ERROR: $i not a file."
            fi
        done
        ;;
```

综合示例 —进一步提高 (cont.)

```
16      -c) shift ; TARGS="-cvf"
17          tar $TARGS archive.tar "$@"
18          ;;
19      *)  echo "$USAGE"
20          exit 0 ;;
21  esac
```

参数选项分析的两种方法

一种是通过 case 语句手工分析，另一种是通过 getopts 命令实现.

getopts命令的语法为：getopts option-string variable

- option-string 为包含所有单字符选项的字符串，这些选项应该赋予一个变量，即 variable
- 通常使用的 variable 变量名为 OPTION
- getopts 支持额外参数，通过在 option-string 中的选项后面加上“:”字符即可实现. 在这种情况下，选项被分析后，额外参数被设置为变量 OPTARG 的值

getopts 分析过程

1. getopts 检查所有参数，找到以“-”字符开头的字符
2. 将“-”后的字符与 option-string 中给出的字符比较
3. 若找到匹配，则 variable 被设置为选项，否则 variable 被设置为“?”字符
   - 若找到匹配且 option-string 中的字符后跟“:”，则读入下一个参数，将其赋值给 OPTARG
4. 重复 1 - 3，直至处理完所有选项
5. 当分析结束后，getopts 返回非零值并退出，并设置变量 OPTIND 作为下一参数的位置索引

getopts 分析过程

1. getopts 检查所有参数，找到以“-”字符开头的字符
2. 将“-”后的字符与 option-string 中给出的字符比较
3. 若找到匹配，则 variable 被设置为选项，否则 variable 被设置为“?”字符
   - 若找到匹配且 option-string 中的字符后跟“:”，则读入下一个参数，将其赋值给 OPTARG
4. 重复 1 - 3，直至处理完所有选项
5. 当分析结束后，getopts 返回非零值并退出，并设置变量 OPTIND 作为下一参数的位置索引

getopts 分析过程

1. getopts 检查所有参数，找到以“-”字符开头的字符
2. 将“-”后的字符与 option-string 中给出的字符比较
3. 若找到匹配，则 variable 被设置为选项，否则 variable 被设置为“?”字符
   - 若找到匹配且 option-string 中的字符后跟“:”，则读入下一个参数，将其赋值给 OPTARG
4. 重复 1 - 3，直至处理完所有选项
5. 当分析结束后，getopts 返回非零值并退出，并设置变量 OPTIND 作为下一参数的位置索引

getopts 分析过程

1. getopts 检查所有参数，找到以“-”字符开头的字符
2. 将“-”后的字符与 option-string 中给出的字符比较
3. 若找到匹配，则 variable 被设置为选项，否则 variable 被设置为“?”字符
   - 若找到匹配且 option-string 中的字符后跟“:”，则读入下一个参数，将其赋值给 OPTARG
4. 重复 1 - 3，直至处理完所有选项
5. 当分析结束后，getopts 返回非零值并退出，并设置变量 OPTIND 作为下一参数的位置索引

getopts 分析过程

1. getopts 检查所有参数，找到以“-”字符开头的字符
2. 将“-”后的字符与 option-string 中给出的字符比较
3. 若找到匹配，则 variable 被设置为选项，否则 variable 被设置为“?”字符
   - 若找到匹配且 option-string 中的字符后跟“:”，则读入下一个参数，将其赋值给 OPTARG
4. 重复 1 - 3，直至处理完所有选项
5. 当分析结束后，getopts 返回非零值并退出，并设置变量 OPTIND 作为下一参数的位置索引

getopts 分析过程

1. getopts 检查所有参数，找到以“-”字符开头的字符
2. 将“-”后的字符与 option-string 中给出的字符比较
3. 若找到匹配，则 variable 被设置为选项，否则 variable 被设置为“?”字符
   - 若找到匹配且 option-string 中的字符后跟“:”，则读入下一个参数，将其赋值给 OPTARG
4. 重复 1 - 3，直至处理完所有选项
5. 当分析结束后，getopts 返回非零值并退出，并设置变量 OPTIND 作为下一参数的位置索引

getopts 示例 —问题描述

写一个脚本 uu.sh，调用 uuencode 将二进制文件文本化.

uuencode 的调用形式为 uuencode [file] name，读入文件 file，将内容编码输出到屏幕. uu.sh 接受如下参数

- -f：指明输入文件名
- -o：指明输出文件名
- -v：指明脚本应输出详细执行信息

uu.sh 可能的命令

