Documentación Proyecto 2

Planificadores

Bernal Téllez Luis Gustavo

Toledo Dilan Gerson Toledo Bedia

Sistemas Operativos

Profesor Ing. Gunnar Wolf

A continuación, se presentará un código en Python capaz de simular una planificación de entre 5 a 8 procesos con máximo de 80 a 120 ticks de duración total. Los procesos aparecerán en cualquier momento de la simulación. Para ello, se toma decisiones para así lograr la planificación, y es eso lo que describiremos a continuación.

Código:

```
from random import randint

procesos = []
primer_proc = 'A'
rango=randint(5,8)

for i in range(rango):
    # Genero los 5 a 8 procesos aleatorios
    llegada=randint(0, 15*i)
    if(i==(rango-1)and llegada<80):
        llegada=80
        print(i)
        #para que la instrucción de 80-120 ticks se cumpla, forzamos a
que el ultimo parametro lo haga
    procesos.append({'nombre': chr( ord(primer_proc)+i ),
                     'llegada': llegada,
                     'duración': randint(4,10)
                     })
print('Lista de procesos:')
print('Proceso   Llegada   Duración')
for proc in procesos:
    print("%2s         %3d        %3d" % (proc['nombre'],
proc['llegada'], proc['duración']))
t = 0
res = ''
# 'A' llega siempre en 0 ('llegada' es aleatorio entre 0 y 5*0)
print('* Inicia ejecución')
for p in sorted(procesos, key=lambda p: p['llegada']):
    print("t=%d" % t)
    # Manejamos el caso de que no haya ningún proceso listo para
    # ejecutar
    if t < p['llegada']:
        demora = p['llegada'] - t
```

```
        res += '-' * demora
        t += demora
        print("    ... %d tick" % demora)
        print("t=%d" % t)
    # El proceso se ejecuta por toda la carga de trabajo que tiene

    res += p['nombre'] * p['duración']
    t += p['duración']
    print("    ⏱ %s %d tick" % (p['nombre'], p['duración']))

print("Planificación realizada: \n" + res)
print("\n\nDuración total: %d" % t)

print("\n\nTabla de ejecución:")
print("Proceso Inicio Fin        T        E      P          R")
PromedioT=0
PromedioE=0
PromedioP=0
PromedioR=0
for proc in procesos:
    inicio = res.index(proc['nombre'])  # primer tick en el que se
ejecuta el proceso
    fin = inicio + proc['duración']    # último tick en el que se ejecuta
el proceso
    t = fin - proc['llegada']          # tiempo total que toma el trabajo
    PromedioT+=t/rango
    espera = t - (fin-inicio)     # tiempo que esperó el proceso para
ejecutarse
    PromedioE+=espera/rango
    penalizacion = t/(fin-inicio) *1.00      # tiempo de penalización
    PromedioP+=penalizacion/rango
    respuesta = 1/penalizacion # fracción de tiempo de respuesta
    PromedioR+=respuesta/rango
    print("%2s      %2d     %2d        %2d       %2d    %2f  %2f" %
(proc['nombre'], inicio,
                                                fin, t, espera,
penalizacion,
                                                  respuesta))
print('Prom                %2f %2f
%2f  %2f'%(PromedioT,PromedioE,PromedioP,PromedioR))
```

Tenemos una planificación de “Primero llegado, primero servido(FCFS), en donde la primera línea importa la función randint del módulo random.

Luego, el código define una lista vacía llamada procesos y una variable primer_proc inicializada en 'A'.

La variable rango se establece como un número entero aleatorio entre 5 y 8, ambos inclusive, utilizando la función randint.

Un bucle for crea un número aleatorio de procesos en el rango de rango. Cada proceso se define como un diccionario con las siguientes claves: 'nombre', 'llegada' y 'duración'. La clave 'nombre' es una letra que se asigna en orden secuencial utilizando el valor de primer_proc y un desplazamiento entero. La clave 'llegada' es un número entero aleatorio entre 0 y un múltiplo de 15 que depende del índice del bucle for. La clave 'duración' es un número entero aleatorio entre 4 y 10.

Después de generar los procesos aleatorios, el código muestra la lista de procesos generada.

El código a continuación inicia la planificación y ejecución de los procesos. Primero, se imprime un mensaje que indica el inicio de la ejecución. Luego, los procesos se ordenan por su tiempo de llegada en orden ascendente utilizando la función sorted con una función lambda como clave de ordenamiento.

A continuación, un bucle for itera sobre los procesos ordenados. Para cada proceso, el código verifica si hay tiempo de inactividad antes de que el proceso llegue. Si hay tiempo de inactividad, se agrega un carácter '-' a la variable res para indicar la inactividad, y se incrementa el tiempo t por la cantidad de tiempo de inactividad. Luego, el proceso se ejecuta durante su tiempo de duración, y se agrega su letra correspondiente a la variable res. Finalmente, se incrementa el tiempo t por la cantidad de tiempo de duración del proceso.

El código luego imprime la planificación realizada y la duración total de la ejecución.

Finalmente, se muestra una tabla con información detallada de la ejecución de cada proceso. Para cada proceso, se muestra su letra correspondiente, el tiempo de inicio, el tiempo de finalización, el tiempo total que tomó el trabajo, el tiempo de espera, la penalización de tiempo, la fracción de tiempo de respuesta y su promedio respectivo.