HOME ASSIGNMENT 2

-Shashwat Sanghavi (121049)

Answer 1)

TRANSITION	EVENT
READY TO SUSPEND	No more memory is available, so the READY process is temporarily swapped out of memory
READY TO BLOCKED	Not possible
READY TO RUN	The process is allocated the CPU by the scheduler
RUN TO BLOCKED	I/O request of the process is fulfilled
RUN TO READY	Time allocated to the particular process expires
BLOCKED TO READY	The awaited event completes (e.g. I/O completion)
RUN TO SUSPEND	Not possible
SWAPPED TO READY OR	When memory becomes available or the reason for swapping is no
SWAPPED TO BLOCKED	longer true
BLOCKED TO RUN	Not possible

Answer 2)

T = 22: P5, P8 - ready/running
P1, P3, P7 - blocked for I/O
T = 37: P1, P3, P8 - ready/running
P5 - blocked suspend (swapped out)
P7 - blocked for I/O
T = 47: P1, P3, P5 - ready/running
P7 - blocked for I/O
P8 – exit

Answer 3)
It will return child process ID if process gets forked otherwise it will return 0.

Answer 4)

Mode switch between threads are done at user privilege. But in order to perform mode switch between processes one need to acquire kernel mode privilege. Switching between user mode and kernel mode is more expensive.

Answer 5)

- One can design scheduling for particular application. This application level thread scheduling will increase the efficiency.
- OS independent:
    - Kernel is not concerned about any user level thread which can make ULTs kernel/OS independent.
- One does not need kernel mode privilege to for thread management which improves it will save time which is required for switching between modes.

Answer 6)

- All of the user level threads run on same kernel level thread. Now if one thread requests for a resource which is not available then that process/kernel level thread will be blocked which will lead to blocking of all user level thread over that kernel level thread.
- In case of multithreading, ULTs cannot take advantage of multiprocessing as all the threads run on same KLT.

Answer 7)

All of the user level threads run on same kernel level thread. Now if one thread requests for a resource which is not available then that process/kernel level thread will be blocked which will lead to blocking of all user level thread over that kernel level thread.

Answer 8)

The mentioned program waits for the i/o request to be satisfied for most of the time. In a multithreaded program, one KLT will be blocked on the request of I/O by one ULT, but other threads will continue to run as they run on different KLTs.

On a uniprocessor machine, a process that would otherwise have to block because of blocking of that thread can continue to run its other threads.

Answer 9)

The threads will not continue to run as with termination of any process, all the associated threads will be terminated.

Answer 10)

Competing processes are one who are competing for the resources and end up with the scenario of race condition.

Cooperating processes are one who may or may not be aware about each other but they share memory and resources and work with cooperation

Answer 11)

Strong Semaphores strictly follow order (FIFO) to remove the processes from waiting queue

Weak semaphore has no such order is followed

Answer 12)

- Monitors are programming structures (classes and methods) which serves equivalently as semaphores.
- Monitors are easier to handle.

Answer 13)

Distinction between blocking and no blocking with respect to messages:

Non-locking sender / non-blocking receiver	Neither party is required to wait
Blocking sender / blocking receiver	• Both sender and receiver are blocked until the message is delivered • Sometimes referred to as a rendezvous • Allows for tight synchronization between processes
Non blocking sender / blocking reciever	• Sender continues sending but receiver is blocked until the requested message arrives • Most useful combination • Sends one or more messages to a variety of destinations as quickly as possible

Answer 14)

No. Busy waiting in more efficient only in the case where approximated waiting time is less than the time for pre-empt the process and reschedule it.

Answer 15)

Both codes prove out to be the same except that the semaphore code returns the value of number of processes waiting which is not given by the other code.

Answer 16)

```
While(1)
{
        semwait(santa);
        if(all_reindeer_ready){
                for all_waiting_reindeer{
                        semSignal(reindeer_wait);
                }
                for all_ reindeer{
                        semSignal(harness) ;
                }
                Deliver Tops;
                for all_reindeer{
                        semSignal(unharness);
                }
        }
        Else if(all_elves_ready){
                for all_waiting_elves{
                        semSignal(elf_wait);
                }
                for all_ reindeer{
                        semSignal(invite) ;
                }
                Consult;
                for all_reindeer{
                        semSignal(unharness);
                }
        }
}
```