Operating Systems

Homework Report

Muhammad Youshay (07103)

Assignment 02 – Stack and Heap Memory Management

November 19, 2023

Table of Contents

1 Code Snippets –

Structures Used, Global Variables, and Initialization

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#pragma pack(1)

#define MEMSIZE 500 //Total Memory Size
#define MAXFRAMES 5 //Maximum number of frames
#define MINSTACKFRAMESIZE 10 //Minimum size of a frame
#define MAXSTACKFRAMESIZE 80 //Maximum size of a frame

struct framestatus {
   int number;
  char name[8];
   int functionaddress;
   int frameaddress;
   char used;
};

struct freelist {
   int start;
   int size;
   struct freelist * next;
};

struct allocated { //This structure was provided to us by Sir Tariq Kamal to
    maintain Heap allocation
  char name[8];
  int start;
  int size;
};


void createframe(char[], int);
void deleteframe();
void createint(char[], int);
void createdouble(char[], double);
void createchar(char[], char);
void createbuffer(char[], int);
void deletebuffer(char[]);
void showmemory();


char memory[MEMSIZE]; //This is the memory array of size 500
```

```
int offset = 10; //used to calculate the offset of the frame
int top = -1; //used to keep track of the top frame
int stackSize = 105; //Initial Stack Size - 105 bytes are used by the frame
    status list
int maxStackSize = 300; //Maximum Stack Size
int maxHeapSize = 300; //Maximum Heap Size
int heapSize = 0; //Initial Heap Size
int topFrame = 395; //Top Frame Address


struct framestatus* frameStatusList = (struct framestatus*) &memory[395];
    //This is the frame status list which is stored at the top of the stack
struct freelist* head = NULL; //This is the head of the free list
struct allocated alloc[18]; //This is the allocated list which is used to
    maintain the heap allocation, maximum 18 buffers can be allocated
bool offsetFlag = false; //This flag is used to check if the offset has been
    calculated or not

void initialize() {
  head = (struct freelist*)malloc(sizeof(struct freelist)); //Allocating
      memory to the head of the free list
  head->start = 0; //Setting the start address of the free list
  head->size = maxHeapSize; //Setting the size of the free list
  head->next = NULL; //Setting the next pointer of the free list to NULL
}
```

Function to create a frame on the stack –

```
void createframe(char functionName[], int functionAddress) {
  printf("Stack Size Before creating the Stack-Frame --> %d\n",stackSize);
  //Checking if the maximum number of frames have been reached or not -
      maximum 5 frames can be created
  if (top >= MAXFRAMES - 1) {
    printf("Cannot create another frame, maximum number of frames have been
        reached\n");
    return;
  }
  //Checking if the function already exists or not - comparing the function
      name with the names of the functions in the frame status list
  for (int i = 0; i <= top; i++) {
    if (strcmp(frameStatusList[i].name, functionName) == 0) {
      printf("Function already exists\n");
      return;
    }
  }
  //Creating a new frame
  top++;
  frameStatusList[top].number = top;
  strcpy(frameStatusList[top].name, functionName);
  frameStatusList[top].functionaddress = functionAddress;
  stackSize = stackSize + MINSTACKFRAMESIZE;
```

```
  frameStatusList[top].frameaddress = MEMSIZE - stackSize;
  frameStatusList[top].used = true;
  if (frameStatusList[top].frameaddress < maxStackSize) {
    printf("Stack overflow, cannot create frame\n");
    top--;
    stackSize -= MINSTACKFRAMESIZE;
    return;
  }
  else {
    printf("Frame created successfully\n");
    printf("Frame %d - Created frame %s at address %d\n", top, functionName,
        frameStatusList[top].frameaddress);
    printf("Stack Size After creating the Stack-Frame --> %d\n",stackSize);
  }
}
```

Function to delete the frame in the stack –

