1. A critical section is a program segment   
(a) which should run in a certain specified amount of time    
(b) which avoids deadlocks    
(c) where shared resources are accessed    
(d) which must be enclosed by a pair of semaphore operations, P and V   
  
Ans: option (c)

2. A solution to the Dining Philosophers Problem which avoids deadlock is

(a) ensure that all philosophers pick up the left fork before the right fork

(b) ensure that all philosophers pick up the right fork before the left fork

(c) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork

(d) None of the above

Ans: option (c)

3. Consider the methods used by processes P1 and P2 for accessing their critical sections whenever needed, as given below. The initial values of shared boolean variables S1 and S2 are randomly assigned.

|  |  |
| --- | --- |
| *Method Used by P1* | *Method Used by P2* |
| while (S1 == S2) ;  Critica1 Section  S1 = S2; | while (S1 != S2) ;  Critica1 Section  S2 = not (S1); |

Which one of the following statements describes the properties achieved?

(a) Mutual exclusion but not progress

(b) Progress but not mutual exclusion

(c) Neither mutual exclusion nor progress

(d) Both mutual exclusion and progress

Ans: option (a)

Explanation:

Principle of Mutual Exclusion: No two processes may be simultaneously present in the critical section at the same time. That is, if one process is present in the critical section other should not be allowed.

P1 can enter critical section only if S1 is not equal to S2, and P2 can enter critical section only if S1 is equal to S2. Therefore Mutual Exclusion is satisfied.

Progress: no process running outside the critical section should block the other interested process from entering critical section whenever critical section is free.

Suppose P1 after executing critical section again want to execute the critical section and P2 dont want to enter the critical section, then in that case P1 has to unnecesarily wait for P2. Hence progress is not satisfied.

4. A counting semaphore was initialized to 10. Then 6 P (wait) operations and 4V (signal) operations were completed on this semaphore. The resulting value of the semaphore is

(a) 0 (b) 8 (c) 10 (d) 12

Ans: option(b)

Explanation:

6P => decrements the semaphore 6 times. Hence , the value becomes 4.

4V => increments the semaphore 4 times. Hence , the value becomes 8.

Note: The positive value of counting semaphore indicates that those many down (P) operations can be carried out successfully. The negative value of counting semaphore indicates that the number of blocked processes.

5. Let m[0]…m[4] be mutexes (binary semaphores) and P[0] …. P[4] be processes.

Suppose each process P[i] executes the following:

  wait (m[i]);wait (m[(i+1) mode 4]);   
  .........  
  release (m[i]); release (m[(i+1)mod 4]);

This could cause

(a) Thrashing                                (b) Deadlock

(c) Starvation, but not deadlock     (d) None of the above

Ans: option (b)

Explanation:

Deadlock occurs, if each each process gets preempted after executing wait(m[i]);

6. The enter\_CS() and leave\_CS() functions to implement critical section of a process are realized using test-and-set instruction as follows:  
 void enter\_CS(X)  
 {  
     while test-and-set(X) ;  
 }  
 void leave\_CS(X)  
 {  
     X = 0;  
 }  
In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements:  
I. The above solution to CS problem is deadlock-free  
II. The solution is starvation free.  
III. The processes enter CS in FIFO order.  
IV More than one process can enter CS at the same time.   
Which of the above statements is TRUE?   
(a) I only  
(b) I and II  
(c) II and III  
(d) IV only  
  
Ans: option (a)  
Explanation:  
The test-and-set instruction is an instruction used to write to a memory location and return its old value as a single atomic (i.e., non-interruptible) operation. Since it is an atomic instruction it guarantees mutual exclusion.

7. The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0=1, S1=0, S2=0.

|  |  |  |
| --- | --- | --- |
| **Process P0** | **Process P1** | **Process P2** |
| while (true) {  wait (S0);  print (0);  release (S1);  release (S2);  } | wait (S1);  Release (S0); | wait (S2);  release (S0); |

How many times will process P0 print '0'?  
(a) At least twice                              (b) Exactly twice   
(c) Exactly thrice                              (d) Exactly once  
  
Ans: option (a)  
Explanation:

P0 will execute first because only S0=1. Hence it will print 0 (for the first time). Also P0 releases S1 and S2. Since S1=1 and S2=1, therefore P1 or P2, any one of them can be executed.

