UNIVERSIDAD DE GUADALAJARA
Red Universitaria e Institución Benemérita de Jalisco
CUCEI
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Seminario de Solución de Problemas de Sistemas Operativos

Profesor: Becerra Velázquez, Violeta Del Rocío

Alumno: Bustos Ruiz Daniel

Código: 215466901

INCO

Sección: D03

NRC: 119896

Calendario 2024A

Actividad 10: Round Robin

14/04/2024

Índice

Objetivo

El objetivo de esta actividad es realizar el algoritmo de round robin basándonos en el anterior algoritmo de planificación llamado First Come First Served, el cual tiene un comportamiento similar, sin embargo, se agrega un quantum o tiempo el cual es el que utilizará cada proceso para hacer uso del CPU.

Desarrollo

Como se menciono anteriormente en el objetivo de la actividad, se hace uso del programa anterior que consiste en el algoritmo de planificación First Come First Served. Como requisito se pide que además de preguntar al usuario cuantos procesos van a entrar, se le deberá preguntar de cuantos segundos debe consistir el quantum. Debido a las condiciones en las que se generan los tiempos de los procesos que comprenden entre 5 y 18, se valida que al pedir el quantum este sea dentro de estos límites.

```
while True:
  try:
    quantum = int(input("Ingrese el quantum deseado: "))
    if quantum < 5 or quantum >= 18:
      print("Por favor, ingrese un numero valido de quantum.")
      input("Presione <enter> para continuar")
      os.system("cls")
      continue
    break
  except ValueError:
    print("Por favor, ingrese un número entero.")
    input("Presione <enter> para continuar")
    os.system("cls")
    continue
```

El quantum será la cantidad de segundos que tendrá el proceso para hacer uso del CPU, no se debe confundir con el tiempo máximo estimado, ya que ese tiempo es el total que le tomará al proceso ejecutarse.

Este número de quantum será agregado a cada proceso creado, como se muestra a continuación en la función encargada de generar todos los procesos, de la misma

forma lo hace la función que genera un nuevo proceso en caso de presionar la tecla N.

```
def generaProcesos(p: int, q: int):
  auxProcesos = []
  #*Genera copia de cada lista que obtiene
  ids = getIds(p)
  tiempos = getTiempos(p)[:]
  operaciones = getOperaciones(p)[:]
  for i in range(0, p):
    tiempo = tiempos.pop()
    operacion = operaciones.pop()
    auxProcesos.append({
      "id": ids.pop(),
      "tiempo": tiempo,
      "tiempoRestante": tiempo,
      "operacion": operacion["operacion"],
      "operacionStr": operacion["operacionStr"],
      "fNum": operacion["fNum"],
      "sNum": operacion["sNum"],
      "resultado": 0,
      "error": False,
      "tiempoTrans": 0,
      "tiempoLlegada": 0,
      "tiempoFinalizacion": 0,
      "tiempoRetorno": 0,
      "tiempoEspera": 0,
      "tiempoRespuesta": 0,
      "tiempoServicio": 0,
      "timeOut": 0,
      "banderaRespuesta": False,
      "quantum": q,
      "bloqueado": False
    })
  return auxProcesos
```

Dentro del bucle principal, en lugar de que la ejecución del proceso sea de acuerdo con el tiempo máximo estimado, será de acuerdo con el quantum y mientras el tiempo restante del proceso no sea igual a 0. En el caso de que el quantum termine o que el proceso sea enviado a bloqueado, se reinicia el quantum y de igual forma

