bash script multi trheading

if you want to execute several operations in parallel in a bash script, you can simply create a function and then run this function in the background. this will let your script continue while the funciton is running in a separate process..

here is an example:

```
#!/bin/bash

run(){
    for i in {1..5}; do
        echo "$1: $VALUE, $(</dev/shm/global)"
        sleep 2
    done
}
echo "global" > /dev/shm/global
VALUE="test"
run 0 &
for i in {1..5}; do
    sleep 2
    VALUE="test$i"
    echo "global$i" > /dev/shm/global
    echo "VALUE set to $VALUE and GLOB to $GLOB"
    run $i &
done
```

this will run the run() function in the background and it will start a new one every 2 seconds.. the output will be something like:

```
psuter@psuter:~/Desktop$ ./multithreading.sh
0: test, global
VALUE set to test1 and GLOB to
1: test1, global1
0: test, global1
VALUE set to test2 and GLOB to
1: test1, global2
0: test, global2
2: test2, global2
VALUE set to test3 and GLOB to
1: test1, global3
0: test, global3
2: test2, global3
3: test3, global3
VALUE set to test4 and GLOB to
0: test, global4
1: test1, global4
3: test3, global4
4: test4, global4
```

```
2: test2, global4
VALUE set to test5 and GLOB to
psuter@psuter:~/Desktop$ 1: test1, global5
5: test5, global5
4: test4, global5
2: test2, global5
3: test3, global5
5: test5, global5
4: test4, global5
3: test3, global5
2: test2, global5
5: test5, global5
4: test4, global5
3: test3, global5
5: test5, global5
4: test4, global5
5: test5, global5
```

you can see several things about the behaviour of such background processes:

1. if the main process finishes (the process i started from the command line), not all run() functions are done yet, so their processes will continue..
2. however, if the main process is still around and you hit ctrl+c, it will kill also all its sub processes.. if the main process is not around anymore, ctrl+c will have no effect on the sub processes..
3. a variable that is set before the function is run and then changed while the funciton is running will remain its original value inside of the function.. that's why in our output, $VALUE does not change.. In order to get “live” data into the background function or back from the background function, you need to use some kind of a rendez-vous, a resource both have access too.. the simplest is a file.. if it's only for a small amount of data, you might want to use something in /dev/shm to keep your harddrives free for real data..

wait for subprocesses

if you want to be able to use ctrl+c to abort the execution of all children even after the main process has completed, you need to keep it runnig for as long as the child processes are running.. my solution looks like this:

```
self=`basename "$0"`
children=`ps aux | grep "$self" | grep -v "grep"`
while [ `echo "$children" | wc -l` -gt 2 ]; do
    sleep 1
    echo "still have `echo "$children" | wc -l` processes, keep waiting: "
    echo "$children"
    children=`ps aux | grep "$self" | grep -v "grep"`
done
```

that's the verbouse version that helped me to understand that i will always see 2 processes when i populate my $children variable through an external command..

the more compact version is this:

```
self=`basename "$0"`
children=`ps aux | grep "$self" | grep -vc "grep"`
while [ $children -gt 2 ]; do
    sleep 1
    echo "still have $children processes, keep waiting: "
    children=`ps aux | grep "$self" | grep -vc "grep"`
done
```