Let us assume that P1 executes and releases S0 (Now value of S0 = 1). Note that P1 process is completed.

Now S0=1 and S2=1, hence either P0 can execute or P2 can execute. Let us check both the conditions:-

1. Let us assume that P2 executes, and releases S0 and completes its execution. Now P0 executes; S0=0 and prints 0 (i.e. second 0). And then releases S1 and S2. But note that P1 and P2 processes has already finished their execution. Again if P0 tries to execute it goes into sleep condition because S0=0. Therefore, minimum number of times '0' gets printed is 2.

2. Now, let us assume that P0 executes. Hence S0=0, (due to wait(S0)), and it will print 0 (second 0) and releases S1 and S2. Now only P2 can execute, because P1 has already completed its execution and P0 cannot execute because S0 = 0. Now P2 executes and releases S0 (i.e. S0=1) and finishes its execution. Now P0 starts its execution and again prints 0 (thrid 0) and releases S1 and S2 (Note that now S0=0). P1 and P2 has already completed its execution therefore again P1 takes its turn, but since S0=0, it goes into sleep condition. And the processes P1 and P2 which could wakeup P0 has already finished their execution.Therefore, maximum number of times '0' gets printed is 2.

***GATE-2012***

8. Fetch\_And\_Add(X,i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

  AcquireLock(L){

         while (Fetch\_And\_Add(L,1))

               L = 1;

   }

  ReleaseLock(L){

         L = 0;

   }

This implementation

(a) fails as L can overflow

(b) fails as L can take on a non-zero value when the lock is actually available

(c) works correctly but may starve some processes

(d) works correctly without starvation

Ans: option (b)

Explanation:

Assume that L=0, that means the lock is now available. A process P1 wants to acquire the lock by executing the AcquireLock() function. We can see that the while loop fails because the Fetch\_And\_Add instruction returns the previous value of L, which was 0. (Note that after the execution of the atomic instruction, the present value of L is now 1). Since the returned value was 0, the while loop fails and P1 process comes out of the while loop and hence acquires the lock.

Now scheduler schedules another process P2 and context switching takes place. P2 tries to acquire the lock by executing the AcquireLock() function. But it goes on executing the while loop infinitely (because the atomic instruction always returns a non-zero value and the while loop condition is always true).

After some time, scheduler again schedules P1. We assume that P2 was stopped soon after the Fetch\_And\_Add() instruction returned the value. Hence L has some non-zero value.  Now P1 releases the lock L. That means now L=0. Assume that again context switch takes place and P2 arrives.

We have assumed that P2 was switched out when it executed the Fetch\_And\_Add instruction and a non-zero value has been returned. Since the returned value was non-zero  the condition becomes true and the statement L=1 is executed. Hence now L=1. Again the while loop is executed, Fetch\_And\_Add instruction will return value 1 and hence again the while loop enters into infinite loop. Now none of the process is able to acquire the lock. Therefore the above implementation fails.

9. Two processes, P1 and P2, need to access a critical section of code. Consider the following synchronization construct used by the processes:

|  |  |
| --- | --- |
| /\* P1 \*/  while (true) {    wants1 = true;    while (wants2 == true);    /\* Critical      Section \*/    wants1=false;  }  /\* Remainder section \*/ | /\* P2 \*/  while (true) {    wants2 = true;    while (wants1==true);    /\* Critical      Section \*/    wants2 = false;  }  /\* Remainder section \*/ |

Here, wants1 and wants2 are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct?

(a) It does not ensure mutual exclusion.

(b) It does not ensure bounded waiting.

(c) It requires that processes enter the critical section in strict alternation.

(d) It does not prevent deadlocks, but ensures mutual exclusion.

