MEMORIA PL1 EEDD

Héctor Barrios Lujan 02552800H

Roberto Riofrío Ortega 51026360D

INDICE

Detalles y justificación de la implementación:

TAD´S

Los TAD´s que hemos implementado son:

1. TAD Procesos
2. TAD Pila
3. TAD Cola de prioridades
4. TAD Lista Enlazada

Definición de las operaciones del TAD (Nombre, argumentos y retorno).

TAD Procesos:

- **void mostrar()**: Muestra en pantalla los datos del proceso (PID, usuario, estado, prioridad, tipo, etc.).
  Argumentos: Ninguno
  Retorno: Ninguno

- **bool getVacio()**: Retorna si el proceso está vacío o no.
  Argumentos: Ninguno
  Retorno: `bool` (verdadero si el proceso está vacío, falso en caso contrario)

- **void setVacio(bool v)**: Establece si el proceso está vacío.
  Argumentos: v (booleano que indica si el proceso está vacío)
  Retorno: Ninguno

- **void crearProceso(int pid)**: Crea un nuevo proceso con un identificador único de proceso (PID).
  Argumentos: `pid` (entero que representa el PID del nuevo proceso)
  Retorno: Ninguno

- **bool getTipo()**: Retorna el tipo de proceso (`false` para proceso normal y `true` para proceso en tiempo real).
  Argumentos: Ninguno

Retorno: `bool` (indicador del tipo de proceso)

- **bool getEstado()**: Retorna el estado del proceso (`true` si está en ejecución, `false` si está parado).
  Argumentos: Ninguno
  Retorno: `bool` (estado del proceso)
- **void setEstado(bool e)**: Establece el estado del proceso (`true` para ejecución, `false` para parado).
  Argumentos: e (booleano que representa el estado)
  Retorno: Ninguno

- **int getPrioridad()**: Retorna la prioridad del proceso.
  Argumentos: Ninguno
  Retorno: `int` (prioridad del proceso)

- **void setPrioridad(int p)**: Asigna una nueva prioridad al proceso.
  Argumentos: p (entero que representa la nueva prioridad)
  Retorno: Ninguno

- **int getPID()**: Retorna el identificador único del proceso.
  Argumentos: Ninguno
  Retorno: `int` (PID del proceso)

- **char* getUsuario()**: Retorna el nombre del usuario que lanzó el proceso.
  Argumentos: Ninguno
  Retorno: `char*` (nombre del usuario)

TAD Pila:

- **void insertar(Proceso v)**: Inserta un proceso en la parte superior de la pila.
  Argumentos: v (de tipo `Proceso`, representa el proceso a insertar)
  Retorno: Ninguno

- **void extraer()**: Elimina el proceso en la parte superior de la pila.

Argumentos: Ninguno
Retorno: Ninguno

- **Proceso cima()**: Retorna el proceso en la parte superior de la pila sin eliminarlo.
  Argumentos: Ninguno
  Retorno: `Proceso` (proceso en la cima de la pila)

- **void mostrar()**: Muestra en pantalla todos los procesos almacenados en la pila, desde el último insertado hasta el primero.
  Argumentos: Ninguno
  Retorno: Ninguno

- **int getLongitud()**: Retorna el número de procesos en la pila.
  Argumentos: Ninguno
  Retorno: `int` (cantidad de procesos en la pila)

TAD Cola de prioridades:

- **void insertar(Proceso v)**: Inserta un proceso en la cola de acuerdo con su prioridad.
  Argumentos: v (de tipo `Proceso`, representa el proceso a insertar en la cola)
  Retorno: Ninguno

- **void eliminar()**: Elimina el proceso con mayor prioridad de la cola (el primer proceso en la cola).
  Argumentos: Ninguno
  Retorno: Ninguno

- **void mostrar()**: Muestra en pantalla todos los procesos almacenados en la cola, en orden de prioridad.
  Argumentos: Ninguno
  Retorno: Ninguno

