/\*

\* ssoo/xalloc.98/xalloc.c

\*

\* CONTENIDO INICIAL:

\* Codigo correspondiente a la Seccion 8.7 del libro:

\* "The C Programing Language", de B. Kernigham y D. Ritchie.

\*

\* En este fichero se incluiran las rutinas pedidas

\*

\*/

#include <unistd.h>

#include <stdio.h>

#include "xalloc.h"

/\*

\* Definicion de la cabecera para huecos y bloques.

\* La union con un campo de tipo Align fuerza que el tama~no

\* de la cabecera sea multiplo del tama~no de este tipo.

\*/

typedef long Align; /\* for alignment to long boundary \*/

union header { /\* block header: \*/

struct {

union header \*ptr; /\* next block if on free list \*/

size\_t size; /\* size of this block \*/

} s;

Align x; /\* force alignment of blocks \*/

};

typedef union header Header;

/\*

\* La lista de huecos esta ordenada por direcciones y es circular.

\* base es el "falso" hueco de tama~no cero que asegura que la lista

\* nunca esta vacia

\*/

static Header base; /\* empty list to get started \*/

/\*

\* freep apuntara al hueco situado en la lista antes del hueco

\* por el que comenzara la busqueda.

\* Necesario para implementar la estrategia next-fit

\*/

static Header \*freep = NULL; /\* start of the free list \*/

#define NALLOC 1024

/\*

\* morecore: ask system for more memory

\*

\* Esta funcion se llama desde xmalloc cuando no hay espacio.

\* Aumenta el tama~no de la zona de datos como minimo en NALLOC\*sizeof(Header)

\* y a~nade esta nueva zona a la lista de huecos usando xfree.

\*

\*/

void imprimirHuecos(){

printf("- Imprimiendo espacios libres:\n");

Header\* p;

printf("Freep : %x\n", freep);

char c;

//scanf("%c",&c);

for(p=freep->s.ptr;;p=p->s.ptr){

printf("Address: %x - Size:%d\n",p,p->s.size);

if(p==freep) break;

}

printf("\n");

}

void copiarDatos(void\* ptr1, void\* ptr2, int nbytes){

//printf("Destino: %x - Fuente: %x\n", ptr1, ptr2);

char\* dest = (char\*)ptr1;

char\* src = (char\*)ptr2;

int i;

for(i = 0;i< nbytes;i++){

dest[i] = src[i];

}

}

static Header \*morecore(size\_t nu)

{

printf("Llamo morecore\n");

char \*cp;

Header \*up;

if (nu < NALLOC)

nu = NALLOC;

cp= sbrk(nu \* sizeof(Align)); // 8k papu :v

if (cp == (char \*) -1)

return NULL;

up = (Header \*) cp;

up ->s.size = nu;

printf("- Imprimiendo espacio libre nuevo:\n");

printf("Address: %x, size: %d\n\n",up, up->s.size);

xfree((void \*)(up+1));

return freep;

}

/\* xmalloc: general-purpose storage allocator \*/

void \*xmalloc (size\_t nbytes)

{

int start = 0; //flag

Header \*p, \*prevp, \*best = NULL;

size\_t nunits;

nunits = (nbytes+sizeof(Align)-1)/sizeof(Align) + (sizeof(Header)-1)/sizeof(Align) + 1; // Los ultimos elementos de la suma son para incluir el header

printf("Numero de units que se necesita: %d\n",nunits);

if(nunits == 0) printf("Pediste cero :( \n");

if (( prevp = freep) == NULL ) {

base.s.ptr = freep = prevp = & base;

base.s.size = 0;

start = 1;

}

printf("Antes de malloc\n");

imprimirHuecos();

if (start == 1){

if ((p = morecore(nunits)) == NULL)

return NULL; /\* none left \*/

}

for (p= prevp->s.ptr;; prevp = p, p = p->s.ptr) {

printf("Hueco actual: %x, size: %d\n", p, p->s.size);

if (p->s.size >= nunits) {

if (best == NULL){

best = p;

} else {

if (best->s.size > p->s.size)

best = p;

}

}

if (p > p->s.ptr || p == p->s.ptr) break;

