Process Control

Every process has a unique **process ID**, a non-negative integer.
Some special processes: Process ID 0 – **swapper**, Process ID 1 – **init**

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

//PROCESS ID
pid_t getpid(void);    //Returns: process ID of calling process
pid_t getppid(void);   //Returns: parent process ID of calling process

uid_t getuid(void);    //Returns: real user ID of calling process
uid_t geteuid(void);   //Returns: effective user ID of calling process

gid_t getgid(void);    //Returns: real group ID of calling process
gid_t getegid(void);   //Returns: effective group ID of calling process
```

The fork Function

The only way a new process is created by the UNIX kernel is when an existing process calls the fork function (this doesn't apply to special processes). The new process created by fork is called the *child process*. This function is called once but returns twice.

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

pid_t fork(void);             //Returns: 0 in child, process ID of child in parent, -1 on error
```

The wait and waitpid Functions

When a process terminates, either normally or abnormally, the parent is notified by the kernel. The termination of a child process is an asynchronous event (it can happen anytime while the parent is running). A process that calls wait or waitpid can:

- block (if all its children are still running), or
- return immediately with the termination status of a child (if a child has terminated and is waiting for its termination status to be fetched), or
- return immediately with an error (if it doesn't have any child processes).

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

pid_t wait(int *statloc);    //statloc - pointer to an integer (termination status)

pid_t waitpid(pid_t pid, int *statloc, int options);
                             //Both return: process ID if OK, 0, or -1 on error
```

The exec Functions

When a process calls on the exec functions, that process is completely replaced by the new program, and the new program starts executing at its main function. The process ID does not change across an exec because a new process is not created. exec merely replaces the current process (its text, data, heap, and stack segments) with a brand new program from disk.

```
#include<unistd.h>

int execl(const char *pathname, const char *arg0, ... /* (char *) 0 */);
int execv(const char *pathname, char *const argv[]);
int execle(const char *pathname, const char *arg0, ... /* (char *) 0, char* const envp[] */);
int execve(const char *pathname, char *const argv[], char *const envp[]);
int execlp(const char *filename, const char *arg0, ... /* (char *) 0 */);
int execvp(const char *filename, char *const argv[]);
                             //All return: -1 on error, no return on success
```

Reference:
Stevens, W. R. Advanced Programming in the UNIX Environment. Addison-Wesley. 1996.