# Exercises-Chapter 3

# Introduction to Processes

## 1. Using the program shown in Figure below, explain what the output will be at Line A.

## int value = 5;

## int main(){

## pid\_t pid;

## pid = fork();

## if (pid ==0){

## value += 15;

## return 0;

## }

## else if (pid>0) {

## wait(NULL);

## printf(“PARENT, Value = %d”, value); //Line A

## return 0;

}

## }

The output will be PARENT: value = 5

**Explanation**:

Fork system call creates new process which is called as Child process and the process which called fork() function is called as Parent process. After the child process created, both Child and Parent processes will execute the instruction followed by fork() system call. The new process shares the parent-process program counter, CPU registers, files which are opened in Parent process.

From the above code, the child process will have the variable "value" as 5, before it is being created. Later it adds 15 to it and then exits by returning 0. But parent process is unaware of what is happening inside child process. So, it simply waits until the child process finishes its execution because of wait(NULL) statement. The parent process has not modified the value of variables "value", so its value will be "5".

## 2. Including the initial parent process, how many processes are created by the program shown in Figure below?

## 

## #include <stdio.h>

## #include<unistd.h>

## 

## int main(){

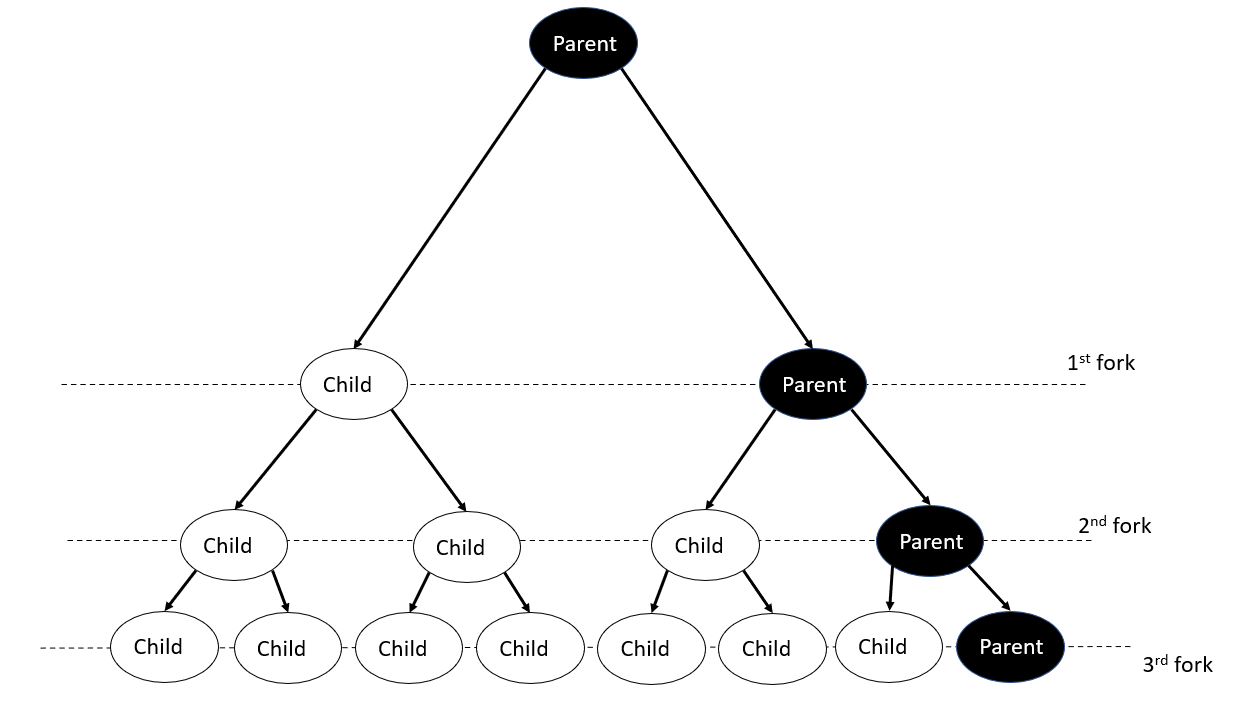
## fork();

## fork();

## fork();

## return;

## }



8 processes are created; 1 parent and 7 children

## 3. When a process creates a new process using the fork() operation, which of the following contents is shared between the parent process and the child process?

