Everything you need to know to start coding your own shell

PID & PPID

A process is an instance of an executing program, that has a unique process ID. This process ID is used by many functions and system calls to interact with and manipulate processes. In order to retrieve the current process' ID, you can use the system call `getpid` (man 2 `getpid`):

```
julien@ubuntu:~/c/shell$ cat pid.c
#include <stdio.h>
#include <unistd.h>

/**
 * main - PID
 *
 * Return: Always 0.
 */
int main(void)
{
    pid_t my_pid;

    my_pid = getpid();
    printf("%u\n", my_pid);
    return (0);
}
julien@ubuntu:~/c/shell$ gcc -Wall -Werror -pedantic pid.c -o mypid && ./mypid
3237
julien@ubuntu:~/c/shell$ ./mypid
3238
julien@ubuntu:~/c/shell$ ./mypid
3239
julien@ubuntu:~/c/shell$
```

Note in the example above, that every time you run the program, a new process is created, and its ID is different.

Each process has a parent: the process that created it. It is possible to get the PID of a parent process by using the `getppid` system call (man 2 `getppid`), from within the child process.

Exercises

0. getppid

Write a program that prints the PID of the parent process. Run your program several times within the same shell. It should be the same. Does `echo $$` print the same value? Why?

1. /proc/sys/kernel/pid_max

Write a shell script that prints the maximum value a process ID can be.

Arguments

The command line arguments are passed through the `main` function: `int main(int ac, char **av);`

- `av` is a `NULL` terminated array of strings
- `ac` is the number of items in `av`

`av[0]` usually contains the name used to invoke the current program. `av[1]` is the first argument of the program, `av[2]` the second, and so on.

Exercises

0. av

Write a program that prints all the arguments, without using `ac` .

1. Read line

Write a program that prints `"$ "` , wait for the user to enter a command, prints it on the next line.

man 3 `getline`

important make sure you read the man, and the RETURN VALUE section, in order to know when to stop reading Keyword: “end-of-file”, or `EOF` (or `Ctrl+D`).

#advanced: Write your own version of `getline` .

```
julien@ubuntu:~/c/shell$ gcc -Wall -Wextra -Werror -pedantic prompt.c -o prompt
julien@ubuntu:~/c/shell$ ./prompt
$ cat prompt.c
cat prompt.c
julien@ubuntu:~/c/shell$
```

2. command line to av

Write a function that splits a string and returns an array of each word of the string.

man `strtok`

#advanced: Write the function without `strtok`

Executing a program

The system call `execve` allows a process to execute another program (man 2 `execve`). Note that this system call does load the new program into the current process' memory in place of the "previous" program: on success `execve` does not return to continue the rest of the "previous" program.

Warning: in this example, `execve` is used without the current environment (last argument), don't forget to add it in your Shell!

```
julien@ubuntu:~/c/shell$ cat exec.c
#include <stdio.h>
#include <unistd.h>

/**
 * main - execve example
 *
 * Return: Always 0.
 */
int main(void)
{
    char *argv[] = {"/bin/ls", "-l", "/usr/", NULL};

    printf("Before execve\n");
    if (execve(argv[0], argv, NULL) == -1)
    {
        perror("Error:");
    }
    printf("After execve\n");
    return (0);
}
julien@ubuntu:~/c/shell$ gcc -Wall -Wextra -Werror -pedantic exec.c -o exec
julien@ubuntu:~/c/shell$ ./exec
Before execve
total 120
drwxr-xr-x   2 root root 61440 Dec  4 00:08 bin
drwxr-xr-x   2 root root  4096 Jul 19 13:47 games
drwxr-xr-x  58 root root  4096 Oct 27 13:10 include
drwxr-xr-x 138 root root  4096 Dec  4 00:08 lib
drwxr-xr-x   3 root root  4096 Nov 10 09:54 lib32
drwxr-xr-x   3 root root  4096 Nov 10 09:54 libx32
drwxr-xr-x  10 root root  4096 Jul 19 13:42 local
drwxr-xr-x   3 root root  4096 Jul 19 13:48 locale
drwxr-xr-x   2 root root 12288 Dec  2 11:03 sbin
drwxr-xr-x 295 root root 12288 Jul 27 08:58 share
drwxr-xr-x   6 root root  4096 Dec  1 11:39 src
julien@ubuntu:~/c/shell$
```

