Universidad Nacional Autónoma de México

Facultad de ingeniería

Sistemas Operativos

Planificación por lotería y multinivel

Grupo 6

Profesor: Gunnar Eyal Wolf Iszaevich

Integrantes:

Camacho Martínez Juan Carlos

Razo Villeda Fernando

Planificador por lotería

Este código es una implementación del algoritmo de planificación por lotería en Python. Primero se define una clase `Proceso` que representa cada proceso que será planificado. Cada proceso tiene un nombre, tiempo de llegada, duración, tiempo restante, prioridad y un indicador de si ha sido completado o no.

Luego se define una clase `planificadorLoteria`, que toma una lista de procesos y los utiliza para realizar la planificación por lotería. El planificadorLoteria tiene un método `imprimeTabla` que imprime una tabla que muestra información sobre los procesos que se van a planificar. El método `horario` es el que realiza la planificación.

El proceso de planificación comienza con la inicialización de algunas variables y un ciclo `while`. Dentro del ciclo `while`, se busca la lista de procesos listos (los procesos que ya han llegado y aún no se han completado) en ese momento. Si no hay procesos listos, se avanza un tick de tiempo y se continúa el ciclo `while`.

Si hay procesos listos, se calcula la lotería. Cada proceso recibe una cantidad de tickets proporcional a su duración. Luego se elige aleatoriamente un ticket y se asigna el proceso correspondiente como el ganador de la lotería. Se decrementa su tiempo restante y se actualiza el orden de ejecución y la tabla de ejecución. Si el proceso ha terminado, se marca como completado.

Después de que todos los procesos hayan sido planificados, se calculan las métricas de ejecución (tiempo de inicio, tiempo de finalización, tiempo de ejecución, tiempo de espera e índice de penalización) para cada proceso y se imprimen en una tabla.

Por último, se crea una lista aleatoria de procesos y se instancia el planificador, se imprime la tabla de procesos y se ejecuta la planificación utilizando el método `horario`.

```python
import random

class Proceso:
    def __init__(self, nombre, tiempo_llegada, duracion):
        self.nombre = nombre
        self.tiempo_llegada = tiempo_llegada
        self.duracion = duracion
        self.tiempo_restante = duracion
        self.prioridad = 0
        self.completado = False

class planificadorLoteria:
    def __init__(self, procesos):
        self.procesos = procesos
        self.tickets = sum([p.duracion for p in procesos])

    def imprimeTabla(self,procesos):
        print('\n\n🔴🔴🔴 Metodo de planificacion por loteria 🔴🔴🔴\n\n')
        print('Lista de procesos:')
        print("Nombre Duracion llegada")
        for i in range(len(procesos)):
            print(f"{procesos[i].nombre}      {str((procesos[i].duracion))+' '}      {procesos[i].tiempo_llegada}")
        print("*Inicia la ejecucion")


    def horario(self):
        tiempo = 0
        orden_ejecucion = ""
        tabla_ejecucion = []
        while not all(p.completado for p in self.procesos):
            print("t=%d" % tiempo)
            procesos_listos = [p for p in self.procesos if p.tiempo_llegada <= tiempo and not p.completado]
            if not procesos_listos:
                tiempo += 1
                continue

            # Calcular lotería
            loteria = []
            for proceso in procesos_listos:
                for i in range(proceso.duracion):
                    loteria.append(proceso)

            #Obtener al ganador
            ganador = random.choice(loteria)
            # Verificar si el proceso está completado
            if ganador.tiempo_restante == ganador.duracion:
                print("   Nuevo proceso!!!")
            ganador.tiempo_restante -= 1

            # Imprimir tick para el ganador
            print("    : {}: 1 tick".format(ganador.nombre))

            #Actualizar el orden de ejecución y la tabla
            orden_ejecucion += ganador.nombre
            if not tabla_ejecucion or tabla_ejecucion[-1][0] != ganador:
                tabla_ejecucion.append([ganador, tiempo, tiempo+1])
            else:
                tabla_ejecucion[-1][2] += 1

            # Verificar si el proceso está completado
            if ganador.tiempo_restante == 0:
                ganador.completado = True
                print("   Proceso terminado!!!")

            tiempo += 1

        # Calcular las métricas de ejecución
        tiempo_total = max([p.tiempo_llegada + p.duracion for p in self.procesos])
        metricas_ejecucion = {}
        for proceso in self.procesos:
            tiempo_inicio = [p[1] for p in tabla_ejecucion if p[0] == proceso][0]
            tiempo_final = [p[2] for p in tabla_ejecucion if p[0] == proceso][-1]
            tiempo_ejecucion = tiempo_final - tiempo_inicio
            tiempo_espera = tiempo_inicio - proceso.tiempo_llegada
            indice_penalizacion = tiempo_ejecucion / proceso.duracion
            metricas_ejecucion[proceso.nombre] = [tiempo_inicio, tiempo_final, tiempo_ejecucion, tiempo_espera, indice_penalizacion]

        # Imprimir la ejecución y la tabla
        print("\n\nPlanificación realizada: " + orden_ejecucion)
        print("\n\nTabla de ejecución:")
        print('{:<7}  {:<7} {:<7} {:<7} {:<7} {:<7}'.format('Proceso','Inicio','Fin','T','E','P'))
        for proceso in self.procesos:
            metricas = metricas_ejecucion[proceso.nombre]
            print('{:<7}  {:<7} {:<7} {:<7} {:<7} {:<7}'.format(proceso.nombre,metricas[0],metricas[1], metricas[2], metricas[3], metricas
```

