Departamento de Sistemas Telemáticos y Computación (GSyC)

Gestión de Procesos

-exec, fork, exit, wait-

Katia Leal Algara
katia.leal@urjc.es
https://gsyc.urjc.es/~katia/

execl, execv, execlp, execvp: ejecutan un fichero

```
#include <unistd.h>
int execl(const char *path, const char *arg, ...)
int execv(const char *path, char *const argv[]);
int execlp(const char *file, const char *arg, ...);
int execvp(const char *file, char *const argv[]);
```

- Reemplazan la imagen del proceso en curso con una nueva.
- Primer argumento: camino del fichero que va a ser ejecutado.
- Opción `l` (lista de argumentos): const char *arg y puntos suspensivos siguientes se consideran como arg0, arg1, ..., argn. La lista de argumentos debe ser terminada por un puntero NULL.
- Opción `v` (vector de punteros a cadenas de caracteres): el primer argumento debe apuntar al nombre de fichero que se esté ejecutando (arg0). El vector debe terminar con un puntero NULL.
- Opción `p` (path Vs file): buscar un fichero ejecutable si el nombre de fichero especificado no contiene un carácter de barra inclinada (/).

execl, execv, execclp, execvp: ejecutan un fichero

- Casi todos los atributos del procesos se conservan.
- Por defecto, los descriptores de fichero se mantienen abiertos.
- En caso de éxito, `exec` no retorna. En caso de error, se retorna un -1 y la variable `errno` toma el valor del error.

Ejemplo:

Para ejecutar:

```
/bin/ls -l /usr/include
```

Usaríamos:

```
execl("/bin/ls",
      "ls",
      "-l",
      "/usr/include”,
      NULL);
```

fork: crear un proceso hijo

```
#include <sys/types.h
#include <unistd.h>
pid_t fork(void);
```

- Crea un proceso hijo que difiere de su proceso padre sólo en su PID y PPID.
- Usa páginas de copia-en-escritura (copy-on-write).
- En caso de éxito, se devuelve el PID del proceso hijo en el hilo de ejecución de su padre, y se devuelve un 0 en el hilo de ejecución del hijo.
- En caso de fallo, se devolverá un -1 en el contexto del padre, no se creará ningún proceso hijo, y se pondrá en errno un valor apropiado.
- El punto del programa donde el proceso hijo comienza su ejecución es justo en el retorno de la función fork, al igual que ocurre con el padre.

exit, wait: terminación normal del proceso

```
#include <stdlib.h>
void exit(int status);
```

- Terminación normal del proceso y devolución de `status` al proceso padre.
- Todos los flujos abiertos se vuelcan y cierran.
- EXIT_SUCCESS y EXIT_FAILURE.
- No devuelve nada ni regresa.

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- Suspende la ejecución del proceso actual hasta que un proceso hijo ha terminado.
- `waitpid` suspende la ejecución del proceso en curso hasta que un hijo especificado por el argumento `pid` ha terminado.
- El ID del proceso del hijo que terminó, o -1 en caso de error (errno se pone a un valor apropiado)

Ejemplo

```
 1 pid = fork();
 2 if (pid == -1){
 3         perror("");
 4         return -1;
 5 }
 6 else if (pid == 0){ /* Child */
 7         execvp(cmd->name, cmd->args);
 8         perror(cmd->name);
 9         exit(EXIT_FAILURE);
10 }
11 else /* Parent */
12         child_pid = wait(&status);
```