3.8) Describe the actions taken by a kernel to context-switch between processes.

Answer: In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

3.11) Including the initial parent process, how many processes are created by the program shown in Figure 3.32?

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

 int main() {
      int i;

      for (i = 0; i < 4; i++) fork();

      return 0;
   }
```

Answer: Sixteen processes are created. The program online includes printf() state-ments to better explain how many processes have been created.

3.12) Explain the circumstances under which the line of code marked printf("LINE J") in Figure 3.33 will be reached.

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

 int main()
 { pid_t pid;

/* fork a child process */ ???

     pid = fork
      if (pid lt; 0) { /* error occurred */
            fprintf(stderr, "Fork Failed");
            return 1;
            }
      else if (pid == 0) { /* child process */
            execlp("/bin/ls","ls",NULL);
```

```
                printf("LINE J");
                }
        else { /* parent process */
        /* parent will wait for the child to complete */
                wait(NULL);
                printf("Child Complete");
        }

       return 0;
 }
```

Answer: The call to exec() replaces the address space of the process with the program specified as the parameter to exec(). If the call to exec() succeeds, the new program will begin running, and control from the call to exec() will never return. In this scenario, the line printf("Line J"); will never be performed. However, if an error occurs in the call to exec(), the function returns control, and therefore the line printf("Line J"); will be performed

3.13) Using the program in Figure 3.34, 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.)

```
/* fork a child process */
pid = fork();

if (pid lt; 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;
```

Answer: A= 0, B = 2603, C = 2603, D = 2600