```
$ uu.sh chap01.pdf
$ uu.sh -f chap01.pdf -o chap01.uu
```

getopts 示例 —读入参数

```
1  #!/bin/sh
2  USAGE="Usage: `basename $0` [-v] [-f file] [-o file]";
3  VERBOSE=false
4
5  while getopts f:o:v OPTION ; do
6          case "$OPTION" in
7          f) INFILE="$OPTARG" ;;
8          o) OUTFILE="$OPTARG" ;;
9          v) VERBOSE=true ;;
10         \?) echo "$USAGE" ;
11                 exit 1 ;;
12         esac
13 done
```

getopts 示例 —错误处理

脚本应该检查输入文件，进行人性化处理

```
14 shift `expr $OPTIND - 1`
15
16 if [ -z "$1" -a -z "$INFILE" ] ; then
17         echo "ERROR: Input file was not specified."
18         exit 1
19 fi
20
21 if [ -z "$INFILE" ] ; then INFILE="$1" ; fi
22
23 : ${OUTFILE:=${INFILE}.uu}
```

getopts 示例 —任务处理

```
25 if [ -f "$INFILE" ] ; then
26     if [ "$VERBOSE" = "true" ] ; then
27         echo -e "uuencoding $INFILE to $OUTFILE ... \c"
28     fi
29     uuencode $INFILE $INFILE > $OUTFILE ; RET=$?
30     if [ "$VERBOSE" = "true" ] ; then
31         MSG="Failed."
32         if [ $RET -eq 0 ] ; then MSG="Done." ; fi
33         echo $MSG
34      fi
35 else
36     echo "ERROR: $INFILE not exists"
37 fi
```

内容提要

函数定义

shell 函数的正式定义为:

name() list ;

- 函数将一个名字 name 与命令清单 list 绑定在一起
- 函数定义时必须要用到 “(” 和 “)” 字符
- shell 函数可以替代二进制文件或 shell 内置同名命令
- 函数在当前 shell 执行
- 可以用 shell 模拟 alias 命令

函数示例 —无参

例: 模拟 csh 的 source 命令

```
1 source() { . "$@" ;}
```

例: 打印当前 PATH 的值

```
1 lspath() {
2     OLDIFS="$IFS"
3     IFS=:
4     for DIR in $PATH ; do echo $DIR ; done
5     IFS=$OLDIFS
6 }
```

函数示例 —无参

例: 模拟 csh 的 source 命令

```
1 source() { . "$@" ;}
```

例: 打印当前 PATH 的值

```
1 lspath() {
2     OLDIFS="$IFS"
3     IFS=:
4     for DIR in $PATH ; do echo $DIR ; done
5     IFS=$OLDIFS
6 }
```

函数示例 —传递参数

```
1  SetPath() {
2      for _DIR in "$@"
3      do
4          if [ -d "$_DIR" ] ; then
5              PATH="$PATH":"$_DIR"
6          fi
7      done
8      export PATH
9      unset _DIR
10 }
```

内容提要

信号处理

- 信号 (signal) 是向一个程序发出的软件中断，它指出发生了一个重要的事件
- 可利用信号户要求程序做不属于正常流程的事情

Num	Name	Action
1	HUP	重起进程
2	INT	中断
3	QUIT	退出
9	KILL	强制杀死 (unblock)
15	TERM	软中止

信号缺省动作

每个信号都有一个与之关联的缺省动作：在接收到该信号时执行的动作. 可能的缺省动作有：

- 中止进程
- 忽略信号
- 内核转储
- 停止进程
- 继续一个停止的进程，等

处理信号

一个脚本或程序可以用三种方式处理信号:

- 不做任何处理而让缺省动作发生
- 忽略信号并继续执行
- 捕获信号并执行一些信号特定命令

捕获信号

trap 命令设置或取消接收到一个信号时的动作，其语法为 trap function signals

- 若不给出 function，则将给定信号的动作重设为缺省动作
- trap常用于清除临时文件，忽略信号，设置计时器等

示例: 清除临时文件

```
1 trap "rm -f $TEMP ; exit 2" 1 2 3 15
```

更复杂的清除

```
1 CleanUp() {
2     if [ -f "$OUTFILE" ] ; then
3         printf "Cleanning up ..."
4         rm -f "$OUTFILE" 2> /dev/null
5         echo "Done."
6     fi
7 }
8 trap CleanUp 1 2 3 15
```

示例: 清除临时文件

