Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Ingeniería en computación

Seminario de solución de problemas de sistemas operativos

NRC 164138

Sección D07

Profesor Javier Rosales Martínez

Diego Armando Sánchez Rubio 217570609

Reporte Práctica 7.

Antecedentes.

Siguiendo con las prácticas de administración de memoria, ahora es el turno de los algoritmos o métodos restantes, que consisten en Peor Ajuste y Siguiente Ajuste, se analizarán ambos y se explicará su funcionamiento y su lógica.

Metodología.

Al igual que la práctica pasada se tiene el mismo arreglo de Memoria Base, ya qué nos será útil para cargar lo archivos a esta memoria.

```
MEMORIA_BASE = [1000, 400, 1800, 700, 900, 1200, 1500]
```

El primer algoritmo que se revisará es peor ajuste, la función recibe los archivos a cargar y la memoria disponible. Se inicializa una lista vacía para guardar las asignaciones.

Por cada archivo, se recorre la memoria en busca del bloque más grande que sea suficiente para contenerlo. Si encuentra uno, guarda su índice y tamaño.

Si se encuentra ese bloque adecuado, el archivo se asigna ahí, se agrega a la lista de asignaciones con su información, y se reduce el tamaño del bloque. Si no hay bloque disponible, se indica que no fue asignado.

Finalmente, devuelve todas las asignaciones realizadas.

```
def peor_ajuste(archivos, memoria):
    asignaciones = []
    for archivo, tam in archivos:
        peor_idx = -1
        peor_bloque = -1
        for i, bloque in enumerate(memoria):
            if bloque >= tam and bloque > peor_bloque:
                peor_bloque = bloque
                peor_idx = i
        if peor_idx != -1:
            asignaciones.append((archivo, tam, peor_idx, MEMORIA_BASE[peor_idx]))
            memoria[peor_idx] -= tam
        else:
            asignaciones.append((archivo, tam, -1, None))
    return asignaciones
```

Ahora revisando el algoritmo Siguiente Ajuste, la función recibe los archivos y la memoria disponible. Se inicializa una lista vacía de asignaciones y una variable inicio que guarda desde qué bloque empezar a buscar.

Por cada archivo, se recorre la memoria desde el último punto donde se hizo una asignación, dando vuelta circular si es necesario. Se busca el primer bloque desde ahí que tenga el tamaño suficiente.

Cuando se encuentra un bloque adecuado, se asigna el archivo, se reduce su tamaño en la memoria y se actualiza inicio para que la siguiente búsqueda

comience desde ese bloque. Si no se encuentra ningún bloque, se marca como no asignado.

Al final, se devuelve la lista con todas las asignaciones.

```python
def siguiente_ajuste(archivos, memoria):
    asignaciones = []
    inicio = 0
    n = len(memoria)
    for archivo, tam in archivos:
        encontrado = False
        for i in range(n):
            idx = (inicio + i) % n
            if memoria[idx] >= tam:
                asignaciones.append((archivo, tam, idx, MEMORIA_BASE[idx]))
                memoria[idx] -= tam
                inicio = idx  # Actualizamos el punto de inicio para el siguiente archivo
                encontrado = True
                break
        if not encontrado:
            asignaciones.append((archivo, tam, -1, None))
    return asignaciones
```

La función que se revisará ahora es asignar, que se encarga de coordinar el proceso de asignación de archivos a bloques de memoria. Lo primero que hace es verificar si hay archivos cargados; si no hay, muestra una advertencia y detiene la ejecución.

Después limpia la tabla donde se muestran las asignaciones anteriores y reinicia la memoria disponible haciendo una copia de la memoria base.

Luego, dependiendo del algoritmo que elija el usuario, se aplica Peor Ajuste o Siguiente Ajuste. Según el caso, se llama a la función correspondiente y se almacenan las asignaciones resultantes.

Una vez obtenidas las asignaciones, se recorren para mostrarlas en la interfaz, indicando el nombre del archivo, su tamaño, el número del bloque asignado (o “No asignado” si no fue posible) y el tamaño original de ese bloque.

Al final, se actualiza visualmente la representación gráfica de los bloques con la función dibujar_bloques.

```python
def asignar(self):
    if not self.archivos:
        messagebox.showwarning("Advertencia", "Primero carga un archivo.")
        return

    self.tree.delete(*self.tree.get_children())
    self.memoria_actual = MEMORIA_BASE.copy()

    if self.algoritmo_var.get() == "peor":
        self.asignaciones = peor_ajuste(self.archivos, self.memoria_actual)
    else:
        self.asignaciones = siguiente_ajuste(self.archivos, self.memoria_actual)

    for archivo, tam, idx, bloque_tam in self.asignaciones:
        bloque_str = str(idx + 1) if idx != -1 else "No asignado"
        bloque_tam_str = f"{bloque_tam} KB" if bloque_tam else "-"
        self.tree.insert("", "end", values=(archivo, f"{tam} KB", bloque_str, bloque_tam_str))

    self.dibujar_bloques()
```

Conclusiones.

Esta práctica al igual que la anterior me resultó bastante fácil de realizar, también fue muy útil para comprender cómo es que funcionan con exactitud los dos métodos o algoritmos que faltaban y ver la diferencia en ambos, en general fue una práctica muy útil y didáctica de hacer.

Referencias.

Eslabon. Estación para Laboratorios Online. (n.d.).

https://labvirtual.webs.upv.es/Best_Fit.htm

Spasojevic, A. (2025, March 27). *¿Qué es la asignación de primer ajuste?* phoenixNAP IT

Glossary. https://phoenixnap.es/glosario/asignaci%C3%B3n-de-primer-ajuste

tkinter — Python interface to Tcl/Tk. (n.d.). Python Documentation.

https://docs.python.org/3/library/tkinter.html