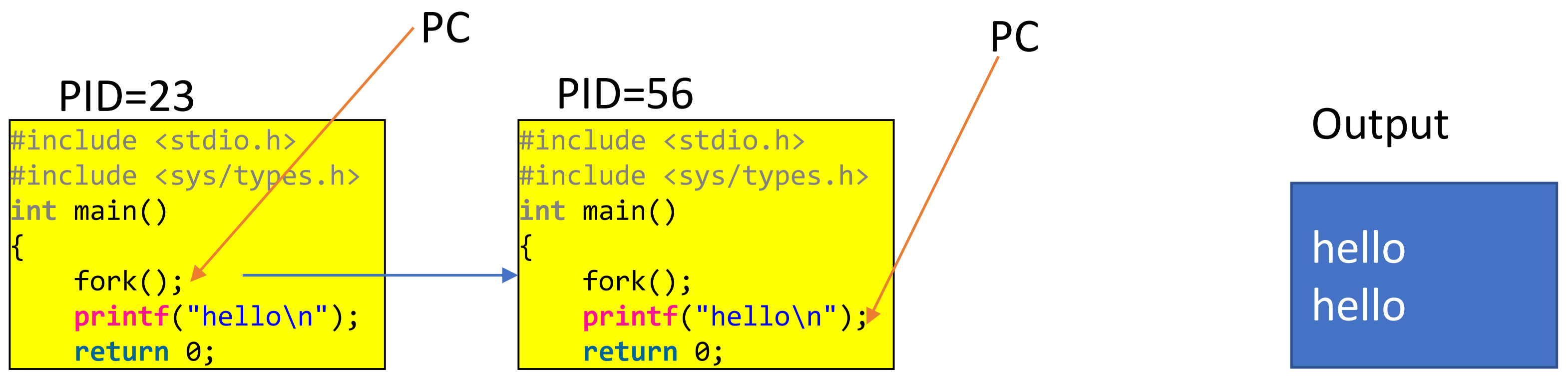
Chapter 3 – Process Fork()

OS Book – page 117

Fork()

- A new process is created by the fork() system call.
- The new process consists of a copy of the address space of the original process.
    - This mechanism allows the parent process to communicate easily with its child process.
- Both processes (the parent and the child) continue execution at the instruction after the fork(),
    - with one difference: the return code for the fork() is zero for the new (child) process, whereas the (nonzero) process identifier of the child is returned to the parent.
- Because the child is a copy of the parent, each process has its own copy of any data.
- After a fork() system call, one of the two processes typically uses the exec() system call to replace the process’s memory space with a new program.
- The exec() system call loads a binary file into memory (destroying the memory image of the program containing the exec() system call) and starts its execution. In this manner, the two processes are able to communicate and then go their separate ways.
- The parent can then create more children; or, if it has nothing else to do while the child runs, it can issue a wait() system call to move itself off the ready queue until the termination of the child.



