- **wait**() system call suspends execution of the calling process until one of its children terminates
- **wait**(): on success, returns the process ID of the terminated child; on error, -1 is returned
- **waitpid**() suspends execution of the calling process until a child specified by *pid* argument has changed state. By default, **waitpid**() waits only for terminated children, but this behavior is modifiable via the *options* ; *wait(&status)* is equivalent to:waitpid(-1, &status, 0);
- < -1 meaning wait for any child process whose process group ID is equal to the absolute value of *pid*.
- -1 meaning wait for any child process.
- > 0 meaning wait for the child whose process ID is equal to the value of *pid*.

```
pid_t child_pid, wpid;
int status = 0;

//Father code (before child processes start)

for (int id=0; id<n; id++) {
    if ((child_pid = fork()) == 0) {
        //child code
        exit(0);
    }
}

while ((wpid = wait(&status)) > 0); // this way, the father waits for all the child process

//Father code (After all child processes end)
```

```
pid_t childPid;  // the child process that the execution will soon run inside of.
childPid = fork();

if(childPid == 0)  // fork succeeded
{
   // Do something
   exit(0);
}

else if(childPid < 0)  // fork failed
{
   // log the error
}

else  // Main (parent) process after fork succeeds
{
    int returnStatus;
    waitpid(childPid, &returnStatus, 0);  // Parent process waits here for child to termina

    if (returnStatus == 0)  // Verify child process terminated without error.
    {
       printf("The child process terminated normally.");
    }

    if (returnStatus == 1)
    {
       printf("The child process terminated with an error!.");
    }
}
```