Creating processes

The system call `fork` (man 2 `fork`) creates a new child process, almost identical to the parent (the process that calls `fork`). Once `fork` successfully returns, two processes continue to run the same program, but with different stacks, datas and heaps.

```
julien@ubuntu:~/c/shell$ cat fork.c
#include <stdio.h>
#include <unistd.h>

/**
 * main - fork example
 *
 * Return: Always 0.
 */
int main(void)
{
    pid_t my_pid;
    pid_t pid;

    printf("Before fork\n");
    pid = fork();
    if (pid == -1)
    {
        perror("Error:");
        return (1);
    }
    printf("After fork\n");
    my_pid = getpid();
    printf("My pid is %u\n", my_pid);
    return (0);
}
julien@ubuntu:~/c/shell$ ./fork
Before fork
After fork
My pid is 4819
julien@ubuntu:~/c/shell$ After fork
My pid is 4820
```

Note: there is no typo in the above example

Using the return value of `fork`, it is possible to know if the current process is the father or the child: `fork` will return `0` to the child, and the PID of the child to the father.

```
julien@ubuntu:~/c/shell$ cat fork.c
#include <stdio.h>
#include <unistd.h>

/**
 * main - fork example
 *
 * Return: Always 0.
 */
int main(void)
{
    pid_t my_pid;
    pid_t child_pid;

    child_pid = fork();
    if (child_pid == -1)
    {
        perror("Error:");
        return (1);
    }
    my_pid = getpid();
    printf("My pid is %u\n", my_pid);
    if (child_pid == 0)
    {
        printf("(%u) Nooooooooo!\n", my_pid);
    }
    else
    {
        printf("(%u) %u, I am your father\n", my_pid, child_pid);
    }
    return (0);
}
julien@ubuntu:~/c/shell$ gcc -Wall -Wextra -Werror -pedantic fork.c -o fork
julien@ubuntu:~/c/shell$ ./fork
My pid is 4869
(4869) 4870, I am your father
julien@ubuntu:~/c/shell$ My pid is 4870
(4870) Nooooooooo!
```

Wait

The `wait` system call (man 2 `wait`) suspends execution of the calling process until one of its children terminates.

```
julien@ubuntu:~/c/shell$ cat wait.c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

/**
 * main - fork & wait example
 *
 * Return: Always 0.
 */
int main(void)
{
    pid_t child_pid;
    int status;

    child_pid = fork();
    if (child_pid == -1)
    {
        perror("Error:");
        return (1);
    }
    if (child_pid == 0)
    {
        printf("Wait for me, wait for me\n");
        sleep(3);
    }
    else
    {
        wait(&status);
        printf("Oh, it's all better now\n");
    }
    return (0);
}
julien@ubuntu:~/c/shell$ gcc -Wall -Wextra -Werror -pedantic wait.c -o wait
julien@ubuntu:~/c/shell$ ./wait
Wait for me, wait for me
Oh, it's all better now
julien@ubuntu:~/c/shell$
```

Exercise: fork + wait + execve

Write a program that executes the command `ls -l /tmp` in 5 different child processes. Each child should be created by the same process (the father). Wait for a child to exit before creating a new child.