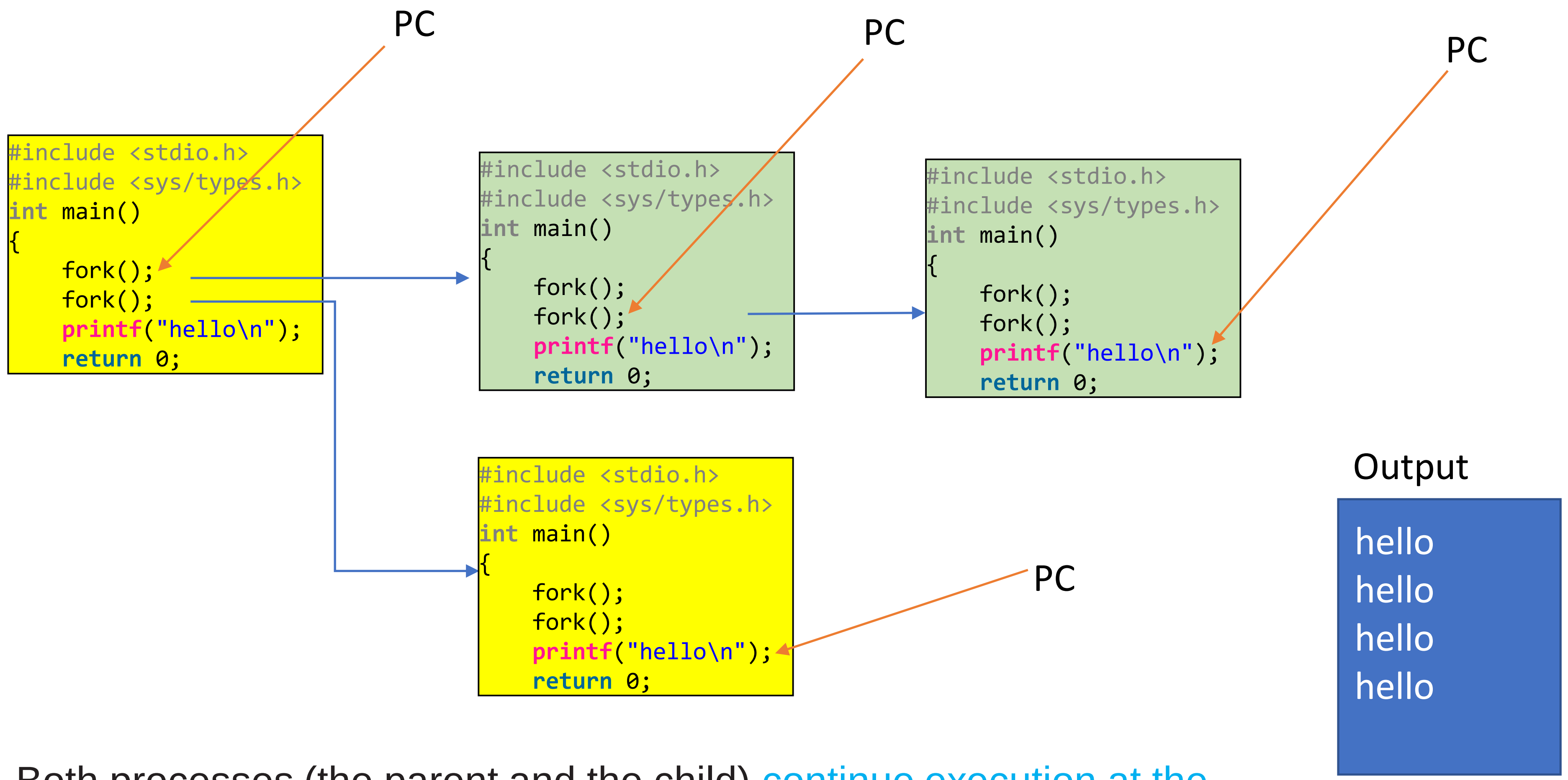
```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    printf("hello\n");
    return 0;
```

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    printf("hello\n");
    return 0;
```

```
hello
hello
```

Both processes (the parent and the child) continue execution at the instruction after the fork(),

PC = Program Counter



```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    printf("hello\n");
    return 0;
```

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    printf("hello\n");
    return 0;
```

