waitpid()

❓ The “status” integer returned by `waitpid()` is not just a simple exit code. It encodes multiple pieces of information about how the child process terminated, and interpreting it requires using specific macros provided by the system.

The `waitpid()` function is used to wait for a specific child process to change its state (e.g. terminate or stop).

It’s commonly used in parent processes to synchronize with child processes and retrieve their exit status.

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

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

- “pid” is the process ID of the child process to wait for
  - It has different interpretations:
    - “-1”: wait for any child process

        - “0”: wait for any child process in the same process group as the calling process
        - “> 0”: wait for the child process with the specified PID
- “status” is a pointer to an integer where the exit status of the child process will be stored
    - Is a bitfield that encodes:
        - The exit status of the child process if exited normally
        - The signal number that caused the child process to terminate if it was killed by a signal
        - Additional information about whether the child process was stopped, continued, or dumped core
    - The raw value of “status” is not meaningful on its own, it must be decoded using the appropiate macros
    - The macros provided by <sys/wait.h> are:
        - WIFEXITED(status)
            - Returns “true” if the child process exited normally (e.g. via “exit()” or returning from “main”)

```
if (WIFEXITED(status))
    printf("Child exited normally.\n");
```

        - WEXITSTATUS(status)
            - If “WIFEXITED(status)” is true, this macro extracts the exit code of the child process (the value passed to “exit()” or returned from “main”)

```
if (WIFEXITED(status))
    printf("Child exit code: %d\n", WEXITSTATUS(status));
```

        - WIFSIGNALED(status)

- Returns “true” if the child process was terminated by a signal

```
if (WIFSIGNALED(status))
    printf("Child terminated by a signal.\n");
```

- WTERMSIG(status)
  - If “WIFSIGNALED(status)” is “true”, this macro extracts the signal number that caused the child process to terminate

```
if (WIFSIGNALED(status))
    printf("Terminating signal: %d\n", WTERMSIG(status));
```

- WCOREDUMP(status)
  - If “WIFSIGNALED(status)” is true, this macro checks whether the child process generated a core dump

```
if (WCOREDUMP(status))
    printf("Child produced a core dump.\n");
```

- WIFSTOPPED(status)
  - Returns “true” if the child process was stopped by a signal (e.g. via “SIGSTOP” or “SIGTSTP”)

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

- WSTOPSIG(status)
  - If “WIFSTOPPED(status)” is true, this macro extracts the signal number that caused the child process to stop

- “options” are flags to modify the behavior of “waitpid()”
  - Common options include:
    - “WNOHANG”: return immediately if no child process has exited

- “WUNTRACED”: also return if a child process has stopped (but not terminated)
- “0”: default behavior (wait for the child process to terminate)

On success, it returns the PID of the child process whose state has changed. If “WNOHANG” is specified and no child process has exited, it returns 0.

On failure, it returns -1 and sets “errno” to indicate the error.