Exercise: super simple shell

Using everything we saw, write a first version of a super simple shell that can run commands with their full path, without any argument.

```
julien@ubuntu:~/c/shell$ l
total 140
drwxrwxr-x  2 julien julien 4096 Dec  4 20:55 .
drwxrwxr-x 17 julien julien 4096 Dec  4 13:04 ..
-rw-rw-r--  1 julien julien  249 Dec  4 13:15 env-environ.c
-rw-rw-r--  1 julien julien  231 Dec  4 13:09 env-main.c
-rwxrwxr-x  1 julien julien 8768 Dec  4 19:52 exec
-rw-rw-r--  1 julien julien  302 Dec  4 19:38 exec.c
-rwxrwxr-x  1 julien julien 8760 Dec  4 20:11 fork
-rw-rw-r--  1 julien julien  438 Dec  4 19:57 fork.c
-rwxrwxr-x  1 julien julien 8656 Dec  4 13:45 mypid
-rw-rw-r--  1 julien julien  179 Dec  4 19:49 pid.c
-rwxrwxr-x  1 julien julien 8656 Dec  4 13:48 ppid
-rw-rw-r--  1 julien julien  180 Dec  4 13:48 ppid.c
-rwxrwxr-x  1 julien julien 8680 Dec  4 13:44 printenv
-rwxrwxr-x  1 julien julien 8760 Dec  4 14:38 prompt
-rwxrwxr-x  1 julien julien 8760 Dec  4 14:38 promptc
-rw-rw-r--  1 julien julien  191 Dec  4 14:17 prompt.c
-rw-rw-r--  1 julien julien  753 Dec  4 20:49 shell.c
-rwxrwxr-x  1 julien julien 8864 Dec  4 20:38 wait
-rw-rw-r--  1 julien julien  441 Dec  4 20:15 wait.c
julien@ubuntu:~/c/shell$ gcc -Wall -Werror -pedantic shell.c -o shell
julien@ubuntu:~/c/shell$ ./shell
#cisfun$ /bin/ls
env-environ.c  exec    fork    mypid  ppid    printenv  promptc   shell    wait
env-main.c     exec.c  fork.c  pid.c  ppid.c  prompt    prompt.c  shell.c  wait.c
#cisfun$ ./ppid
5451
#cisfun$ ./ppid
5451
#cisfun$ ^C
julien@ubuntu:~/c/shell$
```

File information

The `stat` (man 2 stat) system call gets the status of a file. On success, zero is returned. On error, -1 is returned.

```
julien@ubuntu:~/c/shell$ cat stat.c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

/**
 * main - stat example
 *
 * Return: Always 0.
 */
int main(int ac, char **av)
{
    unsigned int i;
    struct stat st;

    if (ac < 2)
    {
        printf("Usage: %s path_to_file ...\n", av[0]);
        return (1);
    }
    i = 1;
    while (av[i])
    {
        printf("%s:", av[i]);
        if (stat(av[i], &st) == 0)
        {
            printf(" FOUND\n");
        }
        else
        {
            printf(" NOT FOUND\n");
        }
        i++;
    }
    return (0);
}
julien@ubuntu:~/c/shell$ ./stat ls /bin/ls /usr/bin/ls
ls: NOT FOUND
/bin/ls: FOUND
/usr/bin/ls: NOT FOUND
julien@ubuntu:~/c/shell$
```

Exercise: find a file in the PATH

Write a program that looks for files in the current `PATH` .

- Usage: `_which filename ...`

Environment

We have seen earlier that the shell uses an environment list, where environment variables are “stored”. The list is an array of strings, with the following format: `var=value`, where `var` is the name of the variable and `value` its value. As a reminder, you can list the environment with the command `printenv`:

```
julien@ubuntu:~/c/shell$ printenv
XDG_VTNR=7
XDG_SESSION_ID=c2
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/julien
SESSION=ubuntu
GPG_AGENT_INFO=/home/julien/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
SHELL=/bin/bash
XDG_MENU_PREFIX=gnome-
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=23068682
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1558
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=julien
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=4
0;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=
01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tl
z=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;
31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;3
1:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;3
1:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;3
5:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=0
1;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pc
x=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:
*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;3
5:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;3
5:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;
36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=0
0;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
QT_ACCESSIBILITY=1
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
SESSION_MANAGER=local/ubuntu:@/tmp/.ICE-unix/1792,unix/ubuntu:/tmp/.ICE-unix/1792
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
PATH=/home/julien/bin:/home/julien/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bi
n:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
DESKTOP_SESSION=ubuntu
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
JOB=dbus
PWD=/home/julien/c/shell_course
XMODIFIERS=@im=ibus
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
```

```
IM_CONFIG_PHASE=1
COMPIZ_CONFIG_PROFILE=ubuntu
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
XDG_SEAT=seat0
HOME=/home/julien
SHLVL=1
LANGUAGE=en_US
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=julien
QT4_IM_MODULE=xim
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/snapd/d
esktop
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-jH9kfagEpM
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/julien/.Xauthority
OLDPWD=/home/julien/c
_=/usr/bin/printenv
julien@ubuntu:~/c/shell$
```