Ans: option (d)

Explanation:

The code ensures the condition of mutual exclusion: Assume P1 is initiated. It sets wants1=true.  Now since wants2 = false, P1 exists from its while loop and enters its critical section. Now suppose context switch takes place and P2 gets executed. Now it sets wants2=true, and now enters the while loop and remains busy till P1 comes out of the critical section and sets wants1=false, because wants1=true( as set by P1). So we can see that the mutual exclusion condition is satisfied.

The code does not prevent deadlock: Assume that P1 starts its execution. It sets wants1=true and then gets preempted. Now P2 starts its execution. P2 sets wants2=true and suddenly gets preempted. Now P1 starts execution; it enters the while loop and finds that wants2=true and remains busy in the while loop. Now P1 gets preempted. P2 enters into execution; it enters the while loop and finds that wants1=true remains busy in the while loop. Hence both P1 and P2 remains busy forever.

10. The atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x. consider the following implementation of P and V functions on a binary semaphore .

void P (binary\_semaphore \*s) {

  unsigned y;

  unsigned \*x = &(s->value);

  do {

     fetch-and-set x, y;

  } while (y);

}

void V (binary\_semaphore \*s) {

  S->value = 0;

}

Which one of the following is true?

(a) The implementation may not work if context switching is disabled in P.

(b) Instead of using fetch-and-set, a pair of normal load/store can be used

(c) The implementation of V is wrong

(d) The code does not implement a binary semaphore

Ans: option (a)

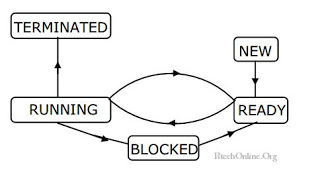
Explanation:

Assume that P1 process enters the P function. Initially if the value of semaphore is 1, according to the definition of the fetch-and-set instruction, the value of y will be 1  when the fetch-and-set instruction is executed. Hence the while loop goes on executing until other processes execute the V function. But if context switching is disabled in P while loop will keep on executing and other processes are not allowed to execute.

Explanation:  
It is mentioned that there are 3 processes. Let it be P1, P2 & P3.  Every process arrives at the barrier, that is it completes its first section of the code. Hence process\_arrived = 3. Now assume that P1 continues its execution and leaves the barrier and makes process\_left=1. Now assume that P1 again invokes the barrier function immediately. At this stage, process\_arrived =4. P1 now waits. Soon P2 leaves the barrier, which makes process\_left=2. Now P3 leaves the barrier which makes process\_left=3. "If" condition is true and now process\_arrived and process\_left is reset to 0.   
We know that P1 is waiting. At this stage assume that P2 invokes the barrier function, hence process\_arrived=1 and it waits. P3 invokes the barrier function, hence process\_arrived=2. All process have reached the barrier but since the process\_arrived=2, all processes keeps on waiting. Hence deadlock.

12. Which one of the following rectifies the problem in the implementation?  
(a) Lines 6 to 10 are simply replaced by process\_arrived--  
(b) At the beginning of the barrier the first process to enter the barrier waits until process\_arrived becomes zero before proceeding to execute P(S).  
(c) Context switch is disabled at the beginning of the barrier and re-enabled at  the end.  
(d) The variable process\_left is made private instead of shared.  
  
Ans: option (b)

. The process state transition diagram in Fig.1.8 is representative of

[](http://3.bp.blogspot.com/-ThylgCC8ImI/UPzPOPhh04I/AAAAAAAAAOg/dUgMV_FV2uU/s1600/gate1996os.JPG)

(a) a batch operating system      
(b) an operating system with a preemptive scheduler   
(c) an operating system with a non-preemptive scheduler      
(d) a uni-programmed operating system.   
  
Ans: option (b)  
Explanation:  
If there is a transition from Running State to ready State then the state diagram is the representative of an operating system with a preemptive scheduler.

***GATE-1996***

2. Which of the following is an example of spooled device?

(a) A line printer used to print the output of a number of jobs.

(b) A terminal used to enter input data to a running program.

(c) A secondary storage device in a virtual memory sytem.

(d) A graphic display device.

Ans: option (a)

Explanation:

Spool stands for simultaneous peripheral operations on-line.

