

Processes



Practice Exercises

- 3.1 Using the program shown in Figure 3.30, explain what the output will be at LINE A.

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

int value = 5;

int main()
{
    pid_t pid;

    pid = fork();

    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d",value); /* LINE A */
        return 0;
    }
}
```

Figure 3.30 What output will be at Line A?

Answer:

The result is still 5, as the child updates its copy of value. When control returns to the parent, its value remains at 5.

- 3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

Answer:

Eight processes are created.

- 3.3 Original versions of Apple’s mobile iOS operating system provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

Answer:

- a. The CPU scheduler must be aware of the different concurrent processes and must choose an appropriate algorithm that schedules the concurrent processes.
 - b. Concurrent processes may need to communicate with one another, and the operating system must therefore develop one or more methods for providing interprocess communication.
 - c. Because mobile devices often have limited memory, a process that manages memory poorly will have an overall negative impact on other concurrent processes. The operating system must therefore manage memory to support multiple concurrent processes.
- 3.4 Some computer systems provide multiple register sets. Describe what happens when a context switch occurs if the new context is already loaded into one of the register sets. What happens if the new context is in memory rather than in a register set and all the register sets are in use?

Answer:

The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and be moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

- 3.5 When a process creates a new process using the `fork()` operation, which of the following states is shared between the parent process and the child process?

- a. Stack
- b. Heap
- c. Shared memory segments

Answer:

Only the shared memory segments are shared between the parent process and the newly forked child process. Copies of the stack and the heap are made for the newly created process.

- 3.6 Consider the “exactly once” semantic with respect to the RPC mechanism. Does the algorithm for implementing this semantic execute correctly

even if the ACK message sent back to the client is lost due to a network problem? Describe the sequence of messages, and discuss whether “exactly once” is still preserved.

Answer:

The “exactly once” semantics ensure that a remote procedure will be executed exactly once and only once. The general algorithm for ensuring this combines an acknowledgment (ACK) scheme combined with timestamps (or some other incremental counter that allows the server to distinguish between duplicate messages).

The general strategy is for the client to send the RPC to the server along with a timestamp. The client will also start a timeout clock. The client will then wait for one of two occurrences: (1) it will receive an ACK from the server indicating that the remote procedure was performed, or (2) it will time out. If the client times out, it assumes the server was unable to perform the remote procedure, so the client invokes the RPC a second time, sending a later timestamp. The client may not receive the ACK for one of two reasons: (1) the original RPC was never received by the server, or (2) the RPC was correctly received—and performed—by the server but the ACK was lost. In situation (1), the use of ACKs allows the server ultimately to receive and perform the RPC. In situation (2), the server will receive a duplicate RPC, and it will use the timestamp to identify it as a duplicate so as not to perform the RPC a second time. It is important to note that the server must send a second ACK back to the client to inform the client the RPC has been performed.

- 3.7 Assume that a distributed system is susceptible to server failure. What mechanisms would be required to guarantee the “exactly once” semantic for execution of RPCs?

Answer:

The server should keep track in stable storage (such as a disk log) of information regarding what RPC operations were received, whether they were successfully performed, and the results associated with the operations. When a server crash takes place and an RPC message is received, the server can check whether the RPC has been previously performed and therefore guarantee “exactly once” semantics for the execution of RPCs.

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```

Figure 3.31 How many processes are created?