/\* CPSC/ECE 3220 simple memory block allocation program

\*

\* the functions work on a single array of memory blocks, each of which

\* can either be free or allocated and each of which has a status byte

\* and payload size byte at each end (i.e., header and trailer fields)

\*

\* block structure

\*

\* +--------+--------+------------------------+--------+--------+

\* | status | size | area to allocate | size | status |

\* +--------+--------+------------------------+--------+--------+

\* |<--- header ---->|<---- payload size ---->|<--- trailer --->|

\* |<----------------------- block size ----------------------->|

\*

\* status byte: 0 => free, 1 => allocated

\* size byte: payload size is limited to 255

\*

\*

\* block structure annotated with pointer values

\*

\* block pointer when considering this block for allocation

\* | => \*(block\_pointer) == status

\* |

\* | block pointer + 1 => \*(block\_pointer + 1) == size

\* | |

\* | | block pointer + 2 == pointer returned to user

\* v v v

\* +--------+--------+------------------------+--------+--------+

\* | status | size | area to allocate | size | status |

\* +--------+--------+------------------------+--------+--------+

\* ^ ^ ^

\* block pointer + size + 2 | |

\* | |

\* block pointer + size + 3 |

\* |

\* block pointer + size + 4

\* == start of next block

\*

\* the allocate function is first fit and traverses blocks until a free

\* block of adequate payload size is found; the top of the free block

\* is split off for allocation if the remaining space is large enough

\* to support a free block of MIN\_PAYLOAD\_SIZE in size along with new

\* header and trailer, otherwise the complete free block is allocated

\*

\* the release function merely changes the status of an allocated block

\* back to free; no coalescing is done in the current implementation

\*/

#include <stdio.h>

#define FREE 0

#define ALLOCATED 1

#define BYTE\_COUNT 256

#define MIN\_PAYLOAD\_SIZE 8

#define MIN\_BLOCK\_SIZE 12

unsigned char \_\_attribute\_\_ ((aligned (65536))) area[BYTE\_COUNT];

void simple\_init(){

/\* top status \*/ area[0] = FREE;

/\* top size \*/ area[1] = BYTE\_COUNT - 4;

/\* bottom size \*/ area[BYTE\_COUNT - 2] = BYTE\_COUNT - 4;

/\* bottom status \*/ area[BYTE\_COUNT - 1] = FREE;

}

void print\_blocks(){

unsigned char \*block\_ptr = area;

printf( "\nblock allocation list\n" );

while( block\_ptr < ( area + BYTE\_COUNT ) ){

printf( "--block at %p\n", block\_ptr );

printf( " top status is %d\n", \*block\_ptr );

printf( " top size is %d\n", \*(block\_ptr+1) );

printf( " bottom size is %d\n", \*(block\_ptr + \*(block\_ptr+1) + 2) );

printf( " bottom status is %d\n", \*(block\_ptr + \*(block\_ptr+1) + 3) );

block\_ptr += \*(block\_ptr+1) + 4;

}

}

unsigned char \*simple\_allocate( unsigned int req\_size ){

unsigned char \*block\_ptr;

unsigned int original\_payload\_size, remaining\_payload\_size;

/\* immediately reject requests that are too large \*/

if( req\_size > BYTE\_COUNT - 4 ) return NULL;

/\* start search \*/

block\_ptr = area;

while( block\_ptr < ( area + BYTE\_COUNT ) ){

if( ( \*block\_ptr == FREE ) && ( \*(block\_ptr+1) >= req\_size ) ){

if( ( \*(block\_ptr+1) - req\_size ) < MIN\_BLOCK\_SIZE ){

\*block\_ptr = ALLOCATED;

\*(block\_ptr + ( \*(block\_ptr+1) + 3 )) = ALLOCATED;

return ( block\_ptr + 2 );

}else{

original\_payload\_size = \*(block\_ptr+1);

remaining\_payload\_size = \*(block\_ptr+1) - req\_size - 4;

\*block\_ptr = ALLOCATED;

\*(block\_ptr+1) = req\_size;

\*(block\_ptr + req\_size + 2) = req\_size;

\*(block\_ptr + req\_size + 3) = ALLOCATED;

\*(block\_ptr + req\_size + 4) = FREE;

\*(block\_ptr + req\_size + 5) = remaining\_payload\_size;

\*(block\_ptr + original\_payload\_size + 2) = remaining\_payload\_size;

return ( block\_ptr + 2 );

}

}

block\_ptr += \*(block\_ptr+1) + 4;

}

return NULL;

}

void simple\_release( unsigned char \*usr\_ptr ){

\*(usr\_ptr-2) = FREE;

\*(usr\_ptr + \*(usr\_ptr-1) + 1) = FREE;

}

/\* test driver \*/

int main(){

unsigned char \*p[8];

simple\_init();

print\_blocks();

p[0] = simple\_allocate( 252 ); /\* uses all 256 bytes \*/

print\_blocks();

simple\_release( p[0] );

print\_blocks();

p[1] = simple\_allocate( 12 ); /\* uses 16 bytes \*/

p[2] = simple\_allocate( 12 ); /\* uses 16 bytes \*/

p[3] = simple\_allocate( 12 ); /\* uses 16 bytes \*/

print\_blocks();

simple\_release( p[2] );

print\_blocks();

simple\_release( p[1] ); /\* does not coalesce in this version \*/

print\_blocks();

return 0;

}

Extension 1 - coding with macros to hide details (macros are expanded in-line)