}

printf("best: %x\n\n", best);

if (best != NULL){

if (best->s.size == nunits || best->s.size == (nunits + 1)){ // Si es exacto o queda espacio de uno no genera hueco

printf("best tiene espacio exacto\n");

prevp->s.ptr = best->s.ptr;

return (void\*) (best+1);

} else {

printf("best es mas grande que el espacio requerido\n");

printf("best: %x - size: %d - siguiente: %x\n", best, best->s.size, best->s.ptr);

printf("Units del nuevo hueco: %d\n", best->s.size - nunits);

best->s.size -= nunits;

Align\* p2 = (Align\*)best; // Esta parte es importante, como p es un puntero de Head y size esta en unidades de Align, casteamos el puntero como Align para poder sumarle el size.

p2 += best->s.size;

p = (Header\*)p2; // El puntero en la posicion correcta se vuelve a castear como Header

p->s.size = nunits;

printf("Nuevo espacio asignado, address:%x - size: %d\n", p, p->s.size);

freep = prevp;

printf("\nDespues de malloc\n");

imprimirHuecos();

printf("Direccion de inicio del nuevo bloque (sin contar la cabecera):%x\n", p+1);

return (void \*)(p + 1);

}

} else {

// si se ha dado toda la vuelta y no se encontro ningun hueco

// disponible para ese tamanio

return NULL;

}

}

/\* xfree: put block ap in the free list \*/

void xfree(void \*ap)

{

Header \*bp, \*p;

bp = (Header \*)ap - 1;

for (p= freep; !(bp > p && bp < p->s.ptr); p = p->s.ptr)

if (p >= p->s.ptr && (bp > p || bp < p->s.ptr))

break; /\* freed block at start or end of arena \*/

/\* Comprueba compactacion con hueco posterior \*/

if ((Header\*)((Align\*)bp + bp->s.size) == p->s.ptr) { // La misma logica de incremento del puntero de la linea 96

bp->s.size += p->s.ptr->s.size;

bp->s.ptr = p->s.ptr->s.ptr;

} else

bp->s.ptr = p->s.ptr;

/\* Comprueba compactacion con hueco anterior \*/

if ((Header\*)((Align\*)p + p->s.size) == bp) { // La misma logica de incremento del puntero de la linea 96

p->s.size += bp->s.size;

p->s.ptr = bp->s.ptr;

} else

p->s.ptr = bp;

// Como la estrategia es best-fit no se asigna a freep el hueco actual

// sino que se empezara la busqueda desde el primer hueco

//freep = p; /\* estrategia next-fit \*/

}

void \*xrealloc(void \* ptr, size\_t size)

{

printf("Antes de realloc\n");

imprimirHuecos();

Header \*bp, \*p;

bp = (Header\*)ptr - 1;

size\_t nunits = (size+sizeof(Align)-1)/sizeof(Align) + (sizeof(Header)-1)/sizeof(Align) + 1;

printf("Me has pedido %d unidades, %d bytes\n",nunits, size);

printf("Bp, dir: %x, size: %d\n",bp, bp->s.size);

for (p= freep; !(bp > p && bp < p->s.ptr); p = p->s.ptr){

if (p >= p->s.ptr && (bp > p || bp < p->s.ptr))

break;

}

printf("p, dir: %x, size: %d\n\n",p, p->s.size);

//printf("Pase el for\n");

if(nunits < bp->s.size){ // Caso 1: Piden menos de lo que hay

printf("Caso 1: reducir tamaño\n");

if(bp->s.size - nunits > 1 || (Header\*)((Align\*)bp + bp->s.size) == p->s.ptr){

Header \*generado = (Header\*)( (Align\*)bp + nunits);

generado->s.size = bp->s.size - nunits;

if((Header\*)((Align\*)bp + bp->s.size) == p->s.ptr){ // compactar a la derecha

generado->s.size += p->s.ptr->s.size;

generado->s.ptr = p->s.ptr->s.ptr;

}

else{

generado->s.ptr = p->s.ptr;

}

p->s.ptr = generado;

bp->s.size = nunits;

freep = p;

}

} else if(nunits > bp->s.size){ // piden mas de lo que hay

int required\_units = nunits - bp->s.size;

int hay\_posterior\_adyacente = ((Header\*)((Align\*)bp + bp->s.size) == p->s.ptr);

int hay\_anterior\_adyacente = ((Header\*)((Align\*)p + p->s.size) == bp);

