Processes

- Let us investigate
  - how processes are created in unix
  - how they execute
  - what parent/child relationship they have

Processes

```c
/* Example PIDs */
/* program: procid.c */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(void)
{
   printf("Process ID: %ld\n", (long)getpid());
   printf("Parent process ID: %ld\n", (long)getppid());
   printf("Owner user ID: %ld\n", (long)getuid());
}
```

Processes

- Let's "fork"
  - what happens when this prog. executes?

```
/* program: twoprocs.c  */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void main(void)
{
    pid_t childpid;

    printf("\nMy shell's PID is %ld\n", (long)getppid());
    if ((childpid = fork()) == 0) {
       fprintf(stderr, "\nI am the child, ID = %ld\n", (long)getpid());
       sleep(20);
       fprintf(stderr, "\nChild slept for 20 secs\n");
          /*   child code goes here  */
    } else if (childpid > 0) {
       fprintf(stderr, "\nI am the parent, ID = %ld\n", (long)getpid());
       sleep(30);
       fprintf(stderr, "\nParent slept for 30 secs\n");
          /*  parent code goes here */
    }

}
```

Processes

- Let's "really fork"
  - what happens when this prog. executes?

```
/* program: chain.c  */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void main(void)
{
    int i;
    int n;
    pid_t childpid;
    n = 6;
    for (i = 1; i < n;  ++i)
       if (childpid = fork())
          break;
    fprintf(stderr,"This is process %ld with parent %ld\n",
           (long)getpid(), (long)getppid());
    sleep(1);

}
```

Processes

- How about this program?

```
/* program: tree.c  */
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void main(void)
{
   int i;
   int n;
   pid_t childpid;

   n = 4;
   for (i = 1; i < n;  i++)
      if ((childpid = fork()) == -1)
         break;
   fprintf(stderr, "This is process %ld with parent %ld\n",
      (long)getpid(), (long)getppid());

    sleep(1);
}
```

Processes

- Using the exec family of system calls
  - execl, execlp, execle, execv, execvp, execvP
  - The exec family of functions replaces the current process image with a new process image.
    - execve is a system call within the kernel
    - the others are library functions that call execve

Processes

- Using the execl system call

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

void main(void)
{
    pid_t childpid;
    int status;

    if ((childpid = fork()) == -1)  {
       perror("Error in the fork");
       exit(1);
    } else  if (childpid == 0) {               /* child code */
       if (execl("/bin/ls", "ls", "-l", NULL) < 0) {
          perror("Exec of ls failed");
          exit(1);
       }
    } else if (childpid != wait(&status))   /* parent code */
       perror("A signal occurred before the child exited");
      else
          fprintf(stderr, "Child terminated normally. So will I!\n");
    exit(0);
}
```

Processes

Difference in the six exec functions is

1. whether the program file to execute is specified by a filename of a pathname
2. whether the argument to the new program are listed one by one or referenced through an array of pointers, and
3. whether the environment of the calling process is passed to the new program or whether a new environment is specified

```
#include <unistd.h>
int  execl(const char *path, const char *arg0, ... /*, (char *) 0 */);
int  execle(const char *path, const char *arg0, ... /*, (char *)0, char *const envp[] */);
int  execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int  execv(const char *path, char *const argv[]);
int  execvp(const char *file, char *const argv[]);
int execvP(const char *file, const char *search_path, char *const argv[]);
```

source: Stevens' *UNIX Networking Programming* book

Processes

from Stevens' *UNIX Networking Programming* book

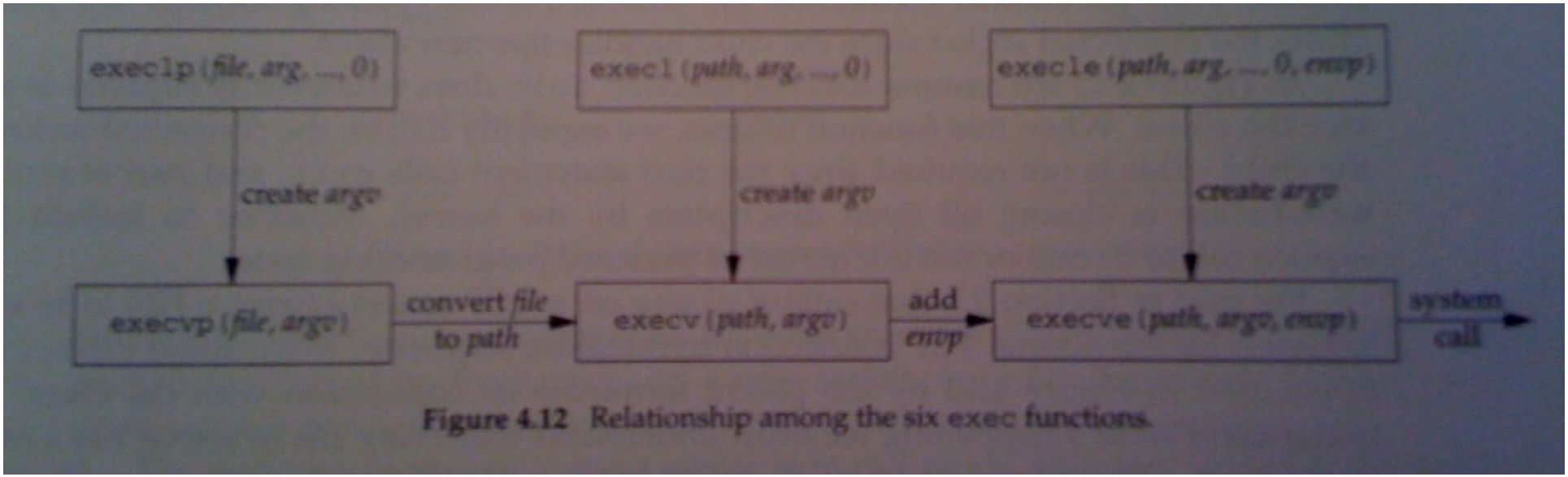

Figure 4.12 Relationship among the six exec functions.