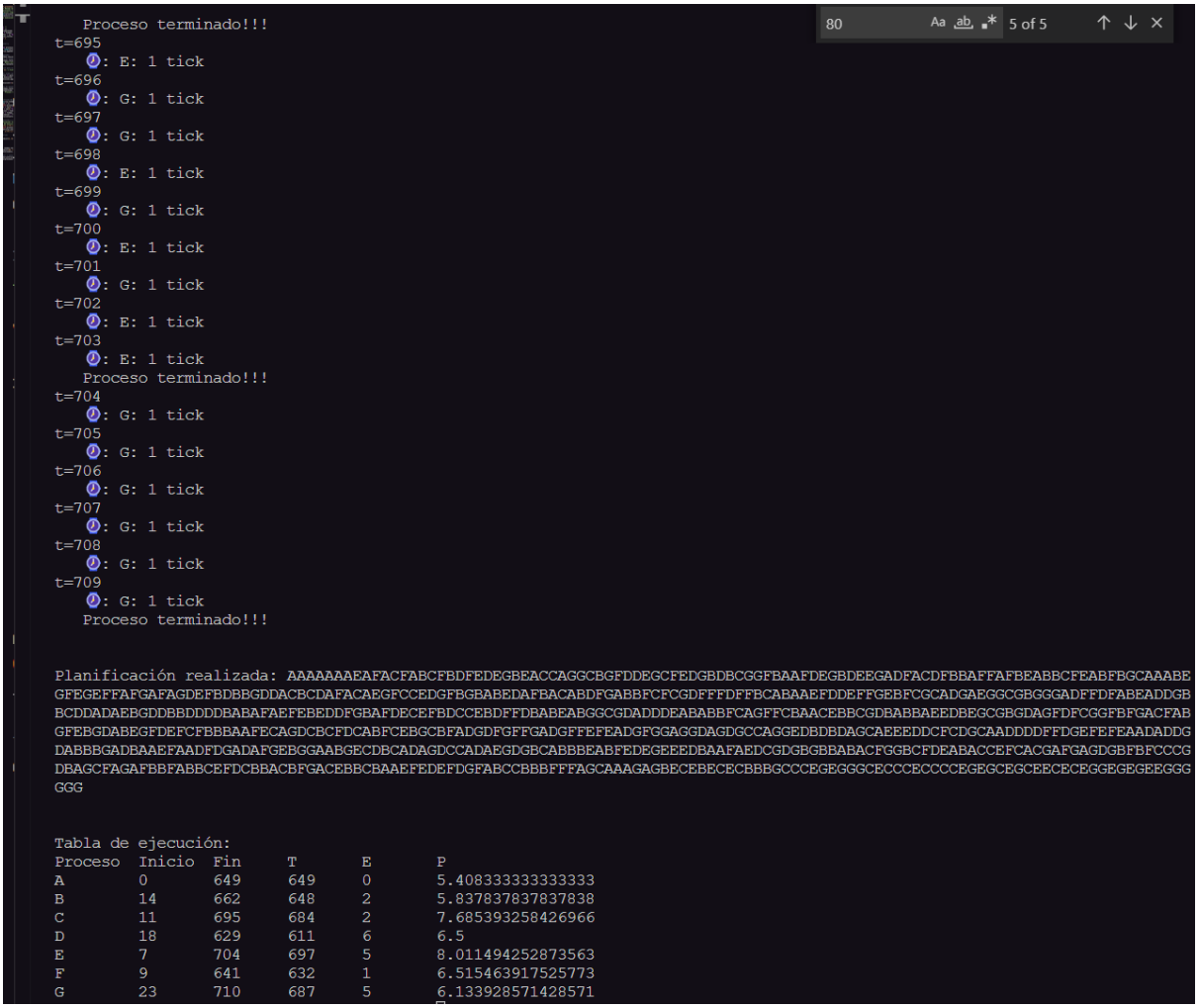
```
85
86  # Generar procesos aleatorios
87  procesos = []
88  for i in range(random.randint(5, 8)):
89      nombre = chr(i+65)
90      if i==0:
91          tiempo_llegada=0
92      else:
93          tiempo_llegada = random.randint(0, 20)
94      duracion = random.randint(80, 120)
95      procesos.append(Proceso(nombre, tiempo_llegada, duracion))
96
97  #Instanciar el planificador y ejecutar la planificación
98  planificador = planificadorLoteria(procesos)
99  planificador.imprimeTabla(procesos)
100 planificador.horario()
```

