Chapter 3-4

Chapter 3 :

3.2 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.

3.4 Using the program shown in following , explain what will be output at Line A.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int value=8;
int main()
{
pid_t pid;

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

if (pid == 0) { /* child process */
                value +=15;
        }
        else { /* parent process */
                /* parent will wait for the child to complete */

                wait(NULL);
                printf(" Parent :value= %d\n",value);/*LINE A*/
                exit(0);
        }
}
```

Answer: Parent :value=8

Chapter 4 :

4.4 Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values

b. Heap memory

c. Global variables

d. Stack memory

Answer: The threads of a multithreaded process share **heap memory** and **global variables**. Each thread has its separate set of register values and a separate stack.

4.7 The program shown in Figure 4.11 uses the Pthreads API. What would be output from the program at `LINE C` and `LINE P`?

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

int value=0;
void    *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
int pid;
pthread_t tid;
pthread_attr_t attr;

        pid = fork();

        if (pid == 0)    {/* child process */
            pthread_attr_init(&attr);
            pthread_create(&tid, &attr, runner, NULL);
            pthread_join(tid, NULL);
            printf(“CHILD: value = %d”, value); /* LINE C*/
        }
        else if (pid > 0) {/* parent process */
                    wait(NULL);
                    printf(“PARENT: value = %d”, value); /* LINE P */
        }
}

void *runner(void *param) {
          value=10;
          pthread_exit(0);
}
```

Answer: Output at `LINE C` is 10. Output at `LINE P` is 0.