



TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

ARQUITECTURA DE COMPUTADORAS

PRACTICA 5

**Presenta:**

Cuevas Hernández Erik Israel-22620202

Osorio Ramírez Marlene Maricela-22620269

Sarmiento Ruiz Edgar Mauricio-22620066

INGENIERIA EN SISTEMAS COMPUTCIONALES

**SEMESTRE:**

5

DOCENTE: ING. EDWARD OSORIO SALINAS

TLAXIACO, OAXACA, 27 DE OCTUBRE DEL 2024



## OBJETIVO

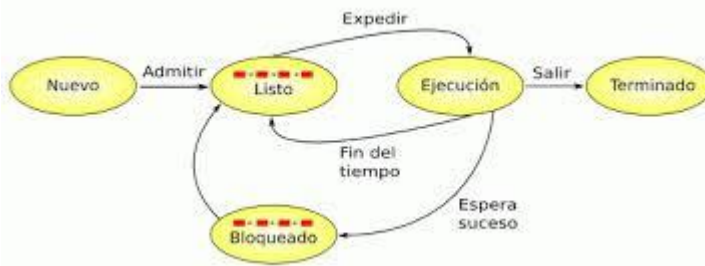
El objetivo de esta práctica es simular la administración de procesos/tareas en la CPU. Para ello, deberá investigar el funcionamiento de un sistema operativo y realizar un resumen de los pasos que se llevan a cabo en la administración de procesos/tareas en la CPU, estas estrategias son conocidas como “*scheduling*”.






### 3.- Pasos que se llevan a cabo en la administración de procesos / tareas en la CPU.

Cada proceso es un procesador virtual en donde se ejecuta una aplicación o una herramienta del sistema operativo. El núcleo de encargarse entonces de administrar recursos del hardware del computador para que sean asignados convenientemente a los procesos.

#### Estados de un proceso

Mientras un proceso se ejecuta puede pasar por distintos estados. Estos se aprecian en la siguiente figura.



-  **Nuevo:** el núcleo está obteniendo los recursos que necesita el procesador para correr, como por ejemplo memoria o disco.
-  **Ejecución:** el proceso está en posesión del procesador, el que ejecuta en sus instrucciones.
-  **Bloqueado:** el proceso espera que se lea un sector del disco, que llegue un mensaje de otro proceso, que transcurra un intervalo de tiempo, o que termine otro proceso.
-  **Listo:** el proceso está activo, pero no está en posesión del procesador.
-  **Terminado:** el proceso terminó su ejecución, pero sigue existiendo para que otros procesos puedan determinar que terminó.



## 4.- Desarrollo

### 4.1 Estrategias de Scheduling

#### 1. First Come First Served

- La implementación es fácil a través de un a cola FIFO
- Los procesos son ejecutados en el orden que llegan a la cola de procesos listos.
- Es adecuado para sistemas por lotes (batch).
- Es un algoritmo no expropiativo: una vez que el procesador le es asignado a un proceso este lo mantiene hasta que se termine o se bloquea. Por ejemplo, al generar un pedido de E/S.

#### 2. Shortest Job First

- El algoritmo asocia a los procesos el largo de su próximo CPU- burst.
- Cuando el procesador queda disponible se le asigna al proceso que tenga menos CPU-burst.
- Si dos procesos tienen el mismo CPU-burst se desempata de alguna forma.
- Su funcionamiento depende de conocer los tiempos de ejecución lo cual en la mayoría de los casos no sucede.
- Dos esquemas
  - No expropiativo: una vez se le asigna el procesador a un proceso no se le podrá quitar.
  - Expropiativo: si un nuevo proceso aparece en la lista de procesos listos con menor CPU-burst, se le quita la CPU para asignarle al nuevo proceso.
- Este algoritmo es óptimo par el tiempo de espera, pero requiere que todos os procesadores participantes estén al comienzo y además hay que haber el tiempo del próximo CPU-burst.
- Es usado para la planificación de largo plazo más que para planificación de corto plazo.

#### 3. Round Robin

- A cada proceso se le brinda un intervalo de tiempo para el uso del procesador.
- Al finalizar el tiempo, el procesador le es expropiado y vuelve al estado pronto al final de la cola.





- c. Es fácil de implementar ya que solamente es necesario una cola de procesos listos. Cuando un proceso consume su quantum incide en los tiempos de retorno.
- d. El quantum debe de ser bastante mayor a lo que lleva realizar un cambio de contexto, sino se tendrá mucho overhead a su vez, el tiempo de quantum incide en los tiempos de retorno.
- e. Es ideal para los sistemas de tiempo compartido.
- f. No hay posposición indefinida.

## 4.2 FIFO

Para simular los procesos ocupamos SimPy, la cual es una biblioteca de simulación de eventos discretos para Python, esta permite modelar sistemas que cambian con el tiempo.

