Thread implementation in shell script

When you execute a Bash script, it will at maximum use a single CPU thread, unless you start subshells/threads. If your machine has at least two CPU threads, you will be able to max-out CPU resources using multi-threaded scripting in Bash. The reason for this is simple; as soon as a secondary 'thread' (read: subshell) is started, then that subsequent thread can (and often will) use a different CPU thread.

& operator

- In linux, this operator make shell to run the command in the background, that is, it is forked and run in a separate sub-shell, as a job, asynchronously
- Running command or script using &, that process will be run in background so that it is possible to keep doing other things in current shell.
- Examples are below. Just add & to end of command.

Below example shows how & operator works in the linux shell

```
$ sleep 10 &
[1] 79287
$ # Type enter to make below result
[1]+  Done                    sleep 10
```

wait

wait is a built-in command of Linux that waits for completing any running process. wait command is used with a particular process id or job id. When multiple processes are running in the shell then only the process id of the last command will be known by the current shell. If wait command is executed this time, then it will be applied for the last command. If no process id or job id is given with wait command then it will wait for all current child processes to complete and returns exit status.

Below is the example

```
first_command &
second_command &

wait
```

Sample 1)

```
#!/bin/bash

function back_ground_process () {
        # Random number between 10 and 15
        sleep_time=$(($RANDOM*12345%6 + 10))

        echo "${1} will sleep for ${sleep_time} seconds"

        sleep ${sleep_time}

        echo "${1} sleep done!"
}

# array in shell script
arr=("a_worker" "b_worker" "c_worker")

# @ means all elements in array
for i in ${arr[@]}; do
    # run back_ground_process function in background
    # pass element of array as argument
    # make log file
    back_ground_process $i > ~/log_${i}.txt &
done

# wait until all child processes are done
wait

echo "All background processes are done!"
```