Header\* next\_hole = p->s.ptr;

int prev\_size = next\_hole->s.size;

int units\_adyacentes\_disponibles = 0;

if(hay\_anterior\_adyacente) units\_adyacentes\_disponibles+=p->s.size;

if(hay\_posterior\_adyacente) units\_adyacentes\_disponibles+=next\_hole->s.size;

if(hay\_posterior\_adyacente && next\_hole->s.size >= required\_units){ // Caso 2: Piden mas y hay un hueco posterior adyacente con espacio suficiente

printf("Caso 2: aumentar tamaño y existe hueco parte posterior del bloque, necesito %d\n",required\_units);

if (next\_hole->s.size == required\_units || next\_hole->s.size == (required\_units + 1)){ // Si es exacto o queda espacio de uno no genera hueco

p->s.ptr = next\_hole->s.ptr;

}

else {

next\_hole->s.size -= required\_units;

Align\* p2 = (Align\*)next\_hole;

p2 += required\_units;

next\_hole = (Header\*)p2;

//p->s.size = required\_units;

}

bp->s.size += required\_units;

freep = p;

}

else if(hay\_anterior\_adyacente && units\_adyacentes\_disponibles >= required\_units){ // Caso 3: Piden mas y hay un hueco anterior adyacente con espacio suficiente

//printf("Base -> Direccion: %x - Size: %d\n", bp, bp->s.size);

//printf("Hueco Anterior -> Direccion: %x - Size: %d\n", p, p->s.size);

printf("Caso 3: aumentar tamaño y existe hueco parte anterior del bloque, necesito %d\n",required\_units);

Header \*siguiente, \*prev, \*new\_bp;

int bp\_size = bp->s.size, i;

if(hay\_posterior\_adyacente){

//printf("H\n");

required\_units -= next\_hole->s.size;

siguiente = next\_hole->s.ptr;

p->s.ptr = next\_hole->s.ptr;

bp->s.size += next\_hole->s.size;

}

else siguiente = next\_hole;

for(prev = p->s.ptr; prev!=p; prev = prev->s.ptr);

//printf("Prev founded\n");

if (p->s.size == required\_units || p->s.size == (required\_units + 1)){ // Si es exacto o queda espacio de uno no genera hueco

prev->s.ptr = siguiente;

new\_bp = p;

//printf("Nu nuevo\n");

}

else {

//printf("Required: %d - PSize: %d - Adyacentes: %d\n", required\_units, p->s.size,units\_adyacentes\_disponibles);

p->s.size -= required\_units;

new\_bp = (Header\*)((Align\*)p + p->s.size);

new\_bp->s.size = bp->s.size + required\_units;

//printf("Nuevo\n");

}

copiarDatos(new\_bp + 1, bp + 1, sizeof(Align)\*(bp\_size - (sizeof(Header)-1)/sizeof(Align) - 1));

//printf("Copio\n");

bp = new\_bp;

freep = prev;

}

else{ // Caso 4: no hay memoria por ningun lado, hay q buscar o pedir (gracias malloc :) )

printf("Caso 4: No hay espacio\n");

printf("BP es: %x\n", bp);

Header \*new\_bp;

printf("Pide malloc\n");

new\_bp = (Header\*)((Header\*)xmalloc(size) - 1);

printf("Sale malloc\n");

if(new\_bp == (NULL )) return NULL;

printf("Copia\n");

copiarDatos(new\_bp + 1, bp+1, sizeof(Align)\*(bp->s.size - (sizeof(Header)-1)/sizeof(Align) - 1));

printf("Limpia\n");

xfree(bp +1);

//freep = bp;

printf("Asigna\n");

bp = new\_bp;

printf("Despues de realloc\n");

imprimirHuecos();

printf("New bp: %x - size: %x\n",bp,bp->s.size);

char c;

scanf("%c",&c);

}

} else {

printf("Me pidieron el mismo tamanio :p\n");

}

printf("Despues de realloc\n");

imprimirHuecos();

printf("Bp, dir: %x, size: %d\n",bp, bp->s.size);

return (void\*)(bp + 1);

}

//para compilar gcc main.c xalloc.c -o main -w