forking

characteristics

- inherits all characteristics from its father
  - variables
  - program counter
  - open files
  - ...
- spawned using `fork()` command
- after forking, each process will proceed to its own instructions and every change in any variable will only be seen by the process who made it

functions

```
pid_t getpid(void)
// returns the PID of the current process
```

```
pid_t getppid(void)
// returns the PID of the father process
```

```
pid_t fork(void)
// spawns a new child process
```

return values

- 0 to the child process
- child's PID to the original process
- -1 if an error occurred

```
pid_t wait(int *status)
// wait until a child process exits
```

parameters

- **status**: keeps the return value of the child process that has been waited
  return values
- PID of the process that has been waited

- -1 if an error occurred

```
pid_t waitpid(pid_t who, int *status, int options)
// wait until a specific child exits
```

parameters

- `who` is the PID of the child process to wait for
- using `WNOHANG` in `options` to do `wait` to a specific process without blocking
- using `0` in `options` means 'wait for any child'
  return values
- -1 if an error occurred

process termination

proper elimination

only when the father calls wait() or waitpid()

zombie process

the child process dies but the father does not do wait(), becoming a zombie
it will keep consuming resources

orphan process

the original parent dies before tje child process, so the child process becomes adopted by `init` the (process 1) (or by another sub-reaper that fulfills the role of `init` for its descendant processes)

exec functions' family

- these functions allow the OS to execute code starting from an executable
- they substitute the current process executable image by another one (complete substitution - code, stack and heap)
- a successful `exec()` never returns

functions

```c
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *file, const char *arg, ..., char *const envp[]);
int execv(const char *path, const char *arg[]);
int execvp(const char *file, const char *arg[]);
```

- p: make use of environment PATH
- v: make use of a pointer to an array of arguments
- l: parameters passed separated by commas
- the first parameter must be the name of the program