```
void deleteframe() {
  //Checking if the stack is empty or not
  if (top == -1) {
    printf("Stack is empty, no frames to delete\n");
    return;
  }
  //Deleting the frame
  int frameAddress = frameStatusList[top].frameaddress;
  printf("Frame Address --> %d\n", frameAddress);
  //Calculating the size of the frame
  int frameSize = 395 - frameAddress;
  printf("Frame Size --> %d\n", frameSize);
  //Setting the memory of the frame to 0
  memset(memory + frameAddress, 0, frameSize);
  printf("Deleting frame %s\n", frameStatusList[top].name);
  stackSize = stackSize - frameSize;
  top--;
  if (top >= 0) {
    frameStatusList[top].frameaddress = MEMSIZE - stackSize;
  }
  topFrame = topFrame + frameSize;
  printf("Frame deleted successfully\n");
}
```

Function to create an int inside the stack –

```
void createint(char integerName[], int value) {
  //It will be created in the top most frame of the stack
  struct framestatus currentframe = frameStatusList[top];
  int currentframeaddress = currentframe.frameaddress;
  printf("Frame %d - Before creating integer %s, the frame is at address %d
      in the Stack\n", top, integerName, currentframeaddress);
```

```
  //Checking if the frame is full or not
  if (395 - currentframeaddress + 4> MAXSTACKFRAMESIZE) {
    printf("The frame is full, cannot create more data on it\n");
    return;
  }
    printf("Stack Size After creating the integer --> %d\n",stackSize);
    topFrame -= sizeof(int);
    char* valuePtr = (char*)&value;
    //copying to the memory array
    memcpy(memory + topFrame, valuePtr, sizeof(int));
    printf("Created integer %s at address %d\n", integerName, topFrame);
    //Offset conditions - as initially we have to create a frame of 10 bytes,
       so when creating an int we need to use those 10 bytes first
    if(topFrame<currentframeaddress){
      int offset = currentframeaddress - topFrame;
      currentframeaddress = topFrame;
      if(offsetFlag == false){
        stackSize += offset;
        offsetFlag = true;
      }
      else{
        stackSize += sizeof(int);
      }
      printf("Stack Size Before creating the integer --> %d\n",stackSize);
    }
    printf("Frame %d - After creating integer %s, the frame is now at address
       %d in the Stack\n", top, integerName, currentframeaddress);
    frameStatusList[top] = currentframe;
    frameStatusList[top].frameaddress = currentframeaddress;
}
```

Function to create double inside the stack –

```
void createdouble(char doubleName[], double value) {
  struct framestatus currentframe = frameStatusList[top];
  int currentframeaddress = currentframe.frameaddress;
  printf("Frame %d - Before creating double %s, is at address %d in the
     Stack\n", top, doubleName, currentframeaddress);
  if (395 - currentframeaddress + 8> MAXSTACKFRAMESIZE) {
    printf("The frame is full, cannot create more data on it\n");
    return;
  }
    printf("Stack Size Before creating the double --> %d\n",stackSize);
    topFrame -= sizeof(double);


//This commented code is for typecasting double to string and then copying it
    to the memory array

      // char valuePtr[sizeof(double)];
      // sprintf(valuePtr, "%lf", value);
```

```
      char* valuePtr = (char*)&value;
      memcpy(memory + topFrame, valuePtr, sizeof(double));
    printf("Created double %s at address %d\n", doubleName, topFrame);
    if(topFrame<currentframeaddress){
      int offset = currentframeaddress - topFrame;
      currentframeaddress = topFrame;
      if(offsetFlag == false){
        stackSize += offset;
        offsetFlag = true;
      }
      else{
        stackSize += sizeof(double);
      }
      printf("Stack Size After creating the double --> %d\n",stackSize);
    }
    printf("Frame %d - After creating integer %s, is now at address %d in the
        Stack\n", top, doubleName, currentframeaddress);
    frameStatusList[top] = currentframe;
    frameStatusList[top].frameaddress = currentframeaddress;
}
```

Function to create char inside the stack –