1.- para esto primero necesitamos instalar Python, en conjunto con la librería SimPy, una vez hecho procedimos a realizar el código que nos permitirá hacer uso de la simulación.

Este código este documentado con las actividades que realiza





```

Practica-5.md  simulacion.py 1 X
C: > Users > Israel Cuevas > Documents > Arquitectura de computadoras > simulacion.py > tarea

1 import simpy
2 import random
3
4 # Definimos el proceso que representa una tarea
5 def tarea(env, nombre, tiempo_ejecucion):
6     print(f'{nombre} llega al sistema en {env.now}')
7     yield env.timeout(tiempo_ejecucion) # Simula el tiempo de ejecución
8     print(f'{nombre} ha terminado su ejecución en {env.now}')
9
10 # Generador de tareas
11 def generador_tareas(env, cola):
12     for i in range(5): # Genera 5 tareas
13         tiempo_ejecucion = random.randint(1, 5) # Tiempo de ejecución aleatorio
14         nombre_tarea = f'Tarea {i+1}'
15         env.process(tarea(env, nombre_tarea, tiempo_ejecucion))
16         yield env.timeout(random.randint(1, 2)) # Tiempo entre llegadas
17
18 # Configuración del entorno
19 env = simpy.Environment()
20 cola_tareas = simpy.Store(env) # Usamos una cola para las tareas
21
22 # Iniciamos el generador de tareas
23 env.process(generador_tareas(env, cola_tareas))
24
25 # Ejecutar la simulación
26 env.run()
27

```

Después se procede a ejecutar desde la ubicación en la que se encuentra,

Este equipo > Documentos > Arquitectura de computadoras				
Nombre	Fecha de modificación	Tipo	Tamaño	
tec-nm-tlaxiaco-arqui-compu	27/10/2024 10:37 p. m.	Carpeta de archivos		
investigacion	14/10/2024 07:03 p. m.	Documento de Mi...	2,370 KB	
simulacion	27/10/2024 10:29 p. m.	Archivo de origen ...	1 KB	

para esto en donde se encuentra la ubicación anotamos CMD, que es el símbolo de sistema, una vez abierto podemos ejecutar nuestro script, haciendo uso de:

Python simulación.py. Donde simulación es el nombre y .py es la extensión del archivo.

```

C:\Users\Israel Cuevas\Documents\Arquitectura de computadoras>python simulacion.py
Tarea 1 llega al sistema en 0
Tarea 1 ha terminado su ejecución en 1
Tarea 2 llega al sistema en 2
Tarea 3 llega al sistema en 4
Tarea 2 ha terminado su ejecución en 5
Tarea 4 llega al sistema en 6
Tarea 5 llega al sistema en 8
Tarea 3 ha terminado su ejecución en 9
Tarea 5 ha terminado su ejecución en 9
Tarea 4 ha terminado su ejecución en 11

```

como se puede apreciar se muestran las tarea/procesos conformen van llegando y el



tiempo en el que llega al sistema, espetando así al primero en llegar, primero en salir (FIFO).

### 4.3 SJF

Una vez mas haciendo uso de SimPy, pero para esta usamos a Shortest Job First.}

- Hacemos usos de las siguientes librerías

```
import simpy
import random
```

- Tenemos también una clase la cual representa el proceso, así como la función la cual simula el comportamiento del proceso, y por supuesto la función la cual nos permite usar nuestra estrategia en este caso shortest job first

```
class Proceso:
    def __init__(self, nombre, tiempo_ejecucion):
        self.nombre = nombre
        self.tiempo_ejecucion = tiempo_ejecucion

def proceso(env, nombre, tiempo_ejecucion):
    print(f'[{env.now}] {nombre} llega al sistema.')
    yield env.timeout(tiempo_ejecucion)
    print(f'[{env.now}] {nombre} ha terminado su ejecución.')
```



```
def scheduler(env):  
    procesos = []  
  
    # Generar procesos  
    for i in range(5):  
        tiempo_ejecucion = random.randint(1, 5)  
        nombre_proceso = f'Proceso {i + 1} (Tiempo: {tiempo_ejecucion})'  
        procesos.append(Proceso(nombre_proceso, tiempo_ejecucion))  
  
    # Ordenar procesos por tiempo de ejecución (SJF)  
    procesos.sort(key=lambda p: p.tiempo_ejecucion)  
  
    print("\n--- Procesos programados (orden SJF) ---")  
    for p in procesos:  
        print(f'{p.nombre} - Tiempo de ejecución: {p.tiempo_ejecucion}')  
  
    print("\n--- Comenzando la ejecución ---")  
  
    for p in procesos:  
        env.process(proceso(env, p.nombre, p.tiempo_ejecucion))  
        yield env.timeout(1) # Esperar un tiempo entre llegadas  
  
# Configuración del entorno  
env = simpy.Environment()  
env.process(scheduler(env))  
env.run()
```