The most common spooling application is print spooling: Printers typically can print only a single document at a time and require seconds or minutes to do so. With spooling, multiple processes can write documents to a print queue without waiting. As soon as a process has written its document to the spool device, the process can perform other tasks, while a separate printing process operates the printer.

3. Which of the following is an example of a spooled device?  
(a) The terminal used to enter the input data for the C program being executed  
(b) An output device used to print the output of a number of jobs.  
(c) The secondary memory device in a virtual storage system  
(d) The swapping area on a disk used by the swapper.  
  
Ans: option (b)

4. Which of the following is true?  
(a) Unless enabled, a CPU will not be able to process interrupts.       
(b) Loop instructions cannot be interrupted till they complete.  
(c) A processor checks for interrupts before executing a new instruction.   
(d) Only level triggered interrupts are possible on microprocessors  
  
Ans: option (a)  
Explanation:  
Interrupts are unexpected events in a sequence of execution of instructions causing an interruption of the normal program flow.  
Option (b) is false.  
Option (c) depends upon the architecture of the processor.  
Option (d) is false because we have level-triggered interrupts and edge-triggered interrupts.

5. System calls are usually invoked by using  
(a) a software interrupt     (b) polling   
(c) an indirect jump         (d) a privileged instruction  
  
Ans: option (d)  
Explanation:  
Software interrupt is generated as the result of the execution of the privileged instruction.

6. A multi-user, multi-processing operating system cannot be implemented on  
hardware that does not support  
(a) Address translation     
(b) DMA for disk transfer  
(c) At least two modes of CPU execution (privileged and non-privileged)   
(d) Demand paging  
  
Ans: option (d)  
Explanation:  
Requirements of a multi-user, multi-processing operating system are:-  
1) Address translation     
2) DMA for disk transfer  
3) At least two modes of CPU execution (user mode and kernel mode)

7. Which of the following actions is/are typically not performed by the operating   
system when switching context from process A to process B?   
(a) Saving current register values and restoring saved register values for process B.      
(b) Changing address translation tables.   
(c) Swapping out the memory image of process A to the disk.      
(d) Invalidating the translation look-aside buffer.   
  
Ans: option (c)  
Explanation:  
Swapping out the memory image of process to the disk occurs only when the process is suspended.

8. A processor needs software interrupt to

(a) test the interrupt system of the processor

(b) implement co-routines

(c) obtain system services which need execution of privileged instructions

(d) return from subroutine

Ans: option (c)

9. A CPU has two modes-privileged and non-privileged. In order to change the mode   
from privileged to non-privileged    
(a) a hardware interrupt is needed   
(b) a software interrupt is needed   
(c) a privileged instruction (which does not generate an interrupt) is needed   
(d) a non-privileged instruction (which does not generate an interrupt) is needed  
  
Ans: option (c)

10. Which of the following does not interrupt a running process?

(a) A device                   (b) Timer

(c) Scheduler process      (d) Power failure

Ans: option (c)

Explanation:

Scheduler process is to determine which process next should run in the CPU. It does not interrupt any running process.

11. Which of the following need not necessarily be saved on a context switch between processes?  
(a) General purpose registers  
(b) Translation look-aside buffer  
(c) Program counter  
(d) All of the above  
  
Ans: option (b)  
Explanation:

A Translation lookaside buffer (TLB) is a CPU cache that memory management hardware uses to improve virtual address translation speed. On a context switch, some TLB entries can become invalid, since the virtual-to-physical mapping is different. The simplest strategy to deal with this is to completely flush the TLB.

12. Which combination of the following features will suffice to characterize an OS as a multi-programmed OS?

(A) More than one program may  be loaded into main memory at the same time for execution.

(B) If a program waits for certain events such as I/O, another program is immediately scheduled for execution.

(C) If the execution of a program terminates, another program is immediately scheduled for execution.

(a) A                    (b) A and B                (c) A and C              (d) A, B and C

Ans: option (a)

Explanation:

Ability of the operating system to manage and hold multiple programs in the memory is known as multi-programmed OS.

