Abhishek Tiwari
2013CSB1001

ASSIGNMENT 04

QUESTION 1.
A total of 7 child processes will be created.
First process forks 3 child.
1st child will fork 2 child and 2nd child will fork 1 child of its own making a total of 7 processes.
So total number of processes is 7+1 (parent process) = 8 processes.

QUESTION 2.
(a.) A multithreaded program tries to seperate a program in many parts and solve them seperately.
If anyone tries to make a program which executes sequentially a multithreaded program,it might take longer time to execute because of the
extra time taken in the context switching.
Any dynamic program which depends on the close previous solution.Instaces of this could be coin change problem, calculating Fibonacci sequence etc.
Greedy algorthims and divide and conquer algorithms are efficient when multi-threading is used.Because one subproblem is independent of other sub problems on the same level.

(b.) If we remove a job from a spooling system, the conditions “No preemption” (i.e. processes holding a resource can release it) may be violated.

QUESTION 3.
If the number of kernel threads is equal to the number of processors, each processor will run a thread.And if number of kernel threads is less than number of processors, one or more processors will remain idle, because nothing will be executing on these processors.
This results in LACK OF RESOURCE UTILIZATION.
But if the number of kernel threads is less than number of processors , it will run slower than the case when number of kernel threads is equal to the number of processors given same number of user threads usually (not when context switching overhead is very high!)

QUESTION 4.
The OS can find if the process is experiencing starvation or not by checking the value of program counter in the PCB of the processes in the wait queue continuously after certain intervals.If the new PC value is same as the old one even after 5-6 polls, it means the the process is not executing, hence starving.
The OS will have to maintain a table of PC for each process currently waiting to execute.Which will definitely be a overhead if the number of processes are too high.
And we can not do much about the starvation of the process for certain set of instrutions already hard coded in the system. So it will be infeasible to keep track of processes starving.
It can be helpful sometimes too, suppose the process which is currently starving has some other resources allocated to it ,OS might kill this process and maybe start it again or something else to free the reosources held by a starving process which might be needed by other processes.

QUESTION 5.
When the processes are in safe state, then there will never any case of a deadlock but if the processes are in usafe state, it doesn't necessarily mean that there will be a deadlock.
In safe state, the processes are always set in the sequence such that the resources they MIGHT

NEED EVER are either free or used by a process before them, but it may happen that the processes only use only a set of resources not all of them (which is usually the case)
For example suppose there is a different resource for matrix multiplication (assume no compiler optimization) and a process was executing this piece of code ->

```
i=2*j+1;
if (i%2==0)    multiply(Matix1,Matrix2);
```

So it's clear that at runtime, process will never need the Matrix multiplication resource.
But for the state to be safe, OS will allocate matrix multiplication resource to it.(which can be ommited , making the State unsafe and avoiding Deadlock!)

QUESTION 6.

No,since there is preemption happening by the scheduler, there will never be a case of monopolization of CPU resources. The job currently executing will always be pre-empted from the CPU for the next process.
However, we can always implement a certain technique such that one particular process always gets the CPU causing other processes to starve.
For example ,suppose Scheduler always chooses the next task to be executed which has smallest temp_pid . And suppose the temp_pid of the current job being executed doesn't change (or remains minimum of all currently waiting to execute) then this process will monopolize the system (which is very rare though !)