- **Proceso verPrimero()**: Retorna el proceso con mayor prioridad sin eliminarlo.
  Argumentos: Ninguno
  Retorno: `Proceso` (proceso con mayor prioridad en la cola)

- **int getLongitud()**: Retorna el número de procesos en la cola.
  Argumentos: Ninguno
  Retorno: `int` (cantidad de procesos en la cola)

TAD Lista enlazada simple:

- **void insertarIZQ(Proceso v)**: Inserta un proceso al inicio (izquierda) de la lista.
  Argumentos: v (de tipo `Proceso`, representa el proceso a insertar)
  Retorno: Ninguno

- **void insertarDER(Proceso v)**: Inserta un proceso al final (derecha) de la lista.
  Argumentos: v (de tipo `Proceso`, representa el proceso a insertar)
  Retorno: Ninguno

- **void eliminarPrimero()**: Elimina el primer proceso de la lista.
  Argumentos: Ninguno
  Retorno: Ninguno

- **void eliminarUltimo()**: Elimina el último proceso de la lista.
  Argumentos: Ninguno
  Retorno: Ninguno
- **Proceso verPrimero()**: Retorna el primer proceso de la lista sin eliminarlo.
  Argumentos: Ninguno

  Retorno: `Proceso` (el primer proceso de la lista)

- **Proceso verUltimo()**: Retorna el último proceso de la lista sin eliminarlo.
  Argumentos: Ninguno
  Retorno: `Proceso` (el último proceso de la lista)

- **void mostrar()**: Muestra en pantalla todos los procesos en la lista.
  Argumentos: Ninguno
  Retorno: Ninguno

- **Proceso menorPrioridad()**: Retorna el proceso con menor prioridad (mayor valor numérico) en la lista.
  Argumentos: Ninguno
  Retorno: `Proceso` (proceso con menor prioridad en la lista)

- **Proceso mayorPrioridad()**: Retorna el proceso con mayor prioridad (menor valor numérico) en la lista.
  Argumentos: Ninguno
  Retorno: `Proceso` (proceso con mayor prioridad en la lista)

- **void buscarProcesosUsuario(string user)**: Muestra todos los procesos que pertenecen al usuario especificado.
  Argumentos: `user` (de tipo `string`, nombre del usuario)
  Retorno: Ninguno

- **Proceso borrarProcesosPID(int pid)**: Elimina y retorna el proceso con el identificador (PID) especificado.
  Argumentos: `pid` (entero que representa el ID del proceso)
  Retorno: `Proceso` (el proceso eliminado de la lista)

- *Proceso* *buscarProcesoPID(int pid): Busca y retorna un puntero al proceso con el identificador (PID) especificado.
  Argumentos: `pid` (entero que representa el PID del proceso)
  Retorno: `Proceso*` (puntero al proceso encontrado, o vacío si no existe)

Dificultades encontradas

Durante el trabajo por lo general no nos hemos encontrado muchas dificultades, exceptuando algunas respecto a la parte final de la práctica con la cola de prioridades y la lista enlazada y algunos de sus métodos. La principal con el funcionamiento de la cola de prioridades a la hora de administrar los procesos y como se dividen en las diferentes colas ya que aparte de tener que gestionar que procesos va primero dependiendo de la prioridad, también hay que ordenarlos en distintas colas dependiendo del tipo de proceso y los procesos en tiempo real y los normales tampoco tienen la misma forma de asignarles una prioridad así que también supone un problema a la hora de asignarle esa prioridad y luego ordenarlos.

Luego en las listas, los métodos de búsqueda de cierto proceso en las listas nos han generado cierta dificultad a la hora de tener que recorrer la lista entera para poder encontrar procesos que estuvieran en medio, porque al no ser un elemento del principio o del final tenías que ir recorriendo la lista comparando elemento a elemento y avanzando al siguiente nodo.

Métodos más importantes

Los métodos más importantes dirían que son los que insertan y eliminan procesos de los diferentes TAD´s , después los métodos que buscan en las listas procesos y los mostrar de todos los TAD´S.

