SISTEMAS OPERATIVOS

Proyecto 2: Planificación por retroalimentación y por lotería

Ing. Gunnar Eyal Wolf Iszaevich
Semestre 2023-2

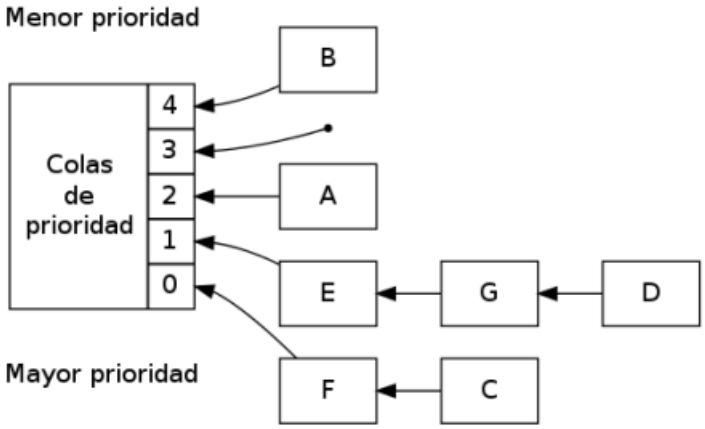
Grupo 6 Sistemas Operativos

Universidad Nacional Autónoma de México | Facultad de Ingeniería

LUIS FERNANDO MORALES ZILLI
No. Cuenta 421085620

Retroalimentación multinivel

Durante este proyecto revisamos los conceptos aprendidos en el tema de administración de procesos. Comenzaremos analizando el proceso de retroalimentación multinivel, este nos permite crear colas de prioridad para que los procesos que se encuentren en las colas de mayor prioridad puedan acceder a tiempo de procesador antes que los que se encuentran en baja prioridad.



A esto podemos añadirle parámetros distintos como cada cuanto es que se van a degradar los procesos después de pasado su quantum o de cuantos ticks consistirá este quantum.

Para implementar este esquema lo que realicé fue crear cuatro colas y estas las uní en una lista, fueron cuatro colas las que se creo para esta implementación tomando en cuenta que la cola cero es la de mayor prioridad y la numero tres la de menor prioridad.

En esta implementación se utilizó un Quantum de dos ticks, pero este se puede variar cambiando su valor en la sección de parámetros. Para generar los parámetros me ayudé de gran medida de la función *randint* esta me permitió generar un numero de procesos aleatorio en cada iteración, también con esta definí la duración de los procesos y su llegada según los ticks.

```
Procesos:
{'id': 'A', 'inicio': 0, 'duracion': 10, 'procesado': 0}
{'id': 'B', 'inicio': 13, 'duracion': 14, 'procesado': 0}
{'id': 'E', 'inicio': 17, 'duracion': 11, 'procesado': 0}
{'id': 'C', 'inicio': 26, 'duracion': 12, 'procesado': 0}
{'id': 'D', 'inicio': 39, 'duracion': 11, 'procesado': 0}
{'id': 'G', 'inicio': 41, 'duracion': 15, 'procesado': 0}
{'id': 'F', 'inicio': 50, 'duracion': 15, 'procesado': 0}
```

La lógica detrás de la implementación de la retroalimentación será descrita a continuación. Para comenzar añadimos el proceso ‘A’ el cual siempre será el que entra primero en el sistema, los demás procesos entraran según su numero de inicio, para ordenarlos se utilizó la función *sorted* tomando en cuenta el parámetro anterior.

El esquema comienza con un ciclo while el cual tiene la condición de ejecutarse siempre y cuando haya elementos en las colas o algún proceso en la lista de procesos que aún no han comenzado a procesarse.

Inmediatamente después del comienzo puede observarse una de las medidas que se tomó en este algoritmo para evitar la inanición. La bandera que se muestra tiene uso casi al final del ciclo, esta bandera será modificada si algún proceso es procesado durante ese tick, si no sucede esto entonces significa que el sistema no tiene ningún proceso para procesar y este continúa aumentando ticks hasta que se llega a un tick donde comience otro proceso. Este nuevo proceso se añadirá a la cola cero.

Este esquema suele tener el problema de la inanición por el tema de los números de inicio aleatorio, es posible que haya momentos donde no se esté procesando nada y solo se pase ticks.

Cuando si hay ticks el sistema entra en un ciclo interno el cual buscará elementos en todas las colas comenzando con las de mayor prioridad. Al encontrar un elemento este será ejecutado el numero de ticks que indique el quantum.