```
1 trap "rm -f $TEMP ; exit 2" 1 2 3 15
```

更复杂的清除

```
1 CleanUp() {
2     if [ -f "$OUTFILE" ] ; then
3         printf "Cleanning up ..."
4         rm -f "$OUTFILE" 2> /dev/null
5         echo "Done."
6     fi
7 }
8 trap CleanUp 1 2 3 15
```

忽略信号

```
trap '' 1 2 3 15
```

或者

```
trap : 1 2 3 15
```

在关键操作期间忽略信号

```
1 trap '' 1 2 3 15
2 DoImportantStuff
3 trap 1 2 3 15
```

忽略信号

```
trap '' 1 2 3 15
```

或者

```
trap : 1 2 3 15
```

在关键操作期间忽略信号

```
1 trap '' 1 2 3 15
2 DoImportantStuff
3 trap 1 2 3 15
```

忽略信号

`trap '' 1 2 3 15`

或者

`trap : 1 2 3 15`

在关键操作期间忽略信号

```
1 trap '' 1 2 3 15
2 DoImportantStuff
3 trap 1 2 3 15
```

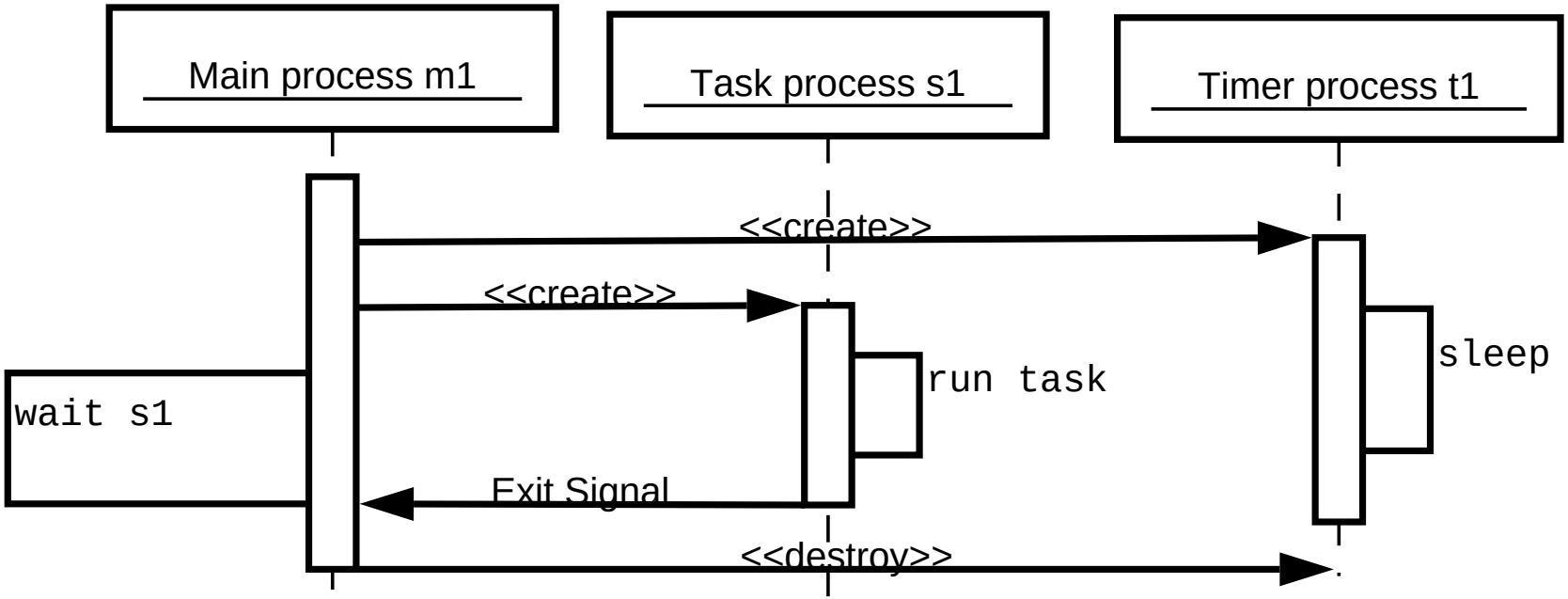
定时器技术

有时候，程序对多任务进行批处理. 为了防止某个任务非正常运行，占有处理器不退出 (如死循环，等待其它进程的信号)，需要一种机制及时终止该任务. 定时器技术可以解决此类问题.

- 通常需要三个进程: 控制主进程，定时器进程，任务子进程
- 各进程之间通过信号机制通信

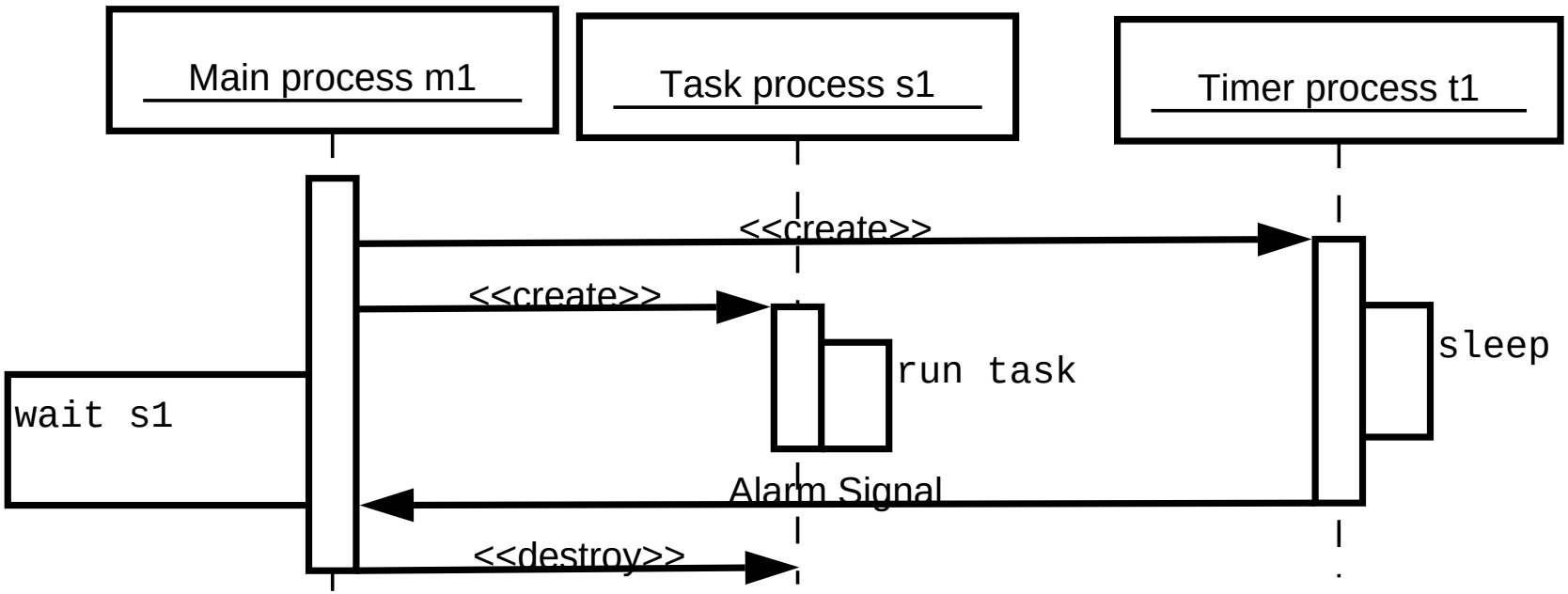
原理

子任务正常返回


Main process m1
Task process s1
