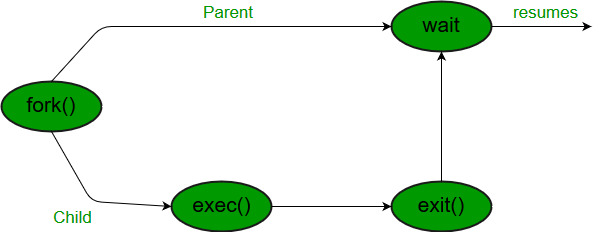
A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction.   
Child process may terminate due to any of these:

* It calls exit();
* It returns (an int) from main
* It receives a signal (from the OS or another process) whose default action is to terminate.



**Syntax in c language:**

|  |
| --- |
| #include  #include  // take one argument status and returns  // a process ID of dead children.  pid\_t wait(int \*stat\_loc); |

If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.   
If only one child process is terminated, then return a wait() returns process ID of the terminated child process.   
If more than one child processes are terminated than wait() reap any ***arbitrarily child*** and return a process ID of that child process.   
When wait() returns they also define **exit status** (which tells our, a process why terminated) via pointer, If status are not **NULL**.  
If any process has no child process then wait() returns immediately “-1”.

**Child status information:**   
Status information about the child reported by wait is more than just the exit status of the child, it also includes

* normal/abnormal termination
* termination cause
* exit status

For find information about status, we use   
**WIF**….macros  
1. **WIFEXITED(status)**: child exited normally   
• **WEXITSTATUS(status)**: return code when child exits  
2. **WIFSIGNALED(status)**: child exited because a signal was not caught   
• **WTERMSIG(status)**: gives the number of the terminating signal  
3. **WIFSTOPPED(status)**: child is stopped   
• **WSTOPSIG(status)**: gives the number of the stop signal