#### Stack

#### Heap

#### Shared memory segments created by shmem

Shared Memory Segments

A process creates a new process using the fork() operation. Shared Memory Segment is the only one shared between parent process and the child process.

4. Define the term context switch. What activities does the OS undertake to context switch from one process to another.

A context switch is when the CPU switches to another process.

The System must save the state of the old process and load the saved state for the new process via a context switch.

## 5. What information needs to kept in a Process Control Block (PCB) and which ones needs to be saved, changed or updated when context switching takes place.

The Process Control Block (PCB) typically contains:

* Process state – running, waiting, etc
* Program counter – location of instruction to next execute
* CPU registers – contents of all process-centric registers
* CPU scheduling information- priorities, scheduling queue pointers
* Memory-management information – memory allocated to the process
* Accounting information – CPU used, clock time elapsed since start, time limits
* I/O status information – I/O devices allocated to process, list of open files

The current process state must be stored before a context switch in the PCB.

This includes the values in the CPU registers (particulary the Program Counter) and OS-specific information (such as priority).

All information about the job is saved before the context switch. Anything already completed is also saved.

## 7. Draw the state transition diagram of a process and describe the activities that trigger each transition. Include the states new, ready, running, waiting and terminated.

## Why is there no transition between ready and waiting or between waiting and running?



A process cannot go from Ready to Waiting, it can only go from Waiting to Ready.

A process can only go from Running to Waiting, not Waiting to Running without being Ready first.

## 8. Describe the various process queues in an OS. What event or program cause a PCB move from one queue to the other?

3 Scheduling Queues:

* Job Queue – set of all process in the system
* Ready Queue – set of all process residing in main memory, ready and waiting to execute
* Device Queue – set of process waiting for an I/O device

\*\*\*Processes migrate among the various queues

## 9. Describe the differences among short-term, medium-term, and long term scheduling.

|  |  |  |
| --- | --- | --- |
| **Long Term** | **Short Term** | **Medium Term** |
| It is a job scheduler. | It is a CPU scheduler. | It is swapping. |
| Speed is less than short term scheduler. | Speed is very fast. | Speed is in between both. |
| It controls the degree of multiprogramming | Less control over the degree of multiprogramming. | Reduce the degree of multiprogramming. |
| Absent or minimal in a time-sharing system. | Minimal in a time-sharing system. | Time-sharing system uses a medium-term scheduler. |
| It selects processes from the pool and load them into memory for execution. | It selects from among the processes that are ready to execute. | Process can be reintroduced into the meat and its execution can be continued. |
| Process state is (New to Ready). | Process state is (Ready to Running). | -- |
| Select a good process, mis of I/O bound, and CPU bound. | Select a new process for a CPU quite frequently. | -- |

## 10. To obtain process information for the UNIX or Linux system in tree format, use the command ps –aef --forest. Name a process that has 2 forked children. How many children does the init process have and describe what each one does?

Bash has 2 forked children.

Init has 3 children: login, kthreadd, and sshd.

## 11. Using the program given below, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

## #include *<*sys/types.h*>*

## #include *<*stdio.h*>*

## #include *<*unistd.h*>*

## int main()

## {

## pid t pid, pid1;

## /\* fork a child process \*/

## pid = fork();

## 

## if (pid < 0) { /\* error occurred \*/

## fprintf(stderr, "Fork Failed");

## return 1;

## }

## else if (pid == 0) { /\* child process \*/

## pid1 = getpid();

## printf("child: pid = %d",pid); /\* A \*/

## printf("child: pid1 = %d",pid1); /\* B \*/

## }

## else { /\* parent process \*/

## pid1 = getpid();

## printf("parent: pid = %d",pid); /\* C \*/

## printf("parent: pid1 = %d",pid1); /\* D \*/

## wait(NULL);

## }

## return 0;

## }

If the actual pid of parent and child are 2600 and 2603 respectively then by executing the program the values can be obtained as,

A = 0  
B = 2603  
C = 2603  
D = 2600

The process is allowed to run with pid 2603 for child and 2600 for parent , At the line A the pid of child is zero and at the line B the non zero pid of child process is for parent.