Resultado del programa:

Como se puede observar, al compilar el programa primero se imprime la lista inicial con todos los procesos, incluyendo detalles como el nombre, la duración y la llegada.

Posteriormente se muestra la ejecución de los procesos hasta terminar con el número de boletos, luego se imprime la planificación realizada para que finalmente se imprima la tabla de ejecución.

```
Lista de procesos:
Nombre Duracion llegada
A      9       0
B      10       19
C      6       8
D      6       9
E      6       0
F      6       16
G      8       13
H      8       0
*Inicia la ejecucion
t=0
   Nuevo proceso!!!
   🕐: H: 1 tick
t=1
   Nuevo proceso!!!
   🕐: A: 1 tick
t=2
   Nuevo proceso!!!
   🕐: E: 1 tick
t=3
   🕐: H: 1 tick
t=4
   🕐: H: 1 tick
t=5
   🕐: A: 1 tick
t=6
   🕐: E: 1 tick
```



Planificación por retroalimentación multinivel

Para esta planeación se tomaron en cuenta 3 niveles de colas, donde cada proceso será asignado por el tiempo de duración de cada proceso en donde de forma inicial si es menor a 95 irá a la cola 1, si es mayor a 95 y menor que 115 irá a la cola 2 y por último irá a la cola 3 para los valores mayores a 115. Una vez el programa inicie su ejecución el tiempo de los procesos irá cambiando, por lo que cada proceso una vez cambie su tiempo de ejecución faltante será reasignado a una nueva cola dependiendo de que tanto le falte para terminar.

Para el desarrollo de este código se tomó en cuenta el ejemplo proporcionado por el profesor por lo que primero se crean de 5 a 8 procesos de forma aleatoria, cada uno con una duración, tiempo de llegada y nombre distinto. Después se coloca la lista de los procesos que se llevarán a cabo e inicia la ejecución del método. En esta ejecución se colocaron los tiempos transcurridos y que proceso estuvo activo en esa brecha de tiempo.

Además de que se agregó un mensaje con el que podemos visualizar cuando un proceso concluyó su ejecución y ya no lo veremos nuevamente en la ejecución del resto del programa.