En este ciclo de procesamiento podremos encontrar restricciones, por ejemplo, si el proceso llega a su máxima duración a mitad del quantum entonces este mostrará un mensaje de que el proceso ha terminado y se terminará prematuramente el proceso. Con esto el proceso será eliminado de la cola donde se encontraba por medio de la función *pop*.

Si se termina el quantum del proceso y este no ha terminado su ejecución entonces se buscará si algún proceso comienza su ejecución en ese tick, para esto dentro de un ciclo while se añadirán los procesos a la cola y se eliminarán de la lista de procesos.

Esto se hace utilizando *pop* en la lista de procesos y *append* en la cola cero. Esto es posible debido a que el proceso mas cercano a iniciar siempre estará en la posición cero de la cola de procesos. Si no se encuentra simplemente se romperá el while.

```
while len(procesos)!=0:
    if tick == procesos[0]['inicio']:
        print('Llego el proceso '+procesos[0]['id'])
        colas[0].append(procesos.pop(0))
    else:
        break
```

Lotería

Considero que este esquema es relativamente mas sencillo que el anterior, debido a que su lógica es un poco mas simple al utilizar el azar para elegir los procesos a ser ejecutados asignándoles una probabilidad a cada uno de ellos. Lo mas complicado quizá fue la función que permite el reacomodo de las probabilidades (en este caso llamada boletos) para que, si después de un tiempo la diferencia de probabilidades es muy grande, esta pueda achicarse haciendo más justa la elección.

Para esta implementación me basé en la misma lógica para crear los procesos por medio de la biblioteca random, sin embargo, en este caso se utilizó para mas cosas, una de ellas fue la elección del proceso a procesar por medio de la función *choices.* Esta función nos permite elegir un elemento por medio de una lista de probabilidades asignadas a cada elemento, esto queda excelente para seguir la lógica propuesta por este esquema.

Una vez elegido el proceso se realizan todas las convenciones para que se vea reflejada su ejecución en el sistema y en la lógica interna, esto nos ayuda a que no se pierda el progreso y al llegar a su limite de duración este pueda ser eliminada de la lista de procesos como también de la lista de probabilidades con la que seguirá jugando el sistema.

En este caso no existe una medida contra la inanición porque después de terminado su quantum el sistema elige inmediatamente otro proceso para ser ejecutado. Esto elimina la posibilidad de inanición y no rompe las reglas del esquema.

Para implementar la funcionalidad de cooperación entre los procesos se realizó una función de reacomodo, donde cada diez ticks se busca el proceso con la mayor cantidad de boletos y el que tiene la menor cantidad. Una vez encontrados se toma la diferencia entre estos, se toma su cuarta parte y se abona al proceso con menos boletos. Esta cantidad será restada del proceso con la mayor cantidad de boletos simulando que se le cedieron al otro proceso.

```
def reacomodo(ids,boletos):
    maxi = max(boletos)
    mini = min(boletos)
    diff = (maxi-mini)/4
    print('\nReacomodando boletos de %s:%f a %s:%f' % (
        ids[boletos.index(maxi)],max(boletos),
        ids[boletos.index(mini)],min(boletos)
    ))
    n_min=boletos.index(mini)
    n_max=boletos.index(maxi)
    boletos[n_min]+=diff
    boletos[n_max]-=diff
    print('Resultado de %s:%f a %s:%f' % (
        ids[n_max],boletos[n_max],
        ids[n_min],boletos[n_min]
    ))
```

Conclusión

Creo que la planificación que mas me agradó fue la de retroalimentación multinivel debido a que mi implementación tomó un poco mas de complejidad al realizar la lógica. Además de que permite que sea mas consistente el procesamiento de un elemento, mientras que en la lotería el hecho de depender de la probabilidad para que se ejecute un proceso podría hacer que un solo proceso tarde mas en completar su ejecución.

Si bien la lotería es más justa al ajustar la probabilidad de ejecución y al elegir de forma aleatoria, también tiene el problema de que sea mas tardado de poder terminar un solo proceso. Por otro lado, la inanición en esta implementación queda prácticamente eliminada por la misma naturaleza de la lógica.

Creo que se podría mejorar el problema de la tardanza para sacar un solo proceso al modificar el quantum de la misma ejecución.

La retroalimentación multinivel tiende a generar inanición, pero esto se debe a la aparición de los procesos que se encuentran alejados y que la duración no tiene suficiente tiempo para alcanzarlos.

Creo que este funcionaría de mejor manera para procesos más cortos, ya que llegaría a funcionar como un esquema FIFO. Siendo así que entre más rápido lleguen más rápido serán procesados y terminados. Creo que este método es el que tiene mas robustez en su construcción y es el que mejor cumple su tarea de administrar la completitud de los procesos.