TAD Pila:

Insertar: Se crea un nuevo nodo, se le asigna a ese nodo un proceso y un nodo que va a ser el último de la pila, se cambia el nombre del nodo de nuevo a ultimo y se suma una a la longitud.

Extraer: Se crea un nodo nuevo si no hay ningún nodo último se devuelve NULL, si hay un nodo último, el nodo creado pasa a señalar a ultimo y ultimo pasa a ser el siguiente de nodo creado, longitud se disminuye en 1 y elimina el nodo creado.

Mostrar: Es una función recursiva que crea un nodo que se le asigna el puntero ultimo y muestra mediante la función mostrar de proceso todas las características del proceso después asigna el puntero al siguiente nodo y así hasta que el siguiente nodo sea nulo.

TAD Cola de prioridades:

Insertar: Primero, se declara un puntero nuevo de tipo pnodoCola . Luego, se crea un nuevo nodo nuevo que almacena el proceso v. Este nodo contendrá el proceso que se va a insertar.

A continuación, se comprueba si la cola ya tiene elementos (si el primero no es nulo). Si la cola está vacía, se manejará más adelante. Si la cola no está vacía, se compara la prioridad del nuevo proceso con la prioridad del primer proceso en la cola. Si el nuevo proceso tiene una prioridad más alta (menor valor numérico), se coloca al frente de la cola. El nuevo nodo apunta al nodo que estaba primero, y primero se actualiza para que apunte al nuevo nodo.

Si el nuevo proceso no tiene la prioridad más alta, se inicializa un puntero aux que apunta al primer nodo. Luego, se recorre la cola con un bucle, avanzando hasta que se encuentra un nodo cuya prioridad sea menor que la del nuevo nodo, o hasta el final de la cola.

Si se encuentra una posición adecuada, es decir, si el siguiente nodo de aux no es nulo, el nuevo nodo se inserta entre aux y aux->siguiente. Si se alcanza el final de la cola, significa que el nuevo proceso tiene la menor prioridad y debe ir al final de la cola, estableciendo que el último nodo actual apunta al nuevo nodo y actualizando ultimo para que apunte al nuevo nodo.

Finalmente, se incrementa el contador longitud, que mantiene un registro del número total de nodos (procesos) en la cola.

Eliminar: Se declara un puntero nodo que apunta al primer nodo de la cola. Se comprueba si la cola está vacía. Si el puntero nodo es nulo, el método termina sin hacer nada. Si hay un nodo, se actualiza el puntero primero para que apunte al siguiente nodo en la cola. Se elimina el nodo. Se verifica si primero se ha vuelto nulo (la cola está vacía) y, si es así, se establece ultimo como nulo. Se

decrementa el contador longitud, que mantiene un registro del número total de nodos en la cola.

Mostrar: Es una función que crea un nodo que se le asigna el puntero primero y muestra mediante la función mostrar de proceso todas las características del proceso después asigna el puntero al siguiente nodo y así hasta que el siguiente nodo sea nulo.

TAD Listas

InsertarIZQ: El método insertarIZQ se encarga de insertar un nuevo proceso al inicio de la lista; primero, se crea un nuevo nodo que almacena el proceso y se comprueba si la lista está vacía; si es así, se establece primero para que apunte al nuevo nodo, de lo contrario, el nuevo nodo se inserta al inicio estableciendo que el nuevo nodo apunta al nodo que era primero y luego actualizando primero para que apunte al nuevo nodo; finalmente, se incrementa el contador longitud para reflejar el nuevo tamaño de la lista.

InsertarDER: Por otro lado, el método insertarDER se encarga de insertar un nuevo proceso al final de la lista; se crea un nuevo nodo que almacena el proceso y se verifica si la longitud de la lista es menor que 1, en cuyo caso se establece primero para que apunte al nuevo nodo; si la lista no está vacía, se inicializa un puntero para recorrer la lista hasta llegar al último nodo y una vez allí, se establece que el siguiente nodo del último nodo apunte al nuevo nodo; al igual que en el método anterior, se incrementa el contador longitud para reflejar el nuevo tamaño de la lista.