que en el programa anterior, será enviado al final de la cola de los listos o de bloqueados, dependiendo el caso.

```
while len(procesosListos) > 0 or len(procesosBloqueados) > 0 or
len(procesosNuevos) > 0:

  #*Toma el primer proceso de la lista de listos
  if len(procesosListos) > 0:
    procesoEjecutar = procesosListos.pop(0)

  #*Comprueba si hay espacio libre en la memoria principal
  if (len(procesosListos) + len(procesosBloqueados)) + 1 < 4 and
len(procesosNuevos) > 0:
    newListo = procesosNuevos.pop(0)
    newListo["tiempoLlegada"] = contadorGlobal
    procesosListos.append(newListo)

  #!Ejecuta el proceso de acuerdo al quantum
  if procesoEjecutar["tiempoRestante"] > 0:
    while procesoEjecutar["quantum"] > 0 and
procesoEjecutar["tiempoRestante"] > 0:

      if procesoEjecutar["banderaRespuesta"] == False:
        procesoEjecutar["tiempoRespuesta"] = contadorGlobal
        procesoEjecutar["banderaRespuesta"] = True

      contadorGlobal += 1

      #*Si todos estan en bloqueado, no tiene porque modificar estos
valores
      if len(procesosBloqueados) < 4:
        procesoEjecutar["tiempoRestante"] -= 1
        procesoEjecutar["tiempoTrans"] += 1
        procesoEjecutar["quantum"] -= 1

      #*Actualiza los tiempos de los bloqueados
      if len(procesosBloqueados) > 0:
        for proceso in procesosBloqueados:
          proceso["timeOut"] += 1
          #*En caso de que se termine el tiempo de bloqueo
          if proceso["timeOut"] == 8:
            proceso["bloqueado"] = False
            procesosListos.append(procesosBloqueados.pop(0))
```

```
      #?Funciones en caso de presionar una tecla
      if msvcrt.kbhit():
        char = msvcrt.getch()
        if char.lower() == b'p':
          pausa = True
        elif char.lower() == b'e':
          if procesoEjecutar["tiempoRestante"] > 0:
            #*Se reinicia el quantum ya que sera enviado a la cola de
listos otra vez
            procesoEjecutar["bloqueado"] = True
            procesoEjecutar["quantum"] = quantum
            procesosBloqueados.append(procesoEjecutar)
            break
        elif char.lower() == b'w':
          procesoEjecutar["tiempoRestante"] = 0
          procesoEjecutar["error"] = True
          calculaTiempos(procesoEjecutar, contadorGlobal, 1)
          # procesosTerminados.append(procesoEjecutar)
          # procesoEjecutar = None
          break
        elif char.lower() == b'n':
          #*Crea un nuevo proceso
          #*Comprueba si hay espacio en la memoria principal
          if (len(procesosListos) + len(procesosBloqueados)) + 1 < 4:
            nuevoProceso = generaProceso(quantum)
            nuevoProceso["tiempoLlegada"] = contadorGlobal
            procesosListos.append(nuevoProceso)
          else:
            procesosNuevos.append(generaProceso(quantum))
        elif char.lower() == b'b':
          pausa = True
          mostrarTabla = True
      #*Bucle infinito hasta que se quite la pausa
      if pausa:
        while True:
          if msvcrt.kbhit():
            char = msvcrt.getch()
            if char.lower() == b'c':
              pausa = False
              mostrarTabla = False
              break
```

```
          if mostrarTabla:
            #*Se imprime la tabla de pausa
            sys.stdout.write('\033[H')
            sys.stdout.write(tabulate(getTablaPausa(procesosTerminado
s, procesosNuevos, procesosListos, procesosBloqueados,
procesoEjecutar, contadorGlobal)[:], headers=columnasPausa,
tablefmt='fancy_grid'))
            sys.stdout.flush()
            time.sleep(0.1)

      #*Muestra la tabla
      linea = getTablaEjecucion(procesosListos, procesosBloqueados,
procesosNuevos, procesosTerminados, procesoEjecutar, contadorGlobal,
quantum)
      sys.stdout.write('\033[H')
      sys.stdout.write(tabulate([linea], headers=columnasEjecucion,
tablefmt='fancy_grid'))
      sys.stdout.flush()
      time.sleep(1)
```

Conclusión

El programa fue sencillo de realizar en términos generales, ya que solo era necesario agregar el quantum a cada proceso y que el proceso usara ese tiempo para ejecutarse. Sin embargo, lo que fue algo complicado era tomar en cuenta algunos aspectos como reiniciar el quantum cada que el proceso terminaba de usar el CPU o que fuera enviado a bloqueado, además que se debía comprobar en cada momento si el proceso había terminado de ejecutarse o no.

Con esta actividad se demuestra el comportamiento de otro algoritmo de planificación, este algoritmo llamado Round Robin funciona de forma bastante similar a otros algoritmos de planificación, lo que nos hace preguntarnos cual seria el objetivo de los demás, sin embargo, este debería ser utilizado dependiendo de las necesidades del problema que se tenga.