Process

- has unique ID of its own. [pid]
- is the unit of single programme flow, which is on memory.
- parallel running is disabled.

fork():

- is a system call in Linux, not in Windows.
- creates a new process[child].
- copies the exact state[parent] and give it to new process[child].
- Two processese [parent, child] will run separately.

Parent Process

- The process that calls `fork()`.

Child Process

- Newly created process by `fork()`.

Uses

- Parallel executions of tasks.
- To make the programme not blocked by the tasks independent to the main process and heavy.

Example: web server

- Handling a request of clients which are independent (like get).

Danger: Zombie process

- Occurs given the parent process end without waiting for the child process's end.

Prevent: `wait()`

- The parent is able to call `wait()` to prevent the zomie process.
- Blocks the process until the one of the children processes comes to end.
- Returns the id of the process which came to end.
- Returns -1 when no child process has found.

Programme writing

> Write Linux program using fork() to prepare an unbalanced-tree with left-sub-tree (LST) containing 3 nodes and right-sub-tree (RST) containing 6 nodes. See process-tree in figure.

> Measure total execution time and plot graph (X axis: instances of execution (execute 5 times of your program) and Y axis: measured execution time for each instance of execution)

Summary of the tree:

```
ROOT
    └LST
        └LST1
        └RST1
    └RST
        └LST2
            └NODE0
            └NODE1
                └NODETAIL
            └NODE2
```

I decided to manually controll the flow, so this is how I made this.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // for fork, pids, things
#include <time.h>   // for time getting
#include <wait.h>   // for... wait().
```

> If the process is not root or not
> 0 when current process is root.

```c
int nroot = 0;
```

> give process a nickname
>
> execute the fork().
>
> prints to stdout as tagname, pid(current), pid(parent for current).

> zero if current is child.
> otherwise will be parent.

```c
pid_t mkproc(const char *name) {
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork failed\n");
        exit(1);
```

```
    } else if (pid == 0) { // Child process
        printf("%s: %d => %d\n", name, getppid(), getpid());
        nroot = 1; // now the process is not root process
    }

    return pid;
}
```

> Waits them all.
>
> if it is not root, process will be terminated.
>
> for beginner's safety.

```
void waitall() {
    while(wait(0) != -1);

    // since it is not a root process, the process shall meet the end
    if(nroot) return exit(0);
}
```

> just making left subtree

```
void LST() {
    if(!mkproc("LST")) {
        mkproc("LST1") && mkproc("RST1");
    }

    waitall();
}
```

```
/// @brief
/// just making right subtree
void RST() {
    if(mkproc("RST")) goto end;
    if(mkproc("LST2")) goto end;
    if(!mkproc("NODE0")) goto end;

    mkproc("NODE1") ? mkproc("NODE2") : mkproc("NODETAIL");

    end:
    waitall();
}
```

```
#define NS_TO_MS 1000000.0
```

```
int main() {
    struct timespec start, end;
    double elapsed;

    timespec_get(&start, TIME_UTC);

    printf("Process start, ROOT: %d\n", getpid());
    LST(); RST();

    timespec_get(&end, TIME_UTC);
    printf("Process end, time: %lf (ms)\n", (double)(end.tv_nsec -
start.tv_nsec) / NS_TO_MS);

    return 0;
}
```

The result for each iteration

```
ae2f@fedora:~/Documents/GitHub/---/d$ ./a.out
Process start, ROOT: 40936
LST: 40936 => 40937
LST1: 40937 => 40938
RST1: 40937 => 40939
RST: 40936 => 40940
LST2: 40940 => 40941
NODE0: 40941 => 40942
NODE1: 40941 => 40943
NODE2: 40941 => 40945
NODETAIL: 40943 => 40944
Process end, time: 3.881086 (ms)
ae2f@fedora:~/Documents/GitHub/---/d$ ./a.out
Process start, ROOT: 40948
LST: 40948 => 40949
LST1: 40949 => 40950
RST1: 40949 => 40951
RST: 40948 => 40952
LST2: 40952 => 40953
NODE0: 40953 => 40954
NODE1: 40953 => 40955
NODE2: 40953 => 40956
NODETAIL: 40955 => 40957
Process end, time: 3.017097 (ms)
ae2f@fedora:~/Documents/GitHub/---/d$ ./a.out
Process start, ROOT: 40960
LST: 40960 => 40961
LST1: 40961 => 40962
RST1: 40961 => 40963
RST: 40960 => 40964
LST2: 40964 => 40965
NODE0: 40965 => 40966
NODE1: 40965 => 40967
NODE2: 40965 => 40968
NODETAIL: 40967 => 40969
Process end, time: 2.946905 (ms)
```

```
ae2f@fedora:~/Documents/GitHub/---/d$ ./a.out
Process start, ROOT: 40902
LST: 40902 => 40903
LST1: 40903 => 40904
RST1: 40903 => 40905
RST: 40902 => 40906
LST2: 40906 => 40907
NODE0: 40907 => 40908
NODE1: 40907 => 40909
NODE2: 40907 => 40910
NODETAIL: 40909 => 40911
Process end, time: 3.482113 (ms)
ae2f@fedora:~/Documents/GitHub/---/d$ ./a.out
Process start, ROOT: 40924
LST: 40924 => 40925
```

```
LST1: 40925 => 40926
RST1: 40925 => 40927
RST: 40924 => 40928
LST2: 40928 => 40929
NODE0: 40929 => 40930
NODE1: 40929 => 40931
NODETAIL: 40931 => 40933
NODE2: 40929 => 40932
Process end, time: 3.262930 (ms)
```

The summary graph