EliminarPrimero: El método eliminarPrimero se encarga de eliminar el primer nodo de la lista; primero, se verifica si la longitud de la lista es menor que 1, en cuyo caso no se realiza ninguna acción; si hay nodos en la lista, se utiliza un puntero aux para guardar la referencia al nodo que se va a eliminar, luego se actualiza primero para que apunte al siguiente nodo en la lista y se decrementa el contador longitud; se establece aux->siguiente a NULL para desasociar el nodo de la lista antes de liberarlo de la memoria usando delete aux.

EliminarUltimo: El método eliminarUltimo se encarga de eliminar el último nodo de la lista; se inicia verificando si la longitud de la lista es menor que 1, en cuyo caso no se realiza ninguna acción; si hay nodos en la lista, se inicializa un

puntero aux para recorrer la lista; si el nodo apuntado por aux es el único nodo (es decir, si aux->siguiente es nulo), se establece primero a NULL y se libera la memoria del nodo; si hay más de un nodo, se recorre la lista hasta llegar al penúltimo nodo y se guarda el puntero al último nodo en p, luego se establece que el penúltimo nodo ya no tiene siguiente al asignar aux->siguiente a NULL, liberando finalmente la memoria ocupada por el nodo apuntado por p y decrementando el contador longitud para reflejar el cambio en el tamaño de la lista.

Mostrar: Es una función que crea un nodo que se le asigna el puntero primero y muestra mediante la función mostrar de proceso todas las características del proceso después asigna el puntero al siguiente nodo y así hasta que el siguiente nodo sea nulo.

BuscarProcesosPID: El método buscarProcesoPID se encarga de buscar un proceso en la lista basado en su PID, en primer lugar, se verifica si la longitud de la lista es menor que 1, lo que indica que la lista está vacía, en cuyo caso el método retorna nullptr para indicar que no se encontró ningún proceso. si hay nodos en la lista, se inicializa un puntero aux que apunta al primer nodo de la lista y se inicia un bucle que recorre todos los nodos hasta que aux sea nulo. Dentro del bucle, se compara el PID del proceso almacenado en el nodo actual con el PID proporcionado como argumento; si hay una coincidencia, se llama al método mostrar() del proceso para mostrar su información y se retorna un puntero que apunta al proceso encontrado. Si se completa el recorrido de la lista sin encontrar el PID correspondiente, el método retorna nullptr, indicando que no se encontró ningún proceso con ese identificador.

Explicación del comportamiento del programa

El funcionamiento del programa para gestionar el servidor con tarjetas gráficas (GPUs) para computación en paralelo en el departamento de Ciencias de la Computación se basa en una simulación que permite la administración eficiente de los procesos. Primero, se generará un conjunto de procesos aleatorios que se almacenarán en una Pila LIFO. Cada proceso contendrá información crítica, incluyendo un PID único, el nombre del usuario que lanzó el proceso, el estado

del proceso (ejecución o parado), su prioridad y un indicador que clasifica el proceso como normal o en tiempo real.

Una vez que se han generado todos los procesos y se encuentran en la pila, el programa comenzará a desapilar estos procesos uno a uno para clasificarlos y encolarlos en las cuatro colas de prioridades disponibles, asignadas a las GPUs. Las GPUs 0 y 1 gestionarán los procesos normales, mientras que las GPUs 2 y 3 se encargarán de los procesos en tiempo real. Antes de que un proceso sea encolado, se establecerá su estado como "parado" y se calculará su prioridad. Para los procesos normales, se asignará una prioridad en un rango entre -19 y +19, sumando 120 a este valor antes de encolarlo para asegurarse de que se utilice un número positivo que permita ordenar adecuadamente la cola. Los procesos en tiempo real recibirán una prioridad entre 0 y 99.

