Operating Systems (02340123)
Functions Reference - Spring 2025

Razi & Yara

June 7, 2025

Contents

I Functions Reference **3**

Topic 1: Process Management 4

II Cheat Sheet **7**

Part I

Functions Reference

Topic 1: Process Management

`fork`

Declaration: `pid_t fork();`
Usage/Explanation: Copies the parent process to the child process and returns with the two processes.

Same code, Same memory, Same environment (files, etc.). But they are separate processes with separate memory spaces and different PIDs.
Parameters: None
Return Values: Returns 0 to child, PID of child to parent, -1 on error.
Additional:

`wait`

Declaration: `pid_t wait(int *wstatus);`
Usage/Explanation: Waits until **any** child process ends, suspending the calling process until a child terminates
Parameters: `wstatus`: Pointer to the variable where the exit status of the child will be stored. If NULL, no status is returned.

To get the value of the status, you can use macros like `WEXITSTATUS(*wstatus)` which return the second byte of the variable, where the exit code of the child is stored.
Return Values: If there are no children or all children have already terminated and waited for, it returns -1. Else waits until a child process ends and returns its PID.
Additional: Can only wait for direct child!

`waitpid`

Declaration: `pid_t waitpid(pid_t pid, int *wstatus, int options);`
Usage/Explanation: Wait until a specific child process ends.
Parameters:

- `pid`: The PID of the child process to wait for. If -1, waits for any child process.

- `wstatus`: Pointer to an integer where the exit status of the child will be stored. If NULL, no status is returned.
- `options`: Options for waiting, such as WNOHANG (do not block if no child has exited). default is 0, which blocks until the child exits.

Return Values: Returns the PID of the child that exited, or -1 on error. If WNOHANG is set and no child has exited, it returns 0.
Additional:

`exit`

Declaration: `void exit(int status);`
Usage/Explanation: Terminates the calling process and releases all of its recourses. The process becomes *zombie* until its parent process requests to check its termination (e.g. wait()) and then clears completely.
Parameters: `status`: The exit status of the process which is returned to the parent process when checked.
Return Values: No return value. The process is terminated immediately, and will never fail.
Additional: The `main()` is not in fact the main function of the process, it is wrapped by `int __libc_start_main()` who collects the return value of `main()` and calls `exit()` with it. This is why we don't use `exit()` in `main()` but rather return from it usually $\Longrightarrow$ `exit` is always called.

`execv`

Declaration: `int execv(const char *filename, char *const argv[]);`
Usage/Explanation: Replaces the current running process code with a new program. (same PID,PPID but different code and memory).
Parameters:

- `filename`: The path to the file containing the program to execute.
- `argv`: An array of pointers to null-terminated strings containing the parameters to pass to the new program. The first element is the name of the process, i.e. `argv[0]=filename`. The last argument must be `NULL` to indicate the end of the array.

Return Values: Returns -1 on error, and does not return on success as the current process is replaced by the new program.
Additional: The `execv()` function is one of the *exec* family of functions, which replace the current process image with a new process image. It does not create a new process; it replaces the current one. The `v` stands for the array of arguments, `p` is for searching in the `PATH` environment variable for the filename.

`getpid, getppid`

Declaration: `pid_t getpid();`
`pid_t getppid();`
Usage/Explanation: `getpid()` returns the PID of the calling process, and `getppid()` returns the PID of the parent process (not real parent).
Parameters: None
Return Values: Returns the PID of the calling process or its parent accordingly.
Additional:

Part II

Cheat Sheet

Function	**Summary**
`fork()`	Creates a new child process, identical to the parent initially.
`wait()`	Suspends execution until a child process ends.