```
//create char func - same as create int and create double - except the size
    changes to char
void createchar(char charName[], char value) {
    struct framestatus currentframe = frameStatusList[top];
    int currentframeaddress = currentframe.frameaddress;
    printf("Frame %d - Before creating char %s, the frame is at address %d in
        the Stack\n", top, charName, currentframeaddress);
    if (395 - currentframeaddress + 1 > MAXSTACKFRAMESIZE) {
        printf("The frame is full, cannot create more data on it\n");
        return;
    }
    printf("Stack Size Before creating the char --> %d\n", stackSize);
    topFrame -= sizeof(char);
    memcpy(memory + topFrame, &value, sizeof(char));
    printf("Created char %s at address %d\n", charName, topFrame);

    if (topFrame < currentframeaddress) {
        int offset = currentframeaddress - topFrame;
        currentframeaddress = topFrame;

        if (offsetFlag == false) {
            stackSize += offset;
            offsetFlag = true;
        }
        else {
            stackSize += sizeof(char);
        }
```

```
        printf("Stack Size After creating the char --> %d\n", stackSize);
    }

    printf("Frame %d - After creating char %s, the frame is now at address %d
        in the Stack\n", top, charName, currentframeaddress);
    frameStatusList[top] = currentframe;
    frameStatusList[top].frameaddress = currentframeaddress;
}
```

Function to create a buffer on the heap –

```
//function to create buffer on the heap
void createbuffer(char bname[], int size) {
  //Checking if the heap is full or not
  if (heapSize + size + 8 > maxHeapSize) {
    printf("The heap is full, cannot create more data\n");
    return;
  }
  //Maintaining the free list - linke a linked list
  struct freelist* temp = head;
  struct freelist* prev = NULL;
  while (temp != NULL) {
    if (temp->size >= size+4)
      break;
    prev = temp;
    temp = temp->next;
  }
  //Checking if a large enough block is available or not in the free list
  if (temp == NULL) {
    printf("Could not find a large enough block\n");
    return;
  }
  //If enough memory in the free list is available then we create a buffer
      through the allocated list
  int index = 0;
  while (strlen(alloc[index].name) != 0)
    index++;
  strcpy(alloc[index].name, bname);
  alloc[index].start = temp->start;
  alloc[index].size = size+4;

  // int* localPointer = (int*)&memory[frameStatusList[top].frameaddress];
  //creating a local pointer to the bufffer address in the stack
  int localPointer = temp->start;
  createint(bname, localPointer);
  if (temp->size == size+4) {
    if (prev == NULL)
      head = temp->next;
    else
      prev->next = temp->next;
  } else {
```

```
    temp->start += (size+4);
    temp->size -= (size+4);
  }
  printf("Heap Size Before creating the buffer --> %d\n", heapSize);
  heapSize += size+4;
  int header = size;
  memcpy(memory + alloc[index].start, &header, 4);
  for (int i = 0; i < size; i++) {
    memory[alloc[index].start + 4 + i] = (char)rand();
  }
  printf("Allocated %d bytes for buffer %s in the Heap\n", size, bname);
  printf("Heap Size After creating the buffer --> %d\n", heapSize);
}
```

Function to delete the buffer from the heap –