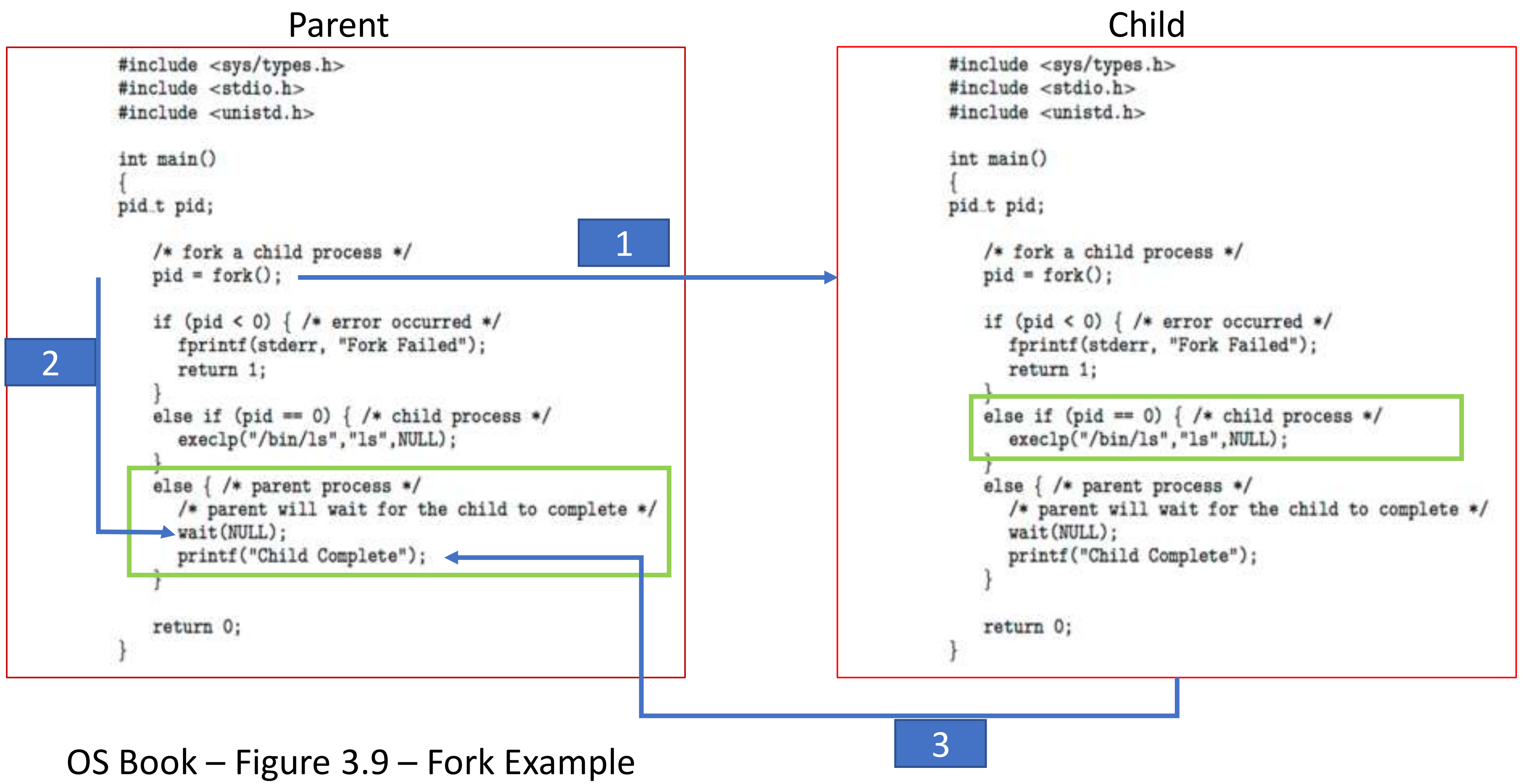
```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    printf("hello\n");
    return 0;
```

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    printf("hello\n");
    return 0;
```

```
hello
hello
hello
hello
```

Both processes (the parent and the child) continue execution at the instruction after the fork(),

PC = Program Counter



Parent

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

int main()
{
pid_t pid;

   /* fork a child 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 to complete */
     wait(NULL);
     printf("Child Complete");
   }

   return 0;
}
```

Child

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

int main()
{
pid_t pid;

   /* fork a child 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 to complete */
     wait(NULL);
     printf("Child Complete");
   }

   return 0;
}
```

OS Book – Figure 3.9 – Fork Example

Output/Console
List (ls exec result)
-file1
-file2
-file3
-file4
.....
.....
Child Complete

Example

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int value = 6;
int main()
{
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
value += 10;
return 0;
}
else if (pid > 0) { /* parent process */
wait(NULL);
printf ("PARENT: value = %d\n",value);
/* LINE A */
return 0;
}
}
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int value = 6;
int main()
{
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
value += 10;
return 0;
}
else if (pid > 0) { /* parent process */
wait(NULL);
printf ("PARENT: value = %d\n",value);
/* LINE A */
return 0;
}
}
```

Output

```
PARENT: value = 6
```

* Output is 5 as the child and parent processes each have their own copy of value.