CS 3305A

Process

Lecture 3

Sept 18th 2023

Fork() Example

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

int main()
{
   pid_t pid;
   int i;
   pid = fork();
   if( pid > 0 ) /* parent */
   {

      for( i=0; i < 10; i++ )
                printf("\t\t\tPARENT %d\n", i);


    }
```

```
else /* child */

{
        for( i=0; i < 10; i++ )
      printf( "CHILD %d\n", i );
}
return 0;
}
```

What is the possible output?

Fork() Example: Possible output

```
PARENT 0
   PARENT 1
   PARENT 2
   PARENT 3
   PARENT 4
   PARENT 5
   PARENT 6
   PARENT 7
   PARENT 8
   PARENT 9
CHILD 0
CHILD 1
CHILD 2
CHILD 3
CHILD 4
CHILD 5
CHILD 6
CHILD 7
CHILD 8
CHILD 9
```

Fork() Example: Possible output

```
                    PARENT 0
                   PARENT 1
                   PARENT 2
                   PARENT 3
                   PARENT 4
                   PARENT 5
                   PARENT 6
CHILD 0
CHILD 1
CHILD 2            PARENT 7
                   PARENT 8
                   PARENT 9

CHILD 3
CHILD 4
CHILD 5
CHILD 6
CHILD 7
CHILD 8
CHILD 9
```

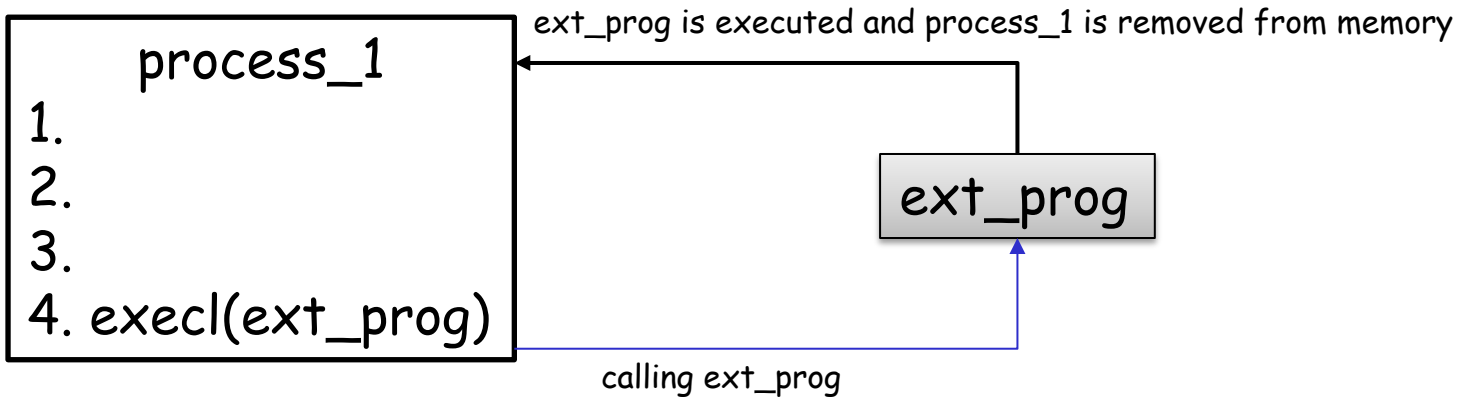
Lots of possible outputs!!

Execution

- Processes get a share of the CPU to give another process a turn
- The switching between the parent and child depends on many factors:
  - machine load, process scheduling
- Output is nondeterministic
  - Cannot determine output by looking at code

execl()

- The system call execl() replace a process (the caller process) with a new loaded program
- execl() loads a binary file into memory (destroying the memory image of the program calling it)
- On success, execl() never returns; on failure, execl() returns -1



execl() Example

Program A:

```
int i = 5;
printf("%d\n",i);
execl( "B", "", NULL);
printf("%d\n",i);
```

Program B (assume binary file):

```
main()
{
    printf("hello\n");
}
```

- What is the output of program A?
  - 5
  - hello
- Why is it not this?
- 5
- hello
- 5
- The execl call replaces program A with program B.

execl(arg_0, arg_1, arg_2, …, arg_N-1, arg_N): 1st argument i.e., arg_0 must be the path/file name, last arg i.e., arg_N must be NULL, and arg 1 to arg N-1 in between. However, in addition to arg_0 and arg_N, at least one argument must be there [for arg_1 to arg_N-1, in absence of any actual arg, at least a dummy argument i.e., " " must be included]

fork() and execl()

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
   pid_t pid;
   pid = fork();
 if (pid > 0)
  {
     wait(NULL);
     printf("Child Complete");
  }
else{
    if (pid == 0) {
      execl("B", "", NULL);
     printf("\n You'll never see this
     line.."); }
  }
}
```

