CS3026 Operating Systems – Assignment 1
Custom memory allocation
Julia Wiktoria Zięba (ID: 51768280)

REPORT

Solution contents

1. mymemory.h – header file containing function prototypes and declarations
2. mymemory.c – C file containing function implementations
3. shell.c – C file, wrapper calling functionalities from mymemory.c file
4. makefile – a makefile for compiling and building the solution

Implemented functionalities:

Following functionalities have been implemented:

1. CGS D3-D1:
    a. makefile
    b. `void initialize();`
    c. `void printsegmenttable();`
    d. `void printmemory();`
    e. created `shell.c`;
2. CGS C3-C1:
    a. `void * mymalloc ( size_t size );`
    b. `Segment_t * findFree ( Segment_t * list, size_t size );`
    c. `void insertAfter (Segment_t * oldSegment, Segment_t * newSegmen t);`
    d. `void splitSegment ( Segment_t * oldSegment, size_t size);`
    a segment and size of memory to be allocated are passed to the function; segment is split into allocated segment of size = size and a free segment of size = oldSegment->size – size;
    e. approperiately updated `shell.c`;
3. CGS B3-B1:
    a. `void myfree ( void * ptr );`
    b. `Segment_t * findSegment ( Segment_t * list, void * ptr );`
    c. `void wipeMemory(void * ptr, size_t size);`
    wipes the deallocated memory from the memory table;
    d. approperiately updated `shell.c`;
4. CGS A5-A1:
    a. merging `segmenttable` (allocated segments are pushed to the front, empty segments are merged into one and pushed to the end);
    b. updating addressed to which segment descriptors point to;
    c. approperiately updated `shell.c`;

How to run the program

1. Open the terminal in the file directory.
2. Run “make” command in order to compile and build the solution.
3. Execute the generated “myprog.exe” file by running “. /myprog” command.

Results of execution

1. Empty memory table and segment table with one free segment are initialised and printed;
2. Memory is allocated three time to the `ptr1`, `ptr2`, and `ptr3`
    a. The state of the memory and segment tables is shown after each allocation;
    b. An error message is displayed if memory cannot be allocated;
3. `ptr1` is freed
    a. The state of the memory and segment tables is shown after freeing the pointer;
4. Memory is defragmented
    a. Segment table is rearranged, so that allocated segments are pushed to the front, and deallocated segments are merged and pushed to the end;
    b. Segment start addresses are updated;
    c. Memory contents are not moved (see Not implemented);
    d. The state of the memory and segment tables is shown after the defragmentation;

Additionally, the `shell.c` contains helper `printf()` statements that walk the user through the execution of the program and indicate what information is being printed to the terminal.

Not implemented

Unfortunately, I was not able to implement the following functionalities for CGS A5-A1:

1. Returning an array of pointers from `mydefrag ( void ** ptrlist );`
2. Moving memory contents in the memory table;