Waiting for a Child Process Exit

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

pid_t waitpid(pid_t pid, int *status, int ops);
```

Waits until the process `pid` exits and sets `*status`

Use `WEXITSTATUS(status)` to get the value that `main` returned or passed to `exit`
#include "csapp.h"

int main() {
    pid_t pid = Fork();
    if (pid == 0) {
        printf("HC: hello from child\n");
        exit(17);
    } else {
        int child_status;
        printf("HP: hello from parent\n");
        Waitpid(pid, &child_status, 0);
        printf("CT: child result %d\n",
               WEXITSTATUS(child_status));
    }
    printf("Bye\n");
    return 0;
}
Child Process that Continues

Without using `waitpid`, a child process can continue after its parent exits:

```c
#include "csapp.h"

int main() {
    pid_t pid = Fork();
    if (pid == 0) {
        Sleep(10);
        printf("Child done\n");
    } else
        printf("Parent of %d done\n", pid);
    return 0;
}
```
 Process IDs Get Recycled

```c
#include "csapp.h"
#define SHOW_EACH_N 1000

int main() {
    int count = 0;
    while (1) {
        pid_t pid = Fork();
        if (pid == 0)
            return count / SHOW_EACH_N;
        else {
            int child_status;
            Waitpid(pid, &child_status, 0);
            if ((count % SHOW_EACH_N) == 0) {
                printf("child %d; pid %d; result %d\n",
                        count, pid, WEXITSTATUS(child_status));
            }
            count++;
        }
    }
}
```
Zombie Processes

Q: If a process ID can be recycled, how do you know that `waitpid` waits for the intended process?

A: `waitpid` must `reap` a process before its ID can be recycled

```c
pid_t init_pid = Fork();
if (init_pid == 0) {
    return 0;
}
....
if (pid == init_pid)
    printf("recycled!\n");
```

A process that has exited but not yet been reaped is a **zombie** or **defunct** process
The init Process

If a parent doesn’t wait for a child, the child process is adopted by init, which is process 1

```c
#include "csapp.h"

int main() {
    pid_t pid = Fork();
    if (pid == 0) {
        printf("Child of %d started\n", getppid());
        Sleep(2);
        printf("Child of %d done\n", getppid());
    } else {
        Sleep(1);
        printf("Parent of %d done\n", pid);
    }
    return 0;
}
```

Child likely (not guaranteed) to print 1 as parent when done
Waiting for Multiple Processes

Use \texttt{-1} in place of a process ID to wait on all children

```c
#include "csapp.h"
#define N 10

int main() {
    pid_t pid[N];
    int i, child_status;

    for (i = 0; i < N; i++) {
        if ((pid[i] = fork()) == 0) {
            /* Child */
            Sleep(i % 3);
            printf("Done %d\n", getpid());
            exit(i);
        }
    }

    for (i = 0; i < N; i++) {
        pid_t wpid = Waitpid(-1, &child_status, 0);
        if (WIFEXITED(child_status))
            printf("Saw %d done with %d\n", wpid, WEXITSTATUS(child_status));
        else
            printf("Child %d terminated abnormally\n", wpid);
    }
}
```
# Alternate Wait Function

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

pid_t wait(int *status);
```

Short for `waitpid(-1, status, 0)`
#include <unistd.h>

int execve(char *prog, char **argv, char **env);

Running Programs

Replaces current process state with executable \texttt{prog}
- Discards current code, stack, and heap
- Preserves process ID

Gives new program \texttt{argv}
\begin{itemize}
  \item \texttt{NULL}-terminated, and
  \item \texttt{argv[0]} matches \texttt{prog} by convention
\end{itemize}

Sets its environment variables to \texttt{env}
\begin{itemize}
  \item ... normally \texttt{environ}
\end{itemize}
Running Programs

```c
#include "csapp.h"

char *dum_argv[] = { "dum", NULL }; 

int main(int argc, char **argv) {
    printf("Dee, pid = %d\n", getpid());
    Execve("dum", dum_argv, environ);
    printf("Never happens!\n");
    return 0;
}
```

```c
#include "csapp.h"

int main(int argc, char **argv) {
    printf("Dum, pid = %d\n", getpid());
    return 0;
}
```
Example: Running a Program as a New Process

```c
#include "csapp.h"

char *ls_argv[] = { "/bin/ls", "-lt", "/usr/include", NULL }; 

int main(int argc, char **argv) { 
    int status; 

    printf("Listing /usr/include...
"); 

    if (Fork() == 0) 
        Execve(ls_argv[0], ls_argv, environ); 

    (void)Wait(&status); 
    printf("Done listing."); 

    return 0; }
```