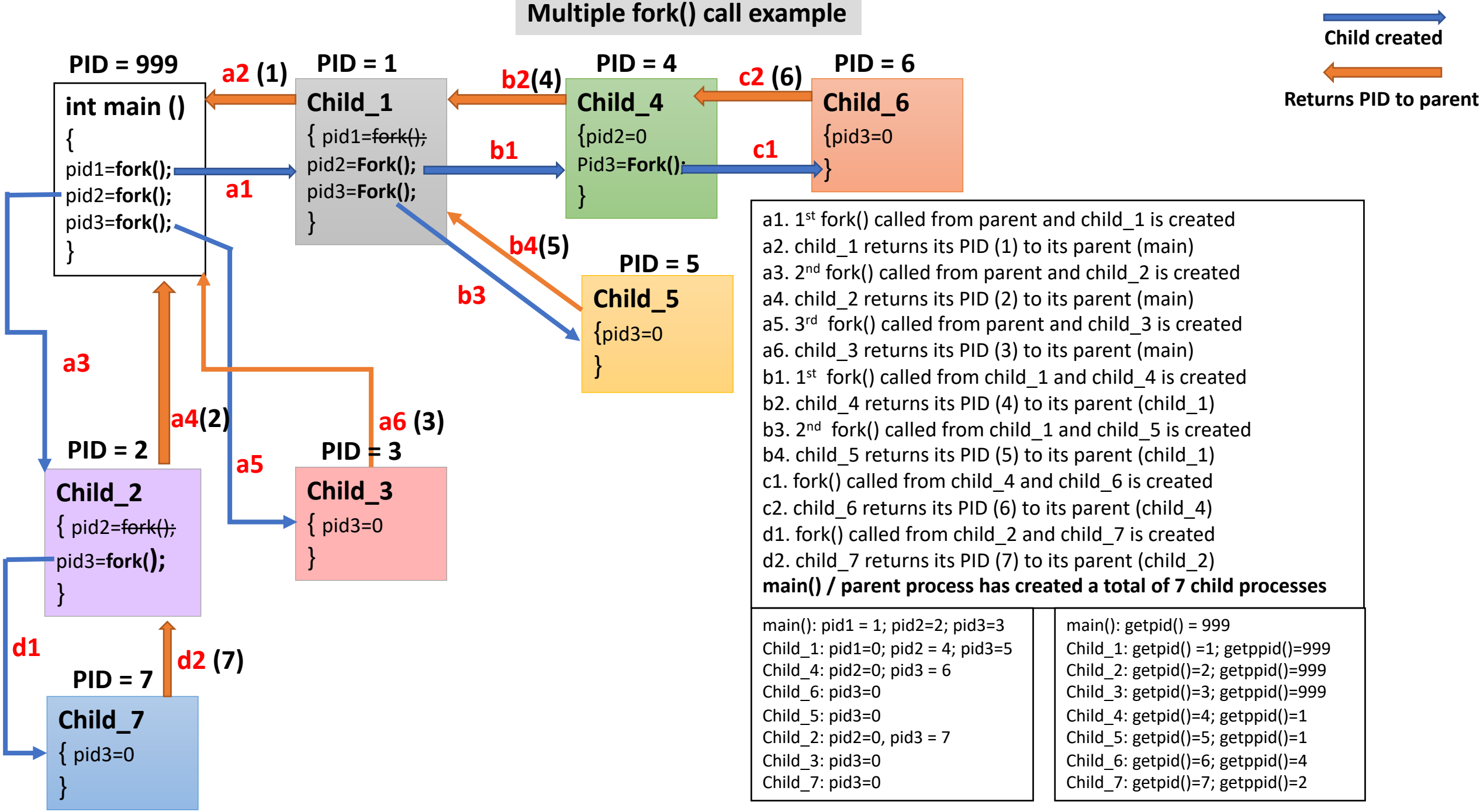
fork() and execl()

- execl() overlays a new program on the existing process
- Child will not return to the old program unless exec fails. This is an important point to remember.

How many Processes are created by this Program?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    fork();
    fork();
    fork();
}
```

Multiple fork() call example



a1. 1st fork() called from parent and child_1 is created
a2. child_1 returns its PID (1) to its parent (main)
a3. 2nd fork() called from parent and child_2 is created
a4. child_2 returns its PID (2) to its parent (main)
a5. 3rd fork() called from parent and child_3 is created
a6. child_3 returns its PID (3) to its parent (main)
b1. 1st fork() called from child_1 and child_4 is created
b2. child_4 returns its PID (4) to its parent (child_1)
b3. 2nd fork() called from child_1 and child_5 is created
b4. child_5 returns its PID (5) to its parent (child_1)
c1. fork() called from child_4 and child_6 is created
c2. child_6 returns its PID (6) to its parent (child_4)
d1. fork() called from child_2 and child_7 is created
d2. child_7 returns its PID (7) to its parent (child_2)
main() / parent process has created a total of 7 child processes

main(): pid1 = 1; pid2=2; pid3=3
Child_1: pid1=0; pid2 = 4; pid3=5
Child_4: pid2=0; pid3 = 6
Child_6: pid3=0
Child_5: pid3=0
Child_2: pid2=0, pid3 = 7
Child_3: pid3=0
Child_7: pid3=0

main(): getpid() = 999
Child_1: getpid() =1; getppid()=999
Child_2: getpid()=2; getppid()=999
Child_3: getpid()=3; getppid()=999
Child_4: getpid()=4; getppid()=1
Child_5: getpid()=5; getppid()=1
Child_6: getpid()=6; getppid()=4
Child_7: getpid()=7; getppid()=2