www.network-uagrm.blogspot.com

```
// MESA EXAMINAORA  - INF323 SM. Sistemas Operativos I. 5/2011

/*1.- (Teorica)Se tiene un ADM-MEN con asignacion contigua multiple y se sabe que
       en este SO todos los procesos que corren en el tienen el mismo tamaño.
       ¿Qué algoritmo de asignacion sería mejor (FF O BF) ?
       ¿porque?
       R.- El ADM-MEN que utilizaria sera FF por el motivo que a la RAM se le
           ha asignado el ADM-MEN contigua multiple por lo tanto no es
           importante que los proceos sean del mismo tamaño ya que la
           RAM contiene espacios libres de distintos tamaños.*/

/*2.- Escribir el codigo de un planificador RR en el cual cada uno
      de los procesos reciban: 1q la primera vez, 2q la segunda vez,
      1q la tercera vez, 2q la cuarta vez.......
      Es decir cuando un proceso P sea PRUN la primera vez,
      recibirá 1q, cuando sea PRUN por 2da vez recibirá 2q,
      cuando se a PRUN 3ra vez 1q, etc (1,2,1,2,1,2,1,2....); */
      int cont = 0, max = 1;
      void planificador(){
        cont++;
        if(cont == max || Finalizo(PRUN)){
            if(Finalizo(PRUN)) FreeMem(PRUN);
            else{
                PRUN.Reg = CPU.Reg;
                Q.Meter(PRUN);
            }
            cont = 0;
            PRUN = Q.Sacar();
            if(max == 1) max = 2;
            else max = 1;
            CPU.Reg = PRUN.Reg;
        }
      }
```

```
/*3.- Escriba un planificador con baja de prioridad, para N colas
      Q[n], Q[N -1],...., Q[1], el cual solo permita la clonacion
      y baja de clones de los procesos de mas alta prioridad. es
      decir, los PCB's nativos con prioridad == N seran los
      unicos que podran generar clones; y los clones de esto
      se podran copiarse en la cola inferior.
      Recuerde que en una misma cola no pueden existir dos clones de
      un mismo PCB, Puede asumir que FreeMem libera los clones de todas
      las colas */
      int k = N;
      void planificador(){
        PCB AUX;
        if(Finalizo(PRUN)) Liberar(PRUN)
        else{
            PRUN.Reg = CPU. Reg;
            int i = PRUN.Prioridad;
            int x = 0;
            if(i == N){
                AUX = PRUN;
                Q[i].Meter(PRUN);
                x = 1;
            }
            if(i == k && i != N)
                Q[k].Meter(PRUN);
            if(k > 1 && AUX != NULL){
                if(!Q[k -1].Existe(AUX.PID))
                    Q[k -1].Meter(AUX);
                else x = 0;
            }
            PRUN.CantCop = PRUN.CantCop + x;
        }
        k--;
        if(k < 1) k = N;
        PRUN = Q[k].Sacar();
        while(kl.Existe(PRUN.PID)){
            Liberar(PRUN);
            PRUN = Q[k].Saca();
        }
        CPU.Reg = PRUN.Reg;

      }
```