13. A process executes thecode  
   fork ();  
   fork ();  
   fork ();  
The total number of child processes created is  
(a) 3 (b) 4 (c) 7 (d) 8  
  
Ans: option (c)  
Explanation:  
For n fork statements,  2n – 1 child processes are created.

***GATE-2008***

14. A process executes the following code

   for (i = 0; i < n; i++) fork();

The total number of child processes created is

(a) n

(b) 2n - 1

(c) 2n

(d) 2n+1 - 1;

Ans: option (b)

Explanation:

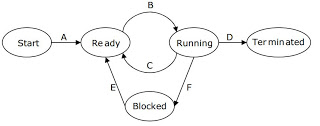
Loop executes for n times. Therefore, fork() is executed for n times.

For n fork statements,  2n – 1 child processes are created.

***GATE-2005***  
15. 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?  
(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  
  
Ans: option (c)  
Explanation:  
Child process will execute the if part and parent process will execute the else part. Assume that the initial value of a = 6. Then the value of a printed by the child process will be 11, and the value of a printed by the parent process in 1. Therefore u+10=x.  
  
Now the second part. The answer is v = y. (After compiling and verifying the above program we got the result. Explanation is given below.)  
  
We know that, the fork operation creates a separate address space for the child. But the child process has an exact copy of all the memory segments of the parent process. Hence the virtual addresses and the mapping (initially) will be the same for both parent process as well as child process. Note that, *the virtual address is same but virtual addresses exist in different processes' virtual address spaces*. And when we print &a, its actually printing the virtual address. Hence the answer is ***v = y***.   
  
The virtual address of parent & child may or may not be pointing to different physical address as explained below.  
  
When a fork() system call is issued, a copy of all the pages corresponding to the parent process is created, loaded into a separate memory location by the OS for the child process. But this is not needed in certain cases. When the child is needed just to execute a command for the parent process, there is no need for copying the parent process' pages, since exec replaces the address space of the process which invoked it with the command to be executed.   
  
In such cases, a technique called copy-on-write (COW) is used. With this technique, when a fork occurs, the parent process's pages are not copied for the child process. Instead, the pages are shared between the child and the parent process. Whenever a process (parent or child) modifies a page[To know what a page is [CLICK HERE](http://en.wikipedia.org/wiki/Fork_%28operating_system%29#Process_address_space)], a separate copy of that particular page alone is made for that process (parent or child) which performed the modification. This process will then use the newly copied page rather than the shared one in all future references.  [Ref: [WIKIPEDIA](http://en.wikipedia.org/wiki/Fork_%28operating_system%29)]

16. In the following process state transition diagram for a uniprocessor system,

assume that there are always some processes in the ready state:

[](http://2.bp.blogspot.com/-6Gr3uODIZ40/UPz9ugS94yI/AAAAAAAAAO0/mDKG4eeaXRI/s1600/gate2009os.JPG)

Now consider the following statements:

I. If a process makes a transition D, it would result in another process making transition A immediately.

II. A process P2 in blocked state can make transition E while another process P1 is in running state.

III. The OS uses preemptive scheduling.

IV. The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?

(a) I and II           (b) I and III           (c) II and III          (d) II and IV

Ans: option (c)

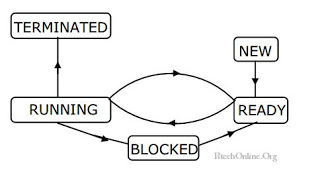
Explanation:

If a process makes a transition D, it would result in another process making transition from ready state to running state .i.e. transition B.

A process moves to ready state when I/O or other events are completed.

If there is a transition from Running State to ready State then the state diagram is the representative of an operating system with a preemptive scheduler.

*1. The process state transition diagram in Fig.1.8 is representative of*

[](http://3.bp.blogspot.com/-ThylgCC8ImI/UPzPOPhh04I/AAAAAAAAAOg/dUgMV_FV2uU/s1600/gate1996os.JPG)

(a) a batch operating system      
(b) an operating system with a preemptive scheduler   
(c) an operating system with a non-preemptive scheduler      
(d) a uni-programmed operating system.   
  
