# wait(), waitpid() — Quick Notes (Linux/Unix)

The **wait family** of system calls lets a parent process:

- Collect the **termination status** of a child process
- Prevent zombie processes
- Optionally wait for a **specific child**

After a child exits → it becomes a **zombie** until the parent calls `wait()` / `waitpid()`.

---

## ✔️Core Behavior

```
fork()    → creates child
child     → finishes / exits
temporarily becomes ZOMBIE
parent    → calls wait()/waitpid() to collect status
```

Once collected → the entry is removed from the process table.

---

## 🛅Function Prototypes

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
```

Return value: - child PID on success - `-1` on error (and sets `errno`)

`wstatus` may be `NULL` if status is not required.

---

## ✅wait()

Waits for **any terminated child**.

- Blocks until *one* child finishes
- Does **not** choose which child

```c
pid_t child = wait(&status);
```

---

## ✅waitpid()

Supports **selective & non-blocking waiting**.

```c
pid_t waitpid(pid_t pid, int *status, int options);
```

| pid value | Meaning |
|---|---|
| `> 0` | Wait for that specific child PID |
| `-1` | Wait for **any child** (like wait()) |
| `0` | Any child in same process group |
| `< -1` | Any child in process group = `abs(pid)` |

**options flags** (bit-wise OR):

| Option | Meaning |
|---|---|
| `0` | Blocking wait |
| `WNOHANG` | Non-blocking → return immediately |
| `WUNTRACED` | Report stopped children |
| `WCONTINUED` | Report resumed children |

---

## 9 Status Inspection Macros

Used on `status` returned by wait/waitpid.

```c
WIFEXITED(status)     // child terminated normally
WEXITSTATUS(status)   // exit() status code

WIFSIGNALED(status)   // child killed by signal
WTERMSIG(status)      // signal number

WIFSTOPPED(status)    // child stopped
WSTOPSIG(status)      // signal that stopped it
```

```
WIFCONTINUED(status)  // child resumed (SIGCONT)
```

Use these macros instead of manually decoding bits.

## ✅Example 1 — Basic wait()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        printf("Child running...\n");
        sleep(2);
        printf("Child exiting with status 5\n");
        exit(5);
    }

    int status;
    pid_t c = wait(&status);

    printf("Parent: collected child %d\n", c);

    if (WIFEXITED(status))
        printf("Exit status = %d\n", WEXITSTATUS(status));
}
```

## ✅Example 2 — waitpid() for a Specific Child

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t p1 = fork();
    if (p1 == 0) { sleep(3); exit(10); }
    pid_t p2 = fork();
    if (p2 == 0) { sleep(1); exit(20); }

    int status;

    // wait only for p2 first
    waitpid(p2, &status, 0);
    printf("Collected p2 → %d\n", WEXITSTATUS(status));

    // now collect remaining child
    waitpid(p1, &status, 0);
    printf("Collected p1 → %d\n", WEXITSTATUS(status));

    return 0;
}
```

---

## ✅Example 3 — Non-Blocking waitpid() (WNOHANG)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        sleep(4);
        exit(7);
    }

    int status;

    while (1) {
        pid_t r = waitpid(pid, &status, WNOHANG);

        if (r == 0) {
            printf("Working… child not finished yet\n");
            sleep(1);
        } else if (r > 0) {
            printf("Child %d exited (%d)\n", r, WEXITSTATUS(status));
            break;
        } else {
            perror("waitpid");
            break;
        }
    }

    return 0;
}
```

---

## ✅Additional Examples (Advanced Scenarios)

### Example 4 — Reaping Multiple Children in a Loop

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    for (int i = 0; i < 3; i++) {
        pid_t pid = fork();

        if (pid == 0) {
            printf("Child %d (pid=%d) exiting...\n", i, getpid());
            exit(10 + i);
        }
    }

    int status;
    pid_t pid;

    while ((pid = wait(&status)) > 0) {
        if (WIFEXITED(status))
            printf("Reaped child %d → exit status %d\n", pid,
WEXITSTATUS(status));
    }

    printf("No more children.\n");
    return 0;
}
```

## Example 5 — Detecting Signal-Termination (WIFSIGNALED)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        printf("Child will be killed using SIGKILL\n");
        raise(SIGKILL);
        return 0;
    }

    int status;
    waitpid(pid, &status, 0);

    if (WIFSIGNALED(status))
        printf("Child terminated by signal %d\n", WTERMSIG(status));

    return 0;
}
```

## Example 6 — Non-Blocking Polling Loop (WNOHANG)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        printf("Child sleeping 4 seconds...\n");
        sleep(4);
        exit(7);
    }

    int status;

    while (1) {
        pid_t r = waitpid(pid, &status, WNOHANG);

        if (r > 0) {
            printf("Child %d exited (%d)\n", r, WEXITSTATUS(status));
            break;
        } else if (r == 0) {
            printf("Working… child not finished yet\n");
            sleep(1);
        } else {
            perror("waitpid");
            break;
        }
    }

    return 0;
}
```

## Example 7 — Waiting for Stopped & Continued Processes

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        raise(SIGSTOP);
        sleep(1);
        raise(SIGCONT);
        exit(0);
    }

    int status;
    waitpid(pid, &status, WUNTRACED | WCONTINUED);

    if (WIFSTOPPED(status))
        printf("Child stopped (signal %d)\n", WSTOPSIG(status));

    kill(pid, SIGCONT);

    waitpid(pid, &status, WUNTRACED | WCONTINUED);

    if (WIFCONTINUED(status))
```

```
        printf("Child continued\n");

    waitpid(pid, &status, 0);
    printf("Child finished\n");

    return 0;
}
```

## Example 8 — Handling ECHILD Gracefully

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/wait.h>

int main() {
    int status;
    pid_t r = waitpid(-1, &status, 0);

    if (r == -1 && errno == ECHILD)
        printf("No child processes to reap\n");

    return 0;
}
```

## Example 9 — Parent Exits First (Orphan Adoption)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        sleep(3);
        printf("Child adopted by init/systemd (pid=%d)\n", getpid());
        exit(0);
    }

    printf("Parent exiting immediately (pid=%d)\n", getpid());
    exit(0);
}
```