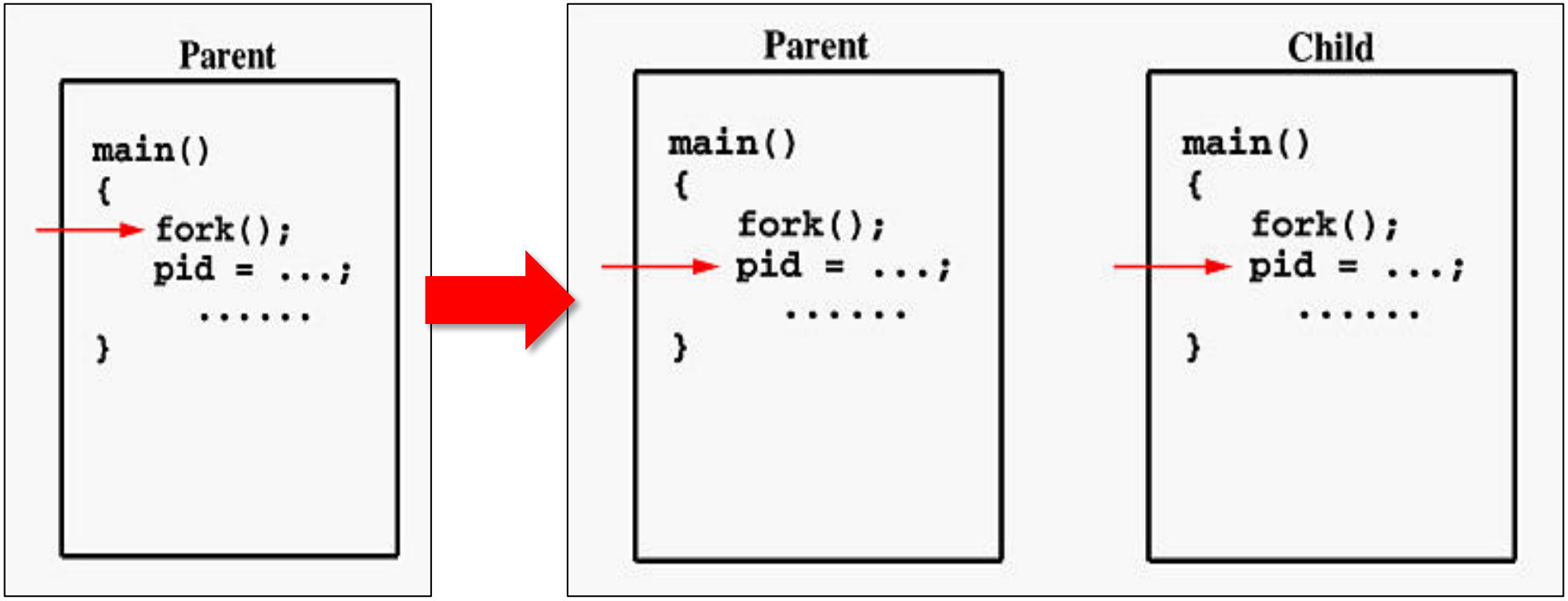
Process Creation

fork()

- A LINUX system call
- Enables a running process to create new process
- Newly created process is called *child process*
- Caller process is called *parent process*

fork()

After a new child process is created, both processes will execute the next instruction after the fork() system call.



fork()

- Takes no argument
- Returns
  - *Child creation unsuccessful*
    - -1 to parent
  - *Child creation successful*
    - Zero to child
    - A positive number (child's process id) to parent

Therefore, testing the return value of fork() can distinguish parent from the child.

Execution

2 possibilities

- ✔ The parent continues to execute concurrently with its children.
- ✔ The parent waits until some or all of its children have terminated.

Example

```
#include <sys/types.h>
#include <unistd.h>

void main(void)
{
    int pid;
    pid = fork();
    if (pid == -1) {
            printf("error in process creation\n")
            exit(1);
    }
    else if (pid == 0) child_code();
    else parent_code();
}
```

Address Space of Child

Unix will make an exact copy of the parent's address space and give it to the child.

Therefore, the parent and child processes have separate address spaces.