Cuando se desencola un proceso, se evaluarán las colas de las GPUs disponibles y el proceso se colocará en la cola que contenga menos procesos en ese momento. Si hay un empate en el número de procesos entre las GPUs, el proceso se asignará a la GPU 0. Al desencolar un proceso, también se cambiará su estado a "ejecución" y se añadirá a una lista correspondiente que facilita la monitorización del sistema. Las listas estarán implementadas dinámicamente y se usarán para almacenar los procesos que se están ejecutando en las GPUs 0 y 1 (normal) o en las GPUs 2 y 3 (tiempo real).

El administrador del sistema podrá realizar varias operaciones a través de estas listas, incluyendo listar todos los procesos en ejecución, buscar un proceso específico mediante su PID, finalizar un proceso si es necesario, y cambiar la prioridad de un proceso ya en ejecución. Este sistema garantizará que los recursos de las GPUs se compartan de manera eficiente y ordenada entre los diferentes usuarios del departamento, maximizando así el rendimiento en las tareas de inteligencia artificial.

Para los métodos de gestor:

La **Opción A** se utilizará para generar 12 procesos aleatorios cada vez que se seleccione, con un límite total de 48 procesos que pueden ser almacenados en la pila. Esto asegura que haya un flujo constante de procesos disponibles para su gestión.

La **Opción B** permite a los usuarios visualizar todos los procesos actualmente almacenados en la pila, proporcionando una visión general de los procesos generados y sus atributos.

La **Opción C** permite eliminar todos los procesos almacenados en la pila que fueron generados a través de la Opción A, facilitando la limpieza del sistema y la preparación para nuevas operaciones.

La **Opción D** extrae los procesos de la pila y los clasifica en una de las cuatro colas de prioridad disponibles, dependiendo del tipo de proceso y asegurando que se almacenen de manera ordenada según su prioridad.

La **Opción E** permite mostrar los procesos que se encuentran en las colas de las GPUs 0 y 1, que están dedicadas a los procesos normales, ofreciendo una forma de monitorear estos procesos.

La **Opción F** proporciona información similar pero para las colas de las GPUs 2 y 3, que se ocupan de los procesos en tiempo real.

La **Opción G** da la posibilidad de borrar todos los procesos que están almacenados en las cuatro colas, liberando recursos y limpiando el sistema.

La **Opción H** extrae los procesos de las colas y los transfiere a sus correspondientes listas (Lista Normal o Lista Tiempo Real), cambiando su estado de "parado" a "ejecución", lo que indica que ahora están en uso.

La **Opción I** permite mostrar los procesos en la Lista Normal en un formato tabular, donde se incluirán columnas para PID, nombre de usuario, tipo de proceso, estado del proceso y prioridad del proceso, facilitando así una clara visualización de la información.

La **Opción J** ofrece la misma funcionalidad que la Opción I pero para la Lista Tiempo Real, permitiendo una fácil comparación y análisis de los procesos en tiempo real.

La **Opción K** busca en las listas y muestra dos procesos: el de menor prioridad en la lista normal y el de mayor prioridad en la lista de tiempo real, ayudando a identificar rápidamente los procesos más y menos críticos.

La **Opción L** solicitará al usuario un nombre de usuario y buscará todos los procesos lanzados por dicho usuario en las listas, mostrando esta información en pantalla.

La **Opción M** pedirá un número de PID para buscar el proceso correspondiente en las listas; si se encuentra, el proceso se eliminará de la lista, su estado cambiará a "parado" y se devolverá a la pila inicial.

La **Opción N** solicitará un número de PID para buscar el proceso en las listas; si existe, se le pedirá al usuario que introduzca una nueva prioridad para actualizarla.

Por último, la **Opción O** reiniciará el programa a su estado inicial, eliminando todos los procesos y configuraciones actuales, permitiendo comenzar de nuevo con un entorno limpio.

Bibliografía

Stackoverflow.com (Para consultar dudas específicas sobre errores o ayuda en algún método)