Ans: option (b)  
Explanation:  
If there is a transition from Running State to ready State then the state diagram is the representative of an operating system with a preemptive scheduler.

2. Which of the following is an example of spooled device?

(a) A line printer used to print the output of a number of jobs.

(b) A terminal used to enter input data to a running program.

(c) A secondary storage device in a virtual memory sytem.

(d) A graphic display device.

Ans: option (a)

Explanation:

Spool stands for simultaneous peripheral operations on-line.

The most common spooling application is print spooling: Printers typically can print only a single document at a time and require seconds or minutes to do so. With spooling, multiple processes can write documents to a print queue without waiting. As soon as a process has written its document to the spool device, the process can perform other tasks, while a separate printing process operates the printer.

Reference: <http://en.wikipedia.org/wiki/Spooling>

3. Which of the following is an example of a spooled device?  
(a) The terminal used to enter the input data for the C program being executed  
(b) An output device used to print the output of a number of jobs.  
(c) The secondary memory device in a virtual storage system  
(d) The swapping area on a disk used by the swapper.  
  
Ans: option (b)

4. Which of the following is true?  
(a) Unless enabled, a CPU will not be able to process interrupts.       
(b) Loop instructions cannot be interrupted till they complete.  
(c) A processor checks for interrupts before executing a new instruction.   
(d) Only level triggered interrupts are possible on microprocessors  
  
Ans: option (a)  
Explanation:  
Interrupts are unexpected events in a sequence of execution of instructions causing an interruption of the normal program flow.  
Option (b) is false.  
Option (c) depends upon the architecture of the processor.  
Option (d) is false because we have level-triggered interrupts and edge-triggered interrupts.

5. System calls are usually invoked by using  
(a) a software interrupt     (b) polling   
(c) an indirect jump         (d) a privileged instruction  
  
Ans: option (d)  
Explanation:  
Software interrupt is generated as the result of the execution of the privileged instruction.

6. A multi-user, multi-processing operating system cannot be implemented on  
hardware that does not support  
(a) Address translation     
(b) DMA for disk transfer  
(c) At least two modes of CPU execution (privileged and non-privileged)   
(d) Demand paging  
  
Ans: option (d)  
Explanation:  
Requirements of a multi-user, multi-processing operating system are:-  
1) Address translation     
2) DMA for disk transfer  
3) At least two modes of CPU execution (user mode and kernel mode)

7. Which of the following actions is/are typically not performed by the operating   
system when switching context from process A to process B?   
(a) Saving current register values and restoring saved register values for process B.      
(b) Changing address translation tables.   
(c) Swapping out the memory image of process A to the disk.      
(d) Invalidating the translation look-aside buffer.   
  
Ans: option (c)  
Explanation:  
Swapping out the memory image of process to the disk occurs only when the process is suspended.

8. A processor needs software interrupt to

(a) test the interrupt system of the processor

(b) implement co-routines

(c) obtain system services which need execution of privileged instructions

(d) return from subroutine

Ans: option (c)

9. A CPU has two modes-privileged and non-privileged. In order to change the mode   
from privileged to non-privileged    
(a) a hardware interrupt is needed   
(b) a software interrupt is needed   
(c) a privileged instruction (which does not generate an interrupt) is needed   
(d) a non-privileged instruction (which does not generate an interrupt) is needed  
  
Ans: option (c)

10. Which of the following does not interrupt a running process?

(a) A device                   (b) Timer

(c) Scheduler process      (d) Power failure

Ans: option (c)

Explanation:

Scheduler process is to determine which process next should run in the CPU. It does not interrupt any running process.

11. Which of the following need not necessarily be saved on a context switch between processes?  
(a) General purpose registers  
(b) Translation look-aside buffer  
(c) Program counter  
(d) All of the above  
  
Ans: option (b)  
Explanation:

A Translation lookaside buffer (TLB) is a CPU cache that memory management hardware uses to improve virtual address translation speed. On a context switch, some TLB entries can become invalid, since the virtual-to-physical mapping is different. The simplest strategy to deal with this is to completely flush the TLB.