Por último se coloca la tabla de ejecución donde podemos ver los datos que se recopilaron a lo largo de la simulación de la planificación.

```python
from collections import deque
from random import randint

procesos = []
primer_proc = 'A'

for i in range(randint(5,8)):
    # Genero los 4 a 8 procesos aleatorios
    procesos.append({'nombre': chr( ord(primer_proc)+i ),
                     'llegada': randint(0, 10*i),
                     'duración': randint(80,120)
                     })

print('Lista de procesos:')
print('{:<7}  {:<7} {:<7}'.format('Nombre','Duracion','llegada'))
for proc in procesos:
    print('{:<7}  {:<7} {:<7}'.format(proc['nombre'], proc['duración'], proc['llegada']))

# Definimos las colas para cada nivel
colas = [deque() for i in range(3)]

# Asignamos los procesos a su nivel correspondiente
for proc in procesos:
    if proc['duración'] < 95:
        colas[0].append(proc)
    elif proc['duración'] < 110 and proc['duración'] > 95:
        colas[1].append(proc)
    else:
        colas[2].append(proc)

# Establecemos el quantum para cada nivel
quantum = [5, 15, 25]

```

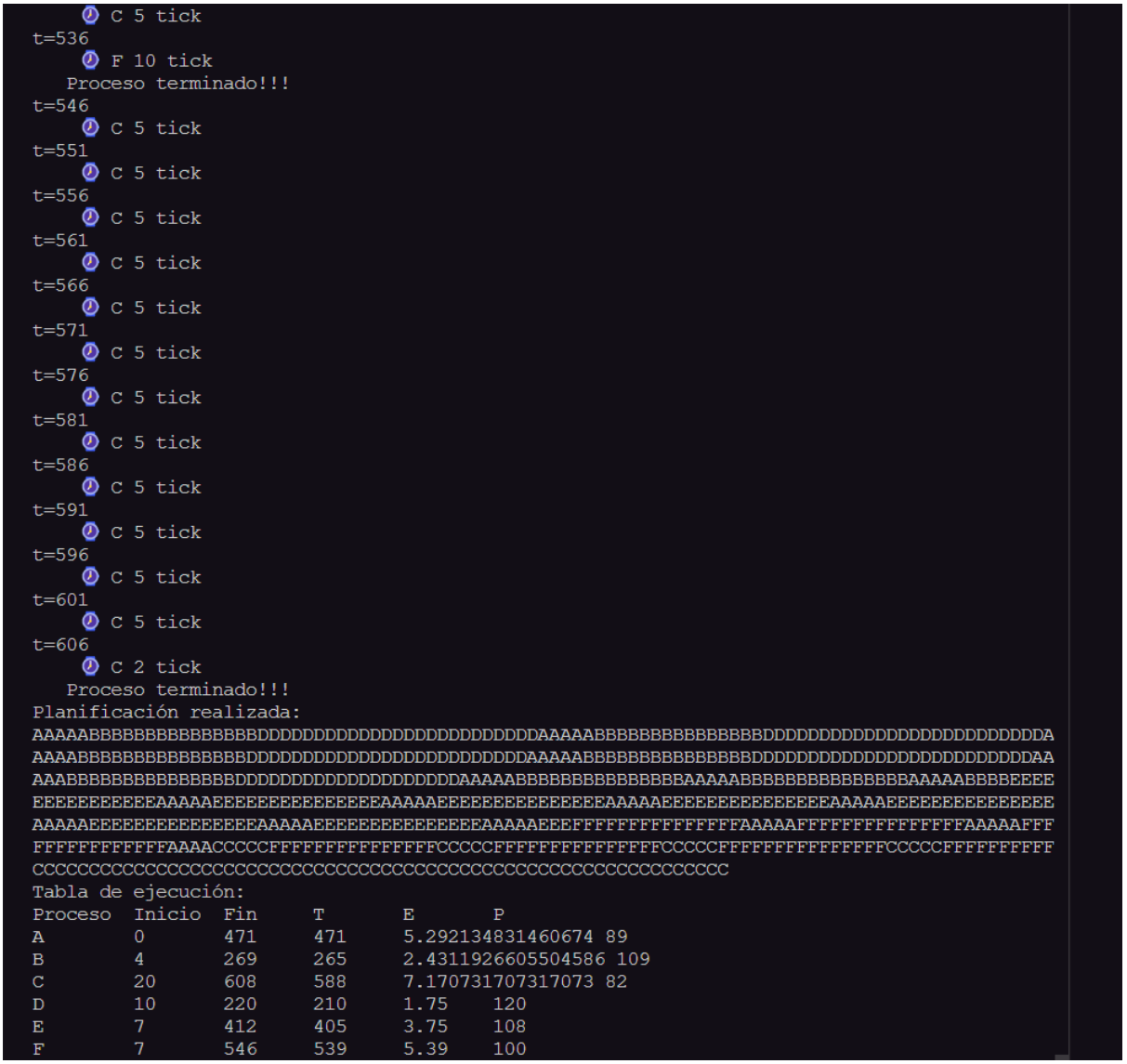
```python
t = 0
res = ''
procesos_terminados = []
print('* Inicia ejecución')
while any(colas):
    for i, cola in enumerate(colas):
        # Ejecutamos los procesos de la cola actual
        while cola:
            p = cola[0]
            if p['nombre'] not in procesos_terminados:
                print("t=%d" % t)
            # Ejecutamos el proceso actual por el quantum establecido
            duracion_restante = p['duración'] - p.get('ejecutado', 0)
            duracion_ejecucion = min(duracion_restante, quantum[i])
            res += p['nombre'] * duracion_ejecucion
            t += duracion_ejecucion
            p['ejecutado'] = p.get('ejecutado', 0) + duracion_ejecucion
            if duracion_ejecucion > 0:
                print("     %s %d tick" % (p['nombre'], duracion_ejecucion))
            # Movemos el proceso actual al nivel anterior si no ha terminado su ejecución
            if duracion_ejecucion < duracion_restante:
                if i-1 >= 0:
                    colas[i-1
                else:
                    colas[i].append(p)
                break
            else:
                if p['nombre'] not in procesos_terminados:
                    print("   Proceso terminado!!!")
                    procesos_terminados.append(p['nombre'])
                    p['fin'] = t
                    p['T'] = p['fin'] - p['llegada']
                    p['E'] = p['T'] / p['duración']
                cola.popleft()

print("Planificación realizada: \n" + res)
print("Tabla de ejecución:")
print('{:<7}  {:<7} {:<7} {:<7} {:<7} {:<7}'.format('Proceso','Inicio','Fin','T','E','P'))
for proc in procesos:
    # print("%2s      %d    %d    %d  %.2f  %.2f" % (proc['nombre'], proc['llegada'], proc['fin'], proc['T'], proc['E'], proc['duración']))
    print('{:<7}  {:<7} {:<7} {:<7} {:<7} {:<7}'.format(proc['nombre'], proc['llegada'], proc['fin'], proc['T'], proc['E'], proc['duración']))
promedio_E = sum([proc['E'] for proc in procesos]) / len(procesos)
promedio_P = sum([proc['duración'] / proc['E'] for proc in procesos]) / len(procesos)
```