```
//function to delete buffer from the heap
void deletebuffer(char bname[]) {
  int index = 0;
  //Checking if the buffer exists or not
  while (strlen(alloc[index].name) != 0) {
    if (strcmp(alloc[index].name, bname) == 0)
      break;
    index++;
  }
  //If the buffer does not exist then we return
  if (strlen(alloc[index].name) == 0) {
    printf("Invalid buffer name\n");
    return;
  }
  int size = alloc[index].size;
  int startAddress = alloc[index].start;
  //Setting the memory of the buffer to 0
  struct freelist* temp = (struct freelist*)malloc(sizeof(struct freelist));
  temp->start = startAddress;
  temp->size = size;
  temp->next = head;
  head = temp;
  for (int i = 0; i < size; i++) {
    memory[startAddress + i] = 0;
  }
  //Deleting the buffer from the allocated list
  strcpy(alloc[index].name, "");
  alloc[index].start = 0;
  alloc[index].size = 0;
  heapSize -= size;
  printf("Deleted buffer %s\n", bname);
}
```

Show Memory - Printing the output function – In this function I have given the user an

option to print the resultant memory in either Integer, double, char or Hexa which they can choose according to the instructions that will be visible to them when they run the program. The stack list details and the heap details will be visible at the top and bottom of the memory display respectively.

```
void showmemory() {
    //Double d taken to print double values in the memory
    double d;
    char command[2];
    int count = 0;
    printf("\n");
    printf("Memory Printing format -->\n");
    printf("Enter I to print values in the form of Integers (The character and
        doubles will also be represented in Integers):\nEnter C to print
        values in the form of Chars (The Integers and Doubles will also be
        represented in Integers):\n Enter H to print values in the form of
        Hexadecimal (The Integers, Doubles Chars all also be represented in
        Hexadecimal):\nEnter D to print values in the form of Doubles (The
        Integers and Chars will also be represented in Doubles)");
    printf("\nInput -->");
    scanf("%s", command);
    if (strcmp(command, "I") == 0 || strcmp(command, "D") == 0 ||
        strcmp(command, "C") == 0 || strcmp(command, "H") == 0) {
        printf("Stack Frame List:\n");
        for (int i = 0; i <= top; i++) {
            printf("Frame %d: Name - %s, Function Address - %d, Frame Address
                - %d\n",
                   frameStatusList[i].number,
                   frameStatusList[i].name,
                   frameStatusList[i].functionaddress,
                   frameStatusList[i].frameaddress);
        }
        printf("\nMemory Contents:\n");
        for (int i = 395; i >= 0; i--) {
            if (strcmp(command, "I") == 0) {
                printf("%d --> %d\n", i, memory[i]);
            } else if (strcmp(command, "C") == 0) {
                printf("%d --> %c\n", i, memory[i]);
            }
            else if (strcmp(command, "H") == 0) {
                printf("%d --> %x\n", i, memory[i]);
            }
            else if (strcmp(command, "D") == 0) {
              memcpy(&d, &memory[i - sizeof(double)], sizeof(double));
              printf("%d --> %lf\n", i, d);
            }
        }
    } else {
        printf("Invalid command\n");
    }
    printf("\nHeap Details:\n");
```

```
    printf("----------------\n");
      struct freelist* temp = head;
  int heapAddress = 0;

  while (temp != NULL) {
    printf("Free Block at address %d, size %d\n", heapAddress + temp->start,
        temp->size);
    temp = temp->next;
  }
  for (int i = 0; i < 10; i++) {
    if (strlen(alloc[i].name) != 0) {
      printf("Allocated Buffer %s at address %d, size %d\n", alloc[i].name,
          alloc[i].start, alloc[i].size);
    }
  }
}
```

Main function to give the user an interactive like environment –

```
int main() {
  initialize();
  memset(memory, 0 , MEMSIZE);
  while (1) {
    char command[10], name[10];
    int address, size, value;
    double dvalue;
    char cvalue;

    printf("\nEnter command: ");
    scanf("%s", command);
    if (strcmp(command, "CF") == 0) {
      scanf("%s %d", name, &address);
      createframe(name, address);
    } else if (strcmp(command, "CI") == 0) {
      scanf("%s %d", name, &value);
      createint(name, value);
    } else if (strcmp(command, "CD") == 0) {
      scanf("%s %lf", name, &dvalue);
      createdouble(name, dvalue);
    } else if (strcmp(command, "CH") == 0) {
      scanf("%s %d", name, &size);
      createbuffer(name, size);
    } else if (strcmp(command, "CC") == 0) {
      scanf("%s %c", name, &cvalue);
      createchar(name, cvalue);
    }
    else if (strcmp(command, "DH") == 0) {
      scanf("%s", name);
      deletebuffer(name);
    } else if (strcmp(command, "SM") == 0) {
      showmemory();
```

