POSIX IEEE Std 1003.1 2017 definitions related to processes (XBD, sec. 3)

Process

- **Live process**: An address space with one or more threads executing within that address space, and the required system resources for those threads. Note: Many of the system resources defined by POSIX.1-2017 are shared among all of the threads within a process. These include the process ID, the parent process ID, process group ID, session membership, real, effective, and saved set-user-ID, real, effective, and saved set-group-ID, supplementary group IDs, current working directory, root directory, file mode creation mask, and file descriptors.
- **Zombie process**: The remains of a live process and before its status information is consumed by its parent process.

Child Process

A new process created (by fork( ), posix_spawn( ), or posix_spawnp( )) by a given process. A child process remains the child of the creating process as long as both processes continue to exist.

Parent Process

The process which created (or inherited) the process under discussion.

Process lifetime

The period of time that begins when a process is created and ends when its process ID is returned to the system.

Process termination

There are two kinds of process termination:

1. **Normal termination** occurs by a return from main( ), when requested with the exit( ), _exit( ), or _Exit( ) functions; or when the last thread in the process terminates by returning from its start function, by calling the pthread_exit( ) function, or through cancellation.
2. **Abnormal termination** occurs when requested by the abort( ) function or when some signals are received.

Consequences of Process Termination

- All of the file descriptors, directory streams, conversion descriptors, and message catalog descriptors open in the calling process shall be closed.
- If the parent process of the calling process has set its SA_NOCLDWAIT flag or has set the action for the SIGCHLD signal to SIG_IGN:
  - The process' status information, if any, shall be discarded.
  - The lifetime of the calling process shall end immediately. If SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to the parent process.
  - If a thread in the parent process of the calling process is blocked in wait( ), waitpid( ), or waitid( ), and the parent process has no remaining child processes in the set of waited-for children, the wait( ), waitid( ), or waitpid( ) function shall fail and set errno to [ECHILD].

  Otherwise:
  - Status information shall be generated.
  - The calling process shall be transformed into a zombie process. Its status information shall be made available to the parent process until the process' lifetime ends.
  - The process' lifetime shall end once its parent obtains the process' status information via a currently-blocked or future call to wait( ), waitid( ) (without WNOWAIT), or waitpid( ).
  - If one or more threads in the parent process of the calling process is blocked in a call to wait( ), waitid( ), or waitpid( ) awaiting termination of the process, one (or, if any are calling waitid( ) with WNOWAIT, possibly more) of these threads shall obtain the process' status information and become unblocked.
  - A SIGCHLD shall be sent to the parent process.
- Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.
- The parent process ID of all of the existing child processes and zombie processes of the calling process shall be set to the process ID of an implementation-defined system process. That is, these processes shall be inherited by a special system process (Comment: UNIX: init).
- ….. See more details in the man page for the _Exit() function/

Status information

Status information is data associated with a process detailing a change in the state of the process. It shall consist of:

- The state the process transitioned into (stopped, continued, or terminated)
- The information necessary to populate the siginfo_t structure provided by waitid( )
- If the new state is terminated:
  - The low-order 8 bits of the status argument that the process passed to _Exit( ), _exit( ), or exit( ), or the low-order 8 bits of the value the process returned from main( ). Note that these 8 bits are part of the complete value that is used to set the si_status member of the siginfo_t structure provided by waitid( ).
  - Whether the process terminated due to the receipt of a signal that was not caught

and, if so, the number of the signal that caused the termination of the process

- If the new state is stopped:
  - The number of the signal that caused the process to stop

A process might not have any status information (such as immediately after a process has started).

Status information for a process shall be generated (made available to the parent process) when the process stops, continues, or terminates except in the following case:

- If the parent process sets the action for the SIGCHLD signal to SIG_IGN, or if the parent sets the SA_NOCLDWAIT flag for the SIGCHLD signal action, process termination shall not generate new status information but shall cause any existing status information for the process to be discarded.

If new status information is generated, and the process already had status information, the existing status information shall be discarded and replaced with the new status information.

Only the process' parent process can obtain the process' status information. The parent obtains a child's status information by calling wait( ), waitid( ), or waitpid( ). Except when waitid( ) is called with the WNOWAIT flag set in the options argument, the status information obtained by a wait function shall be consumed (discarded) by that wait function; no two calls to wait( ), waitid( ) (without WNOWAIT), or waitpid( ) shall obtain the same status information.

Process Group

A collection of processes that permits the signaling of related processes. Each process in the system is a member of a process group that is identified by a process group ID. A newly created process joins the process group of its creator.

Orphaned Process Group

A process group in which the parent of every member is either itself a member of the group or is not a member of the group's session.

Session

A collection of process groups established for job control purposes. Each process group is a member of a session. A process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership; see setsid( ). There can be multiple process groups in the same session.

Job

A set of processes, comprising a shell pipeline, and any processes descended from it, that are all in the same process group.

Job Control

A facility that allows users selectively to stop (suspend) the execution of processes and continue (resume) their execution at a later point. The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter.

Foreground Process Group (or Foreground Job)

A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has at most one process group of the session as the foreground process group of that controlling terminal.

Controlling Process
The session leader that established the connection to the controlling terminal. If the terminal subsequently ceases to be a controlling terminal for this session, the session leader ceases to be the controlling process.

Controlling Terminal
A terminal that is associated with a session. Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to all processes in the foreground process group associated with the controlling terminal.

Multi-Threaded Program

A program whose executable file was produced by compiling with **c99** using the flags output by getconf POSIX_V7_THREADS_CFLAGS, and linking with **c99** using the flags output by getconf POSIX_V7_THREADS_LDFLAGS and the **–l pthread** option, or by compiling and linking using a non-standard utility with equivalent flags. Execution of a multi-threaded program initially creates a single-threaded process; the process can create additional threads using pthread_create( ) or SIGEV_THREAD notifications.