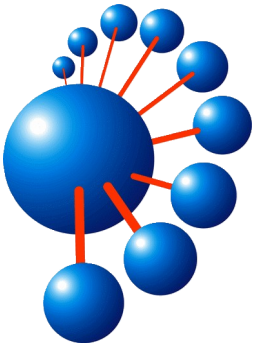

Proyecto 3 Sistemas Operativos.

Manejo de Memoria.

Profesora: Cecilia Hernández.
Alumnos: Diego Varas.
Jeremías Torres.

Junio 12, 2017.

Introducción

En éste proyecto se pide implementar el manejador de memoria malloc de C. Para esto se debe solicitar memoria al sistema operativo usando la llamada sbrk() y la implementación considera una lista doblemente enlazada.

Además de nuestra función malloc, se implementará la función free para poder liberar la memoria.

Nuestro malloc está implementado con el criterio “First Fit” el cual consiste en recorrer la lista desde el head buscando un bloque libre.

Otra cosa a considerar en la implementación es que se debe hacer Spliting y Merge. El primero consiste en dividir un bloque libre cuando la memoria a reservar es menor el. El segundo consiste en “fusionar” dos bloques contiguos que estén libres.

Algoritmos de Solución

En esta función reservaremos la memoria que necesitamos para nuestro malloc. Por línea de comando entrará a esta función el valor dado y creará un bloque con su respectivo tamaño ingresado, a continuación cada vez que hagamos malloc creará un nodo con el tamaño indicado y se irá enlazando con las demás reservas de memoria.

```
void extendHeap(int n){
        void* head = sbrk(0);
        void* tmp = sbrk(n + BLOCK_SIZE);
        void* ptr = head + BLOCK_SIZE;
        memblock block = (memblock)head;
        block->size = n;
        block->prev = NULL;
        block->next = NULL;
        block->mip = ptr;
        block->magic = 0x77777777;
        block->free = 1;
        addBlock(block);
}
```

En la función mimalloc() lo primero que debe hacer es buscar en find_block un bloque en el que insertar. Si esta función retorna un bloque vacío entonces llamamos a split_block, la cual en caso de que encuentre un bloque vacío y el tamaño del malloc sea menor al tamaño del bloque encontrado, creará un nodo contiguo al nodo ocupado y apuntará al nodo vacío el cual se le restará la memoria utilizada de él. Si no, llamará a extendHeap ya antes mencionada.

```
void* mimalloc(int size){
        memblock block = find_block2(size);
        if(block == NULL){
                extendHeap(size);
                block = find_block2(size);

        }
        block->free = 0;
        block->magic = 0x12345678;
        split_block(block, size);
        return block->mip;
}
```

En la función mifree liberamos los malloc utilizados pasando el puntero por parámetro. Cuando encuentre el bloque correspondiente al puntero dado cambiará sus valores de free y magic además de llamar a la función fusion() la cual hace el trabajo de merge que es juntar dos bloques libres contiguos.

```
void mifree(void* ptr){
	memblock actual = lista;
	while(actual != NULL){
		if(actual->mip == ptr){
			actual->free = 1;
			actual->magic = 0x77777777;
			if(actual->next != NULL)
				fusion(actual);
			if(actual->prev != NULL && actual->prev->free == 1)
				fusion(actual->prev);
			if(actual->next == NULL)
				brk(actual->mip + actual->size);
			return;
		}
		actual = actual->next;
	}
}
```

Conclusión

Logramos implementar el malloc y free con las llamadas a sistema sbrk y brk, utilizando el método “First Fit”.

Además comprendimos el manejo de espacio libre con su direccionamiento utilizando spliting y merge los cuales son utilizados por malloc y free. Todo esto nos ayudó a comprender el funcionamiento del Heap y nuevas formas de reservar memoria como las implementadas.