- ✚ Por último, hacemos la ejecución, se aprecia el como los procesos van llegando al sistema e igual en que momento acaban variando tanto el orden de los procesos, con lo cual observamos que no necesariamente es el primero que llega forzosamente debe de acabar primero.



```
C:\Users\Israel Cuevas\Documents\Arquitectura de computadoras> python sjf.py

--- Procesos programados (orden SJF) ---
Proceso 1 (Tiempo: 2) - Tiempo de ejecución: 2
Proceso 5 (Tiempo: 2) - Tiempo de ejecución: 2
Proceso 2 (Tiempo: 3) - Tiempo de ejecución: 3
Proceso 3 (Tiempo: 3) - Tiempo de ejecución: 3
Proceso 4 (Tiempo: 4) - Tiempo de ejecución: 4

--- Comenzando la ejecución ---
[0] Proceso 1 (Tiempo: 2) llega al sistema.
[1] Proceso 5 (Tiempo: 2) llega al sistema.
[2] Proceso 1 (Tiempo: 2) ha terminado su ejecución.
[2] Proceso 2 (Tiempo: 3) llega al sistema.
[3] Proceso 5 (Tiempo: 2) ha terminado su ejecución.
[3] Proceso 3 (Tiempo: 3) llega al sistema.
[4] Proceso 4 (Tiempo: 4) llega al sistema.
[5] Proceso 2 (Tiempo: 3) ha terminado su ejecución.
[6] Proceso 3 (Tiempo: 3) ha terminado su ejecución.
[8] Proceso 4 (Tiempo: 4) ha terminado su ejecución.

C:\Users\Israel Cuevas\Documents\Arquitectura de computadoras>
```

#### 4.4 RR

Por último, tenemos al round Robin de igual forma utilizamos SimPy.

- Hacemos uso de las librerías

```
import simpy
import random
```

- Tenemos a nuestra clase y a la función que nos permite simular, esta función genera un conjunto de proceso con tiempos de ejecución aleatorios, para después imprimir en lista de todos los procesos simulados, utilizando un bucle para procesar cada uno de los procesos en la cola, haciendo uso del algoritmo Round Robin.



```
class Proceso:
    def __init__(self, nombre, tiempo_ejecucion):
        self.nombre = nombre
        self.tiempo_ejecucion = tiempo_ejecucion

def proceso(env, nombre, tiempo_ejecucion, quantum):
    while tiempo_ejecucion > 0:
        tiempo_actual = min(quantum, tiempo_ejecucion)
        print(f'[{env.now}] {nombre} se ejecuta por {tiempo_actual} unidades de tiempo.')
        yield env.timeout(tiempo_actual)
        tiempo_ejecucion -= tiempo_actual

        if tiempo_ejecucion > 0:
            print(f'[{env.now}] {nombre} se interrumpe y vuelve a la cola.')
        else:
            print(f'[{env.now}] {nombre} ha terminado su ejecución.')

def scheduler(env, quantum):
    procesos = []

    # Generar procesos
    for i in range(5):
        tiempo_ejecucion = random.randint(5, 15)
        nombre_proceso = f'Proceso {i + 1} (Tiempo: {tiempo_ejecucion})'
        procesos.append(Proceso(nombre_proceso, tiempo_ejecucion))

    print("\n--- Procesos programados ---")
    for p in procesos:
        print(f'{p.nombre} - Tiempo de ejecución: {p.tiempo_ejecucion}')

    print("\n--- Comenzando la ejecución ---")

    while procesos:
        p = procesos.pop(0) # Obtener el primer proceso en la cola
        yield env.process(proceso(env, p.nombre, p.tiempo_ejecucion, quantum))

# Configuración del entorno
quantum = 3 # Tiempo de quantum
env = simpy.Environment()
env.process(scheduler(env, quantum))
env.run()
```