Timer process t1
<<create>>
<<create>>
wait s1
run task
sleep
Exit Signal
<<destroy>>

原理

子任务阻塞，定时器进程触发 Alarm 信号



代码示例: 信号处理函数

```
1  AlarmHandler()
2  {
3      echo "Got SIGALARM, cmd took too long"
4      kill ${CHPROCIDS:-$!}
5      if [ $? -eq 0 ] ; then
6          TIMERPROC=
7          echo "Sub process killed."
8      fi
9  }
```

代码示例: 设置定时器

```
10 setTimer()
11 {
12     DEF_TOUT=${1:-10};
13     if [ $DEF_TOUT -ne 0 ] ; then
14         sleep $DEF_TOUT && kill -s 14 $$ &
15         echo "settimer $!"
16         TIMERPROC=$!
17     fi
18 }
19
20 UnsetTimer()
21 {
22     CHPROCIDS=
23     [ -n "$TIMERPROC" ] && kill $TIMERPROC
24 }
```

代码示例: 主进程

```
25 trap AlarmHandler 14
26
27 if [ -f date.sh ] ; then
28     echo " -->   run date.sh "
29     setTimer 3
30     sh date.sh &
31     CHPROCIDS="$CHPROCIDS $!"
32     wait $!
33     echo " -->   end date.sh "
34     UnsetTimer
35 fi
36 wait
```