(method) append(__x: Any, /) -> None

See Real World Examples From GitHub

Resultados del programa:

```
Lista de procesos:
Nombre   Duracion llegada
A        89      0
B        109     4
C        82      20
D        120     10
E        108     7
F        100     7
* Inicia ejecución
t=0
    A 5 tick
t=5
    B 15 tick
t=20
    D 25 tick
t=45
    A 5 tick
t=50
    B 15 tick
t=65
    D 25 tick
t=90
    A 5 tick
t=95
    B 15 tick
t=110
    D 25 tick
t=135
    A 5 tick
t=140
    B 15 tick
t=155
    D 25 tick
t=180
    A 5 tick
t=185
    B 15 tick
t=200
    D 20 tick
   Proceso terminado!!!
t=220
    A 5 tick
t=225
    B 15 tick
t=240
    A 5 tick
t=245
    B 15 tick
t=260
    A 5 tick
t=265
    B 4 tick
   Proceso terminado!!!
```



- ¿Cómo se compararía este método con los otros abordados?
  En comparación con los otros métodos vistos nos parece que el método retroalimentación multinivel es bastante similar al método Round Robin ya que en este también se utilizan los quantums, sin embargo en ambos métodos se utilizan de forma distinta y a nuestro parecer el método por retroalimentación por nivel termina siendo un poco más justo que el método Round Robin.
  Ahora realizando la comparación del método por lotería nos parece que puede llegar a ser más justo que por otros métodos, sin embargo por la misma lotería perdería eficiencia al estar la posibilidad que un proceso está congelado por más tiempo del que debiera, pudiendo llegar a la inanición..