/\* macros for header and trailer fields based on \*/

/\* block\_ptr variable and payload size field \*/

#define HEADER\_SIZE 2

#define CONTROL\_FIELDS\_SIZE 4

#define USER\_PTR (block\_ptr+HEADER\_SIZE)

#define TOP\_STATUS (\*(block\_ptr))

#define TOP\_SIZE (\*(block\_ptr+1))

#define PAYLOAD\_SIZE (TOP\_SIZE)

#define BLOCK\_SIZE (TOP\_SIZE+CONTROL\_FIELDS\_SIZE)

#define BOTTOM\_SIZE (\*(block\_ptr+PAYLOAD\_SIZE+2))

#define BOTTOM\_STATUS (\*(block\_ptr+PAYLOAD\_SIZE+3))

#define TOP\_STATUS\_OF\_NEXT\_BLOCK (\*(block\_ptr+PAYLOAD\_SIZE+4))

#define TOP\_SIZE\_OF\_NEXT\_BLOCK (\*(block\_ptr+PAYLOAD\_SIZE+5))

Functions become:

void print\_blocks(){

unsigned char \*block\_ptr = area;

printf( "\nblock allocation list\n" );

while( block\_ptr < ( area + BYTE\_COUNT ) ){

printf( "--block at %p\n", block\_ptr );

printf( " top status is %d\n", TOP\_STATUS );

printf( " top size is %d\n", TOP\_SIZE );

printf( " bottom size is %d\n", BOTTOM\_SIZE );

printf( " bottom status is %d\n", BOTTOM\_STATUS );

CPSC/ECE 3220 – In-Class Assignment 14 Name: \_\_\_\_\_Rajat Sethi\_\_\_\_\_\_\_\_\_\_

Consider the basic program rather than the extensions.

1. If the payload size is 32 bytes, what is the size of the allocated block?

36 Bytes

2. Why is the payload size limited to 255 bytes in this program?

Size is only one byte, with an unsigned range of 0-255

3. When a free block is found, is the top part allocated (i.e., lower memory addresses), or is the bottom part allocated (i.e., higher memory addresses)?

The top part.

4. Does the user receive the address of the allocated block or the address of the payload area within the allocated block?

Address of the payload.

5. Does the user return the address of the allocated block or the address of the payload area within the allocated block? What does this mean for marking the returned block as free?

Address of the payload. Mark the payload as free and then subtract two bytes as well.

block\_ptr += BLOCK\_SIZE;

}

}

unsigned char \*simple\_allocate( unsigned int req\_size ){

unsigned char \*block\_ptr;

unsigned int remaining\_payload\_size;

/\* immediately reject requests that are too large \*/

if( req\_size > BYTE\_COUNT - CONTROL\_FIELDS\_SIZE ) return NULL;

/\* start search \*/

block\_ptr = area;

while( block\_ptr < ( area + BYTE\_COUNT ) ){

if( ( \*block\_ptr == FREE ) && ( PAYLOAD\_SIZE >= req\_size ) ){

if( ( PAYLOAD\_SIZE - req\_size ) < ( MIN\_BLOCK\_SIZE ) ){

TOP\_STATUS = ALLOCATED;

BOTTOM\_STATUS = ALLOCATED;

return ( USER\_PTR );

}else{

remaining\_payload\_size = PAYLOAD\_SIZE - req\_size - CONTROL\_FIELDS\_SIZE;

/\* change bottom size before changing top size \*/

BOTTOM\_SIZE = remaining\_payload\_size;

TOP\_STATUS = ALLOCATED;

TOP\_SIZE = req\_size;

BOTTOM\_SIZE = req\_size;

BOTTOM\_STATUS = ALLOCATED;

TOP\_STATUS\_OF\_NEXT\_BLOCK = FREE;

TOP\_SIZE\_OF\_NEXT\_BLOCK = remaining\_payload\_size;

return ( USER\_PTR );

}

}

block\_ptr += BLOCK\_SIZE;

}

return NULL;

}

void simple\_release( unsigned char \*user\_ptr ){

unsigned char \*block\_ptr = user\_ptr - HEADER\_SIZE;

TOP\_STATUS = FREE;

BOTTOM\_STATUS = FREE;

}

**Extension 2** – add a free list to speed the search

**Extension 3** – coalesce free blocks and add zero-size allocated blocks at the top and bottom to remove need to bounds-check when coalescing

#define BOTTOM\_STATUS\_OF\_PRIOR\_BLOCK (\*(block\_ptr-1))

#define BOTTOM\_SIZE\_OF\_PRIOR\_BLOCK (\*(block\_ptr-2))

void simple\_init(){

/\* permanently-allocated blocks at top and bottom \*/

/\* of original area to simplify code to coalesce \*/

/\* top status \*/ area[0] = ALLOCATED;

/\* top size \*/ area[1] = 0;

/\* bottom status \*/ area[2] = 0;

/\* bottom size \*/ area[3] = ALLOCATED;

/\* top status \*/ area[4] = FREE;

/\* top size \*/ area[5] = BYTE\_COUNT - 12;

/\* bottom size \*/ area[BYTE\_COUNT - 6] = BYTE\_COUNT - 12;

/\* bottom status \*/ area[BYTE\_COUNT - 5] = FREE;

/\* top status \*/ area[BYTE\_COUNT - 4] = ALLOCATED;

/\* top size \*/ area[BYTE\_COUNT - 3] = 0;

/\* bottom size \*/ area[BYTE\_COUNT - 2] = 0;

/\* bottom status \*/ area[BYTE\_COUNT - 1] = ALLOCATED;

}