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 6.

Antecedentes.

En esta práctica se revisará el funcionamiento de la memoria, se realizará una práctica dónde demostrarán dos técnicas de administración de memoria, en este caso consisten en Primer Ajuste y Mejor Ajuste, se hará una implementación en Python junto con un archivo que contendrá procesos.

Metodología.

En este caso me dedicaré solamente a explicar los algoritmos que se utilizaron para la administración de memoria y cómo es que funcionan, además de cómo es que se hace la asignación y cómo carga el archivo.

Lo primero es que mediante un arreglo se establece una memoria base con segmentos de memoria con tamaño establecido, esto para que al momento de procesar los archivos tengan en dónde guardarse desde un principio.

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

Ahora revisando el primero de los algoritmos que es Primer Ajuste, está establecido de la siguiente manera, lo que recibe la función son los archivos que se van a cargar y la memoria actual que hay en el sistema, es decir que es una copia de la memoria base, después se inicializa una lista vacía de las asignaciones que se harán, para después con un ciclo for recorrer todo el archivo e ir sacando, después con otro ciclo for busca en cada bloque de memoria uno que se adecue al tamaño del archivo que se quiere cargar, una vez se encuentra lo agrega a la lista de asignaciones y se decrementa el tamaño de memoria de ese bloque en especifico, en caso de no encontrar un bloque del tamaño, también se agrega a la lista de asignaciones, pero indicando que no cabe en ningún lado con un None.

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

Ahora revisando el segundo de los algoritmos que es Mejor Ajuste, la función recibe los archivos a cargar y la memoria disponible actual. Se inicializa una lista vacía para guardar las asignaciones.

Por cada archivo, se busca el bloque **más pequeño posible** que aún así tenga el tamaño suficiente para guardarlo. Para esto, se recorren todos los bloques y se va guardando el que mejor se ajuste (es decir, el más chico que sirva).

Si se encuentra un bloque adecuado, se asigna ahí el archivo, se guarda la asignación (nombre, tamaño, índice y tamaño original del bloque), y se reduce el tamaño del bloque. Si no hay ningún bloque disponible, se guarda también en la lista de asignaciones con -1 y None, indicando que no cabe.

Finalmente, la función devuelve todas las asignaciones.

```
def mejor_ajuste(archivos, memoria):
    asignaciones = []
    for archivo, tam in archivos:
        mejor_idx = -1
        mejor_bloque = None
        for i, bloque in enumerate(memoria):
            if bloque >= tam:
                if mejor_bloque is None or bloque < mejor_bloque:
                    mejor_bloque = bloque
                    mejor_idx = i
        if mejor_idx != -1:
            asignaciones.append((archivo, tam, mejor_idx, MEMORIA_BASE[mejor_idx]))
            memoria[mejor_idx] -= tam
        else:
            asignaciones.append((archivo, tam, -1, None))
    return asignaciones
```

Ahora revisando la función asignar, lo primero que hace es verificar si ya se han cargado archivos; si no, muestra una advertencia y se detiene.

Luego, limpia cualquier resultado anterior de la interfaz (self.tree) y reinicia la memoria disponible usando una copia de MEMORIA_BASE.

Después revisa qué algoritmo seleccionó el usuario. Si eligió "primer", usa la función de Primer Ajuste, y si no, aplica Mejor Ajuste (aunque aquí podrías expandirlo a más algoritmos).

Una vez hechas las asignaciones, se recorre cada resultado para mostrarlo en la tabla. Si el archivo fue asignado a un bloque, se indica el número del bloque y su tamaño original; si no, se marca como “No asignado”.

Finalmente, se llama a self.dibujar_bloques() para actualizar visualmente los bloques en la interfaz.

```
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() == "primer":
        self.asignaciones = primer_ajuste(self.archivos, self.memoria_actual)
    else:
        self.asignaciones = mejor_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 fue un buen ejercicio para ver cómo es que funcionan estos métodos de administración de memoria y ver la diferencia entre unos y otros, la verdad no considero que haya sido una práctica difícil, es bastante fácil implementarlo una vez conoces el cómo funciona el método.

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