- ¿Para qué tipo de carga es más apto y menos apto?
  La planificación por lotería es más adecuada para sistemas de tiempo compartido, donde varios procesos comparten los recursos de la CPU, y la carga de trabajo no es predecible y puede ser muy variable. Es muy útil en estos casos porque asigna un número de lotería a cada proceso, de modo que cuanto más tiempo de CPU necesite un proceso, más probabilidades tendrá de ganar la lotería.

  Por otro lado, la planificación multiuso es más adecuada para sistemas de tiempo real, donde se espera que los procesos cumplan plazos determinados y la carga de trabajo es predecible y más estable. La planificación multiuso funciona mejor en estos casos porque permite que los procesos tengan prioridades diferentes y se asigna la CPU en función de su prioridad.

- ¿Qué tan susceptible resulta a producir inanición?

  La planificación por lotería puede ser susceptible a producir inanición si los procesos con menor duración nunca son seleccionados como ganadores. Esto puede ocurrir si el número de tickets asignados a cada proceso no se asigna adecuadamente, dando una ventaja injusta a los procesos más largos. Si se asigna un número de tickets proporcional a la duración de cada proceso, se reduce la probabilidad de inanición.

  En la planificación multiuso, la inanición también puede ocurrir si se priorizan procesos de alta prioridad y no se permiten que los procesos de baja prioridad se ejecuten en absoluto. Sin embargo, los sistemas operativos modernos utilizan diferentes técnicas para evitar la inanición, como dar una prioridad relativa a cada proceso en función del tiempo transcurrido desde que se ejecutó por última vez.

- ¿Qué tan justa sería su ejecución?

  La planificación por lotería es considerada justa porque cada proceso tiene una probabilidad de ser elegido en cada ronda. A medida que un proceso completa su duración, la cantidad de tickets en la lotería se reduce y la probabilidad de ser elegido disminuye. Esto significa que los procesos más cortos tienden a ser completados antes y los más largos tienen una probabilidad de ser completados proporcional a su tamaño.

  Sin embargo, esta justicia no es absoluta, ya que todavía existe la posibilidad de que un proceso más pequeño se quede sin ser elegido durante varias rondas consecutivas y, por lo tanto, experimente una mayor espera en la cola. Además, los procesos que llegan antes pueden tener más oportunidades de ser elegidos si tienen una duración similar a los procesos que llegan más tarde.

En general, la planificación por lotería es una de las técnicas más justas de planificación de procesos, pero no es perfecta y puede beneficiar a ciertos tipos de procesos en detrimento de otros.

La justicia de la ejecución de la planificación multinivel depende de la equidad en la asignación de recursos y la consideración de las necesidades de todos los proyectos.

- ¿Qué modificaciones requeriría para planificar procesos con necesidades de tiempo real? (aunque sea tiempo real suave).

1. **Incluir prioridades de tiempo real:** se deben agregar prioridades de tiempo real a los procesos que requieren tiempo real para que el planificador pueda darles prioridad sobre otros procesos no críticos.
2. **Asignación de recursos:** se debe garantizar que los recursos críticos estén disponibles para los procesos en tiempo real en todo momento. Esto puede incluir la asignación exclusiva de recursos críticos, como CPU, memoria y ancho de banda de red.
3. **Tiempo de respuesta garantizado:** se deben garantizar los tiempos de respuesta de los procesos de tiempo real. Esto puede lograrse mediante la asignación de cuotas de tiempo para cada proceso, lo que garantiza que cada proceso reciba una cantidad adecuada de tiempo de CPU.
4. **Eliminación de la sobrecarga de tareas:** para garantizar que los procesos de tiempo real no se vean afectados por la sobrecarga de tareas, se puede establecer un límite máximo de procesos que se pueden ejecutar en el sistema.
5. **Implementación de técnicas de preempción:** para garantizar que los procesos de tiempo real tengan la oportunidad de ejecutarse, se pueden implementar técnicas de preempción que permitan al planificador interrumpir la ejecución de procesos no críticos en caso de que se agoten las cuotas de tiempo asignadas.