Home Assignment

-Anmol Anubhai 121004

1) Events are:

I. Ready to Run-: process allocated to the CPU by dispatcher.
II. Ready to Non-Resident: memory is over committed and a process is temporarily swapped out of the memory
III. Run to Ready: Caused by Thin Quantum expiration
IV. Run to Blocked: process issues on I/O and other kernel requests.
V. Blocked to Ready: awaited event
VI. Blocked to Non Resident**-** memory is over committed and a process is temporarily swapped out of the memory

2)

For time 22	For time 37	For time 47
P1: Blocked (I/O) P3: Blocked (I/O) P5: Ready/Running. P7: Blocked (I/O) P8: Ready/Running.	P1: Ready/Running. P3: Ready/Running. P5: Blocked/Suspended. P7: Blocked (I/O) P8: Ready/Running.	P1: Ready/Running. P3: Ready/Running. P5: Ready/Suspended. P7: Blocked (I/O) P8: Exits

3)

The PID of the child process is returned in the parent's thread of execution and a 0 is returned in the child's thread of execution.

Output:

```
0

<child PID>
```

4)

Mode switch between threads may be better than a mode switch between processes because,

- The switching process requires the Operating System to process more information.
- Memory is shared by threads so lesser memory for thread execution or switching.

5)

Disadvantages of ULTs over KLTs.

- Thread switching does not require kernel mode privileges because all the thread data structure are stored within the user address space of a single process.
- Scheduling can also be application specific.

6)
Disadvantages of ULTs compared to KLTs.

- Many system calls are blocking when a ULT executes a system call. Not only the particular thread is blocked but also the threads along with that in the process are also blocked.
- In a pure ULT strategy, a multi-threaded application can't take advantage multiprocessing, a kernel assigns one process to only one processor, and therefore only a single thread within a process can be executed at a time.

7)
ULT thread structure of a process is not visible to the OS/kernel which schedules on the basis of the process. Hence, once a thread is blocked the entire process is blocked and consequently all the threads in that process ae also blocked.

8)
One to one mapping between the ULT and KLT allows on or more threads within the process to issue blocking system call while continue to run, because KLT in multi-threaded program enables at least on thread to issue a blocking system call independently without the influence of other threads and thereby allowing other threads to uninterruptly continue with their execution, however in single threaded system counter parts of the multi-threaded program machine generally spends a lot of time waiting for the I/O operations.

9)
If a process exits and there are still threads of that process running, they will not continue to run. All the threads share the same address space, all the threads are suspended at the same time.

10)
Competing process competes for resources whereas the co-operating process shares the resources.

11)
Difference between strong and weak semaphores is that the strong semaphores specifies in the order in which the processes are being removed from the waiting queue whereas the weak semaphores don't e.g. FIFO

12)
A monitor is a construct which allows the threads to have mutual exclusion. Same functionality as the semaphores.

13)
Distinction between blocking and non-blocking with respect to messages.

- Blocking send, Blocking receive: Both the sender and the receiver are blocked until the message is delivered.
- Non-Blocking send, Blocking receive: The sender may continue, the receiver is blocked until the requested message arrives.
- Non-Blocking send, Non-blocking receive: Neither the sender nor the receiver is required to wait.

14)
Busy waiting can be more efficient if the waiting time is lesser than the time it takes to preempt and reschedule a thread.

15)
Both are the same. s.count is for the number of process that can run simultaneously. When it turns to 0, every upcoming process which invoked semwait() should be blocked.

16) Santa Claus problem

```
Loop
{
        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);
                }
        }
}
```