 Ejecución



```
C:\Windows\System32\cmd.exe

C:\Users\Israel Cuevas\Documents\Arquitectura de computadoras>python round_robin.py

--- Procesos programados ---
Proceso 1 (Tiempo: 15) - Tiempo de ejecución: 15
Proceso 2 (Tiempo: 14) - Tiempo de ejecución: 14
Proceso 3 (Tiempo: 12) - Tiempo de ejecución: 12
Proceso 4 (Tiempo: 5) - Tiempo de ejecución: 5
Proceso 5 (Tiempo: 7) - Tiempo de ejecución: 7

--- Comenzando la ejecución ---
[0] Proceso 1 (Tiempo: 15) se ejecuta por 3 unidades de tiempo.
[3] Proceso 1 (Tiempo: 15) se interrumpe y vuelve a la cola.
[3] Proceso 1 (Tiempo: 15) se ejecuta por 3 unidades de tiempo.
[6] Proceso 1 (Tiempo: 15) se interrumpe y vuelve a la cola.
[6] Proceso 1 (Tiempo: 15) se ejecuta por 3 unidades de tiempo.
[9] Proceso 1 (Tiempo: 15) se interrumpe y vuelve a la cola.
[9] Proceso 1 (Tiempo: 15) se ejecuta por 3 unidades de tiempo.
[12] Proceso 1 (Tiempo: 15) se interrumpe y vuelve a la cola.
[12] Proceso 1 (Tiempo: 15) se ejecuta por 3 unidades de tiempo.
[15] Proceso 1 (Tiempo: 15) ha terminado su ejecución.
[15] Proceso 2 (Tiempo: 14) se ejecuta por 3 unidades de tiempo.
[18] Proceso 2 (Tiempo: 14) se interrumpe y vuelve a la cola.
[18] Proceso 2 (Tiempo: 14) se ejecuta por 3 unidades de tiempo.
[21] Proceso 2 (Tiempo: 14) se interrumpe y vuelve a la cola.
[21] Proceso 2 (Tiempo: 14) se ejecuta por 3 unidades de tiempo.
[24] Proceso 2 (Tiempo: 14) se interrumpe y vuelve a la cola.
[24] Proceso 2 (Tiempo: 14) se ejecuta por 3 unidades de tiempo.
[27] Proceso 2 (Tiempo: 14) se interrumpe y vuelve a la cola.
[27] Proceso 2 (Tiempo: 14) se ejecuta por 2 unidades de tiempo.
[29] Proceso 2 (Tiempo: 14) ha terminado su ejecución.
[29] Proceso 3 (Tiempo: 12) se ejecuta por 3 unidades de tiempo.
[32] Proceso 3 (Tiempo: 12) se interrumpe y vuelve a la cola.
[32] Proceso 3 (Tiempo: 12) se ejecuta por 3 unidades de tiempo.
[35] Proceso 3 (Tiempo: 12) se interrumpe y vuelve a la cola.
[35] Proceso 3 (Tiempo: 12) se ejecuta por 3 unidades de tiempo.
[38] Proceso 3 (Tiempo: 12) se interrumpe y vuelve a la cola.
[38] Proceso 3 (Tiempo: 12) se ejecuta por 3 unidades de tiempo.
[41] Proceso 3 (Tiempo: 12) ha terminado su ejecución.
[41] Proceso 4 (Tiempo: 5) se ejecuta por 3 unidades de tiempo.
[44] Proceso 4 (Tiempo: 5) se interrumpe y vuelve a la cola.
[44] Proceso 4 (Tiempo: 5) se ejecuta por 2 unidades de tiempo.
[46] Proceso 4 (Tiempo: 5) ha terminado su ejecución.
[46] Proceso 5 (Tiempo: 7) se ejecuta por 3 unidades de tiempo.
[49] Proceso 5 (Tiempo: 7) se interrumpe y vuelve a la cola.
[49] Proceso 5 (Tiempo: 7) se ejecuta por 3 unidades de tiempo.
[52] Proceso 5 (Tiempo: 7) se interrumpe y vuelve a la cola.
[52] Proceso 5 (Tiempo: 7) se ejecuta por 1 unidades de tiempo.
[53] Proceso 5 (Tiempo: 7) ha terminado su ejecución.

C:\Users\Israel Cuevas\Documents\Arquitectura de computadoras>
```





## Conclusiones.

En esta práctica de simulación de los distintos procesos de la CPU, hemos podido observar el cómo se comportan de acuerdo con el tipo de estrategia que se implementa, desde el planteamiento del proceso para ver que estrategia es convenientemente mejor para realizarlo hasta el cómo afecta al orden el que se ejecuta dicho proceso.

Estos se ven afectado directamente al tiempo de espera, al de repuesta y al rendimiento de los mismos.

## Referencias

del computador para que sean asignados convenientemente a los procesos., L. A. M. I. del N. del S. O. es I. L. P. C. P. es un P. V. en D. se E. U. A. o. U. H. del S. O. E. N. D. E. E. de A. L. R. del H. (n.d.). *Administración de Procesos (versión beta)*. Uchile.Cl. Retrieved October 28, 2024, from <https://users.dcc.uchile.cl/~lmateu/CC41B/Apuntes/admin-procesos.pdf>

(N.d.). Edu.Uy. Retrieved October 28, 2024, from <https://www.fing.edu.uy/inco/cursos/sistoper/recursosTeoricos/6-SO-Teo-Planificacion.pdf>

