1. A thread is usually defined as a “light weight process” because an operating system (OS) maintains smaller data structures for a thread than for a process. In relation to this, which of the following is TRUE? Justify your answer and brief the same with suitable sketch. Also, justify each option.

(A) On per-thread basis, the OS maintains only CPU register state

(B) The OS does not maintain a separate stack for each thread

(C) On per-thread basis, the OS does not maintain virtual memory state

(D) On per-thread basis, the OS maintains only scheduling and accounting information

Solution :(C)

Os,on per thread basis,maintains ONLY 2 things : CPU Register state and stack space.It does not maintain anything else for individual thread.Code segment and Global variables are shared.Even TLB and Page Tables are also shared since they belong to same process.

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, (and a thread ID.) As you can see, for a single thread of control –there is one program counter, and one sequence of instructions that can be carried out at any given time and for multi-threaded applications-there are multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code, data, and certain structures such as open files.

2. 2. Consider the following code fragment: if (fork() == 0) { a = a + 5; printf("%d,%d\n", a, &a); } else { a = a –5; printf("%d, %d\n", a, &a); } Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is TRUE? Justify your answer.

(A) u = x + 10 and v = y

(B) u = x + 10 and v != y

(C) u + 10 = x and v = y

(D) u + 10 = x and v != y

Text

Description automatically generated

3. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle? Justify your answer.

(A) 0%

(B) 10.6%

(C) 30.0%

(D) 89.4%

ANS: 10.6%

Let three processes be a0, a1 and a2. Their execution time is 10, 20 and 30 respectively. a0 spends first 2 time units in I/O, 7 units of CPU time and finally 1 unit in I/O. a1 spends first 4 units in I/O, 14 units of CPU time and finally 2 units in I/O. a2 spends first 6 units in I/O, 21 units of CPU time and finally 3 units in I/O.

idle p0 p1 p2 idle  
0 2 9 23 44 47

Total time spent = 47  
Idle time = 2 + 3 = 5  
Percentage of idle time = (5/47)\*100 = 10.6 %

4. Group 1 contains some CPU scheduling algorithms and Group 2 contains some applications. Match entries in Group 1 to entries in Group 2. Also, justify the same.

Group I Group II

(P) Gang Scheduling (1) Guaranteed Scheduling

(Q) Rate Monotonic Scheduling (2) Real-time Scheduling

(R) Fair Share Scheduling (3) Thread Scheduling

(a) P – 3 Q – 2 R – 1 (b) P – 1 Q – 2 R – 3 (c) P – 2 Q – 3 R – 1 (d) P – 1 Q – 3 R – 2

Answer (A)

Gang scheduling for parallel systems that schedules related threads or processes to run simultaneously on different processors.

Rate monotonic scheduling is used in real-time operating systems with a static-priority scheduling class. The static priorities are assigned on the basis of the cycle duration of the job: the shorter the cycle duration is, the higher is the job’s priority.

Fair Share Scheduling is a scheduling strategy in which the CPU usage is equally distributed among system users or groups, as opposed to equal distribution among processes. It is also known as Guaranteed scheduling.

5. A certain computation generates two arrays a and b such that a[i]=f(i) for 0 ≤ i < n and b[i]=g(a[i]) for 0 ≤ i < n. Suppose this computation is decomposed into two concurrent processes X and Y such that X computes the array a and Y computes the array b. The processes employ two binary semaphores R and S, both initialized to zero. The array a is shared by the two processes. The structures of the processes are shown below. Process X: Process Y: private i; private i; for (i=0; i < n; i++) { for (i=0; i < n; i++) { a[i] = f(i); EntryY(R, S); ExitX(R, S); b[i]=g(a[i]); } } Which one of the following represents the CORRECT implementations of ExitX and EntryY? Justify your answer. (A) ExitX(R, S) { P(R); V(S);} EntryY (R, S) { P(S); V(R);} (B) ExitX(R, S) { V(R); V(S);} EntryY(R, S) { P(R); P(S);} (C) ExitX(R, S) { P(S); V(R);} EntryY(R, S) { V(S); P(R);} (D) ExitX(R, S) { V(R); P(S);} EntryY(R, S) { V(S); P(R);}

Answer: (C)

Explanation:

The purpose here is neither the deadlock should occur

nor the binary semaphores be assigned value greater

than one.

A leads to deadlock

B can increase value of semaphores b/w 1 to n

D may increase the value of semaphore R and S to

2 in some cases