```
    }else if (strcmp(command, "DF") == 0) {
  deleteframe();
}
    else {
      printf("Invalid command\n");
    }
  }
  free(head);
  return 0;
}
```

2 MakeFile

Make file for running the Schedulers has been added to the zip file. Use the commands 'make build' and then 'make run' to run the generated output file. Once you are inside the .out file, you'll have to insert the input.

```
build:
	gcc new.c -o out
run:
	./out
clean:
	rm out
rebuild: clean build
```

3 Outputs - Some Test Cases

INPUT -

```
Enter command: CF MAIN 2023
Stack Size Before creating the Stack-Frame --> 105
Frame created successfully
Frame 0 - Created frame MAIN at address 385
Stack Size After creating the Stack-Frame --> 115

Enter command: CI x 12
Frame 0 - Before creating integer x, the frame is at address 385 in the Stack
Stack Size After creating the integer --> 115
Created integer x at address 391
Frame 0 - After creating integer x, the frame is now at address 385 in the Stack

Enter command: CI x 32
Frame 0 - Before creating integer x, the frame is at address 385 in the Stack
Stack Size After creating the integer --> 115
Created integer x at address 387
Frame 0 - After creating integer x, the frame is now at address 385 in the Stack

Enter command: CI y 2
Frame 0 - Before creating integer y, the frame is at address 385 in the Stack
Stack Size After creating the integer --> 115
Created integer y at address 383
Stack Size Before creating the integer --> 117
Frame 0 - After creating integer y, the frame is now at address 383 in the Stack

Enter command: SM

Memory Printing format -->
Enter I to print values in the form of Integers (The character and doubles will also be represented in Integers):
Enter C to print values in the form of Chars (The Integers and Doubles will also be represented in Integers):
 Enter H to print values in the form of Hexadecimal (The Integers, Doubles Chars all also be represented in Hexadecimal):
Enter D to print values in the form of Doubles (The Integers and Chars will also be represented in Doubles)
Input -->I
```

OUTPUT -

```
Input -->I
Stack Frame List:
Frame 0: Name - MAIN, Function Address - 2023, Frame Address - 383

Memory Contents:
395 --> 0
394 --> 0
393 --> 0
392 --> 0
391 --> 12
390 --> 0
389 --> 0
388 --> 0
387 --> 32
386 --> 0
385 --> 0
384 --> 0
383 --> 2
382 --> 0
381 --> 0
380 --> 0
379 --> 0
```

```
16 --> 0
15 --> 0
14 --> 0
13 --> 0
12 --> 0
11 --> 0
10 --> 0
9 --> 0
8 --> 0
7 --> 0
6 --> 0
5 --> 0
4 --> 0
3 --> 0
2 --> 0
1 --> 0
0 --> 0

Heap Details:
----------------
Free Block at address 0, size 300

Enter command:
```

INPUT - Checking for DF

```
Enter command: CF NEWMAIN 2024
Stack Size Before creating the Stack-Frame --> 117
Frame created successfully
Frame 1 - Created frame NEWMAIN at address 373
Stack Size After creating the Stack-Frame --> 127

Enter command: CI Y 45
Frame 1 - Before creating integer Y, the frame is at address 373 in the Stack
Stack Size After creating the integer --> 127
Created integer Y at address 379
Frame 1 - After creating integer Y, the frame is now at address 373 in the Stack

Enter command: DF
```

OUTPUT -

```
Input -->I
Stack Frame List:
Frame 0: Name - MAIN, Function Address - 2023, Frame Address - 383