*[CLICK TO KNOW MORE](http://en.wikipedia.org/wiki/Translation_lookaside_buffer" \t "_blank)*

12. Which combination of the following features will suffice to characterize an OS as a multi-programmed OS?

(A) More than one program may  be loaded into main memory at the same time for execution.

(B) If a program waits for certain events such as I/O, another program is immediately scheduled for execution.

(C) If the execution of a program terminates, another program is immediately scheduled for execution.

(a) A                    (b) A and B                (c) A and C              (d) A, B and C

Ans: option (a)

Explanation:

Ability of the operating system to manage and hold multiple programs in the memory is known as multi-programmed OS.

13. A process executes the code  
   fork ();  
   fork ();  
   fork ();  
The total number of child processes created is  
(a) 3 (b) 4 (c) 7 (d) 8  
  
Ans: option (c)  
Explanation:  
For n fork statements,  2n – 1 child processes are created.

14. A process executes the following code

   for (i = 0; i < n; i++) fork();

The total number of child processes created is

(a) n

(b) 2n - 1

(c) 2n

(d) 2n+1 - 1;

Ans: option (b)

Explanation:

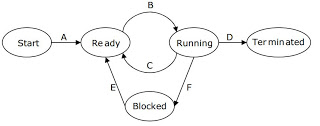
Loop executes for n times. Therefore, fork() is executed for n times.

For n fork statements,  2n – 1 child processes are created.

15. 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?  
(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  
  
Ans: option (c)  
Explanation:  
Child process will execute the if part and parent process will execute the else part. Assume that the initial value of a = 6. Then the value of a printed by the child process will be 11, and the value of a printed by the parent process in 1. Therefore u+10=x.  
  
Now the second part. The answer is v = y. (After compiling and verifying the above program we got the result. Explanation is given below.)  
  
We know that, the fork operation creates a separate address space for the child. But the child process has an exact copy of all the memory segments of the parent process. Hence the virtual addresses and the mapping (initially) will be the same for both parent process as well as child process. Note that, *the virtual address is same but virtual addresses exist in different processes' virtual address spaces*. And when we print &a, its actually printing the virtual address. Hence the answer is ***v = y***.   
  
The virtual address of parent & child may or may not be pointing to different physical address as explained below.  
  
When a fork() system call is issued, a copy of all the pages corresponding to the parent process is created, loaded into a separate memory location by the OS for the child process. But this is not needed in certain cases. When the child is needed just to execute a command for the parent process, there is no need for copying the parent process' pages, since exec replaces the address space of the process which invoked it with the command to be executed.   
  
In such cases, a technique called copy-on-write (COW) is used. With this technique, when a fork occurs, the parent process's pages are not copied for the child process. Instead, the pages are shared between the child and the parent process. Whenever a process (parent or child) modifies a page[To know what a page is [CLICK HERE](http://en.wikipedia.org/wiki/Fork_%28operating_system%29" \l "Process_address_space" \t "_blank)], a separate copy of that particular page alone is made for that process (parent or child) which performed the modification. This process will then use the newly copied page rather than the shared one in all future references.  [Ref: [WIKIPEDIA](http://en.wikipedia.org/wiki/Fork_%28operating_system%29" \t "_blank)]

16. In the following process state transition diagram for a uniprocessor system,

assume that there are always some processes in the ready state:

[](http://2.bp.blogspot.com/-6Gr3uODIZ40/UPz9ugS94yI/AAAAAAAAAO0/mDKG4eeaXRI/s1600/gate2009os.JPG)

Now consider the following statements:

I. If a process makes a transition D, it would result in another process making transition A immediately.

II. A process P2 in blocked state can make transition E while another process P1 is in running state.

III. The OS uses preemptive scheduling.

IV. The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?

(a) I and II           (b) I and III           (c) II and III          (d) II and IV

Ans: option (c)

Explanation:

If a process makes a transition D, it would result in another process making transition from ready state to running state .i.e. transition B.

A process moves to ready state when I/O or other events are completed.

If there is a transition from Running State to ready State then the state diagram is the representative of an operating system with a preemptive scheduler.