Actually, every process comes with an environment. When a new process is created, it inherits a copy of its parent's environment. To access the entire environment within a process, you have several options:

- via the `main` function
- via the global variable `environ` (man environ)

main

So far we have seen that `main` could have different prototypes:

- `int main(void);`
- `int main(int ac, char **av);`

There is actually another prototype:

- `int main(int ac, char **av, char **env);`

where `env` is a `NULL` terminated array of strings.

```
julien@ubuntu:~/c/shell$ cat env-main.c
#include <stdio.h>

/**
 * main - prints the environment
 *
 * Return: Always 0.
 */
int main(int ac, char **av, char **env)
{
    unsigned int i;

    i = 0;
    while (env[i] != NULL)
    {
        printf("%s\n", env[i]);
        i++;
    }
    return (0);
}
julien@ubuntu:~/c/shell$ gcc -Wall -Werror -pedantic env-main.c -o printenv && ./printenv
XDG_VTNR=7
XDG_SESSION_ID=c2
CLUTTER_IM_MODULE=xim
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/julien
SESSION=ubuntu
GPG_AGENT_INFO=/home/julien/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
SHELL=/bin/bash
XDG_MENU_PREFIX=gnome-
VTE_VERSION=4205
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=23068682
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1558
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=julien
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=4
0;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=
01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tl
z=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;
31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;3
1:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;3
1:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;3
5:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=0
1;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pc
x=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:
*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;3
5:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;3
5:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;
36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=0
0;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
```

```
QT_ACCESSIBILITY=1
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
SESSION_MANAGER=local/ubuntu:@/tmp/.ICE-unix/1792,unix/ubuntu:/tmp/.ICE-unix/1792
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
PATH=/home/julien/bin:/home/julien/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bi
n:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
DESKTOP_SESSION=ubuntu
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
JOB=dbus
PWD=/home/julien/c/shell
XMODIFIERS=@im=ibus
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
IM_CONFIG_PHASE=1
COMPIZ_CONFIG_PROFILE=ubuntu
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
XDG_SEAT=seat0
HOME=/home/julien
SHLVL=1
LANGUAGE=en_US
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=julien
QT4_IM_MODULE=xim
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/snapd/d
esktop
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-jH9kfagEpM
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/julien/.Xauthority
OLDPWD=/home/julien/c
_=./printenv
julien@ubuntu:~/c/shell$
```

Exercises

0. printenv with environ

Write a program that prints the environment using the global variable `environ`.

1. env vs environ

Write a program that prints the address of `env` (the third parameter of the `main` function) and `environ` (the global variable). Are they they same? Does this make sense?

2. getenv()

Write a function that gets an environment variable. (without using `getenv`)

- Prototype: `char *_getenv(const char *name);`

man 3 `getenv`

3. PATH

Write a function that prints each directory contained in the the environment variable `PATH`, one directory per line.

4. PATH

Write a function that builds a linked list of the `PATH` directories.

5. setenv

Write a function that changes or adds an environment variable (without using `setenv`).

- Prototype: `int _setenv(const char *name, const char *value, int overwrite);`

man 3 `setenv`

6. unsetenv

Write a function that deletes the variable name from the environment (without using `unsetenv`).

- Prototype: `int _unsetenv(const char *name);`

man 3 `unsetenv`