Example

prog7.c

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

int main()
{
    int var = 13;
    pid_t pid = fork();
    for(int i=0; i<3 && pid==0; i++)
    {
        pid = fork();
        var = var + 13;
        printf("%d: %d\n", pid, var);
    }
    printf("Good Bye\n");
    return 0;
}
```

Output

```
kowshika@kowshika-Inspiron-3442:~$ gcc prog7.c -o prog7
kowshika@kowshika-Inspiron-3442:~$ ./prog7
Good Bye
5657: 26
Good Bye
0: 26
5658: 39
Good Bye
0: 39
5659: 52
Good Bye
0: 52
Good Bye
kowshika@kowshika-Inspiron-3442:~$
```

Example

prog8.c

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

int main()
{
    int var = 13, x=0;
    pid_t pid = fork();
    if(!pid) fork();
    else var++;
    for(int i = 0; i<2 && x==0;i++)
    {
        if(pid==0) printf("%d\n", var++);
        else x = fork();
    }
    printf("%d Good Bye\n", var);
    return 0;
}
```

Output

```
kowshika@kowshika-Inspiron-3442:~$ gcc prog8.c -o prog8
kowshika@kowshika-Inspiron-3442:~$ ./prog8
14 Good Bye
13
14
15 Good Bye
13
14
15 Good Bye
14 Good Bye
14 Good Bye
```

Output

```
kowshika@kowshika-Inspiron-3442:~$ ./prog8
14 Good Bye
14 Good Bye
13
14
15 Good Bye
14 Good Bye
13
14
15 Good Bye
```

Same output as previous run, in different order

Address Space of Child

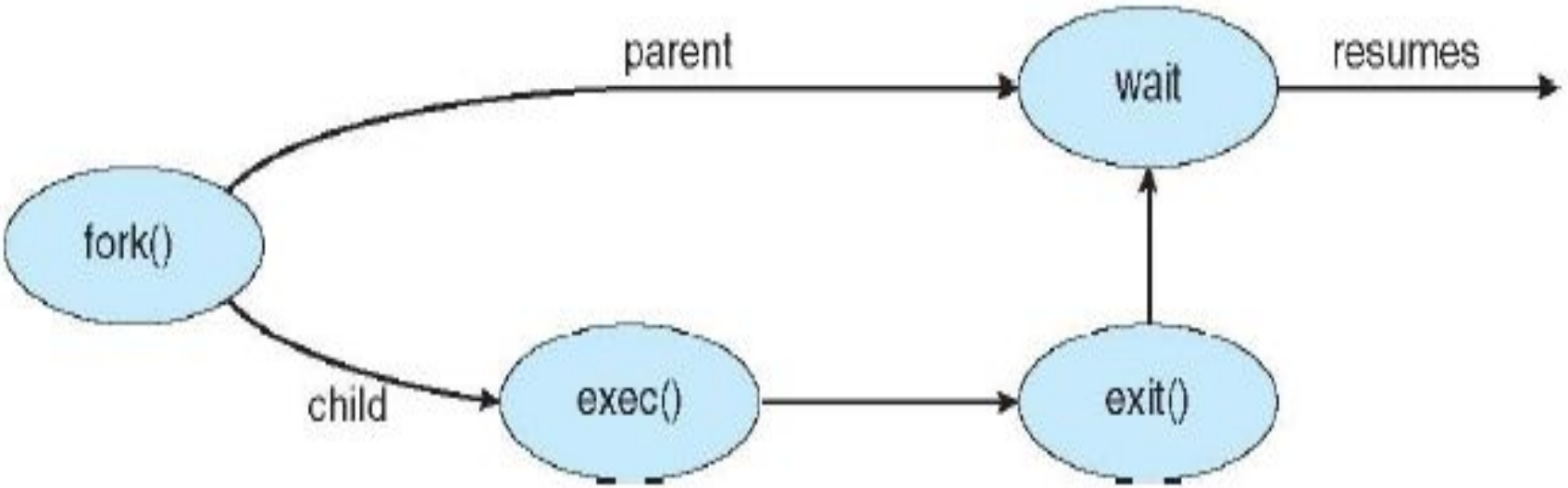
exec() can be used after a fork() to replace the child process' address space with a new program

No new process is not created, the process identifier (PID) does not change, but the machine code, data, heap, and stack of the process are replaced by those of the new program.

Example

```
#include <sys/types.h>
#include <studio.h>
#include <unistd.h>
int main()
{
pid_t  pid;
      /* fork another process */
      pid = fork();
      if (pid < 0) { /* error occurred */
            fprintf(stderr, "Fork Failed");
            return 1;
      }
      else if (pid == 0) { /* child process */
            execlp("/bin/ls", "ls", NULL);
      }
      else { /* parent process */
            /* parent will wait for the child */
            wait (NULL);
            printf ("Child Complete");
      }
      return 0;
}
```

Graphical Representation



Some System Calls

getpid() – returns process id

getppid() – returns parent process id

wait() – suspends execution of calling process until any one of it's children has terminated

waitpid(#pid) – suspends execution of calling process until a particular child (the one with process id *#pid*) has terminated