Memory Contents:
395 --> 0
394 --> 0
393 --> 0
392 --> 0
391 --> 12
390 --> 0
389 --> 0
388 --> 0
387 --> 32
386 --> 0
385 --> 0
384 --> 0
383 --> 2
382 --> 0
381 --> 0
380 --> 0
379 --> 0
378 --> 0
377 --> 0
376 --> 0
```

In the test case above we created a new frame b the Name of NEWMAIN and called DF. As we can see from the output that it has been removed from the status list and its contents have also been removed from the stack.

INPUT - Checking for Heap Conditions

```
Enter command: CH buf1 20
Frame 0 - Before creating integer buf1, the frame is at address 383 in the Stack
Stack Size After creating the integer --> 117
Created integer buf1 at address 385
Frame 0 - After creating integer buf1, the frame is now at address 383 in the Stack
Heap Size Before creating the buffer --> 0
Allocated 20 bytes for buffer buf1 in the Heap
Heap Size After creating the buffer --> 24

Enter command: SM
```

OUTPUT -

```
28 --> 0
27 --> 0
26 --> 0
25 --> 0
24 --> 0
23 --> 60
22 --> -37
21 --> -90
20 --> -77
19 --> -21
18 --> -23
17 --> -69
16 --> -15
15 --> -15
14 --> 73
13 --> -112
12 --> 82
11 --> -82
10 --> -42
9 --> 108
8 --> -31
7 --> -124
6 --> -66
5 --> 35
4 --> 41
3 --> 0
2 --> 0
1 --> 0
0 --> 20

Heap Details:
----------------
Free Block at address 24, size 276
Allocated Buffer buf1 at address 0, size 24

Enter command:
```

INPUT - Checking for Heap Conditions

```
Enter command: CH buf2 95
Frame 0 - Before creating integer buf2, the frame is at address 383 in the Stack
Stack Size After creating the integer --> 117
Created integer buf2 at address 381
Stack Size Before creating the integer --> 121
Frame 0 - After creating integer buf2, the frame is now at address 381 in the Stack
Heap Size Before creating the buffer --> 24
Allocated 95 bytes for buffer buf2 in the Heap
Heap Size After creating the buffer --> 123

Enter command: SM
```

OUTPUT -

```
28 --> -121
27 --> 0
26 --> 0
25 --> 0
24 --> 95
23 --> 60
22 --> -37
21 --> -90
20 --> -77
19 --> -21
18 --> -23
17 --> -69
16 --> -15
15 --> -15
14 --> 73
13 --> -112
12 --> 82
11 --> -82
10 --> -42
9 --> 108
8 --> -31
7 --> -124
6 --> -66
5 --> 35
4 --> 41
3 --> 0
2 --> 0
1 --> 0
0 --> 20

Heap Details:
----------------
Free Block at address 123, size 177
Allocated Buffer buf1 at address 0, size 24
Allocated Buffer buf2 at address 24, size 99

Enter command:
```

INPUT - Checking for Heap Conditions

```
Enter command: DH buf1
Deleted buffer buf1

Enter command: SM
```

OUTPUT -

```
28 --> -121
27 --> 0
26 --> 0
25 --> 0
24 --> 95
23 --> 0
22 --> 0
21 --> 0
20 --> 0
19 --> 0
18 --> 0
17 --> 0
16 --> 0
15 --> 0
14 --> 0
13 --> 0
12 --> 0
11 --> 0
10 --> 0
9 --> 0
8 --> 0
7 --> 0
6 --> 0
5 --> 0
4 --> 0
3 --> 0
2 --> 0
1 --> 0
0 --> 0

Heap Details:
---------------
Free Block at address 0, size 24
Free Block at address 123, size 177
Allocated Buffer buf2 at address 24, size 99

Enter command:
```

In the test cases above we created a new buffer buf1 in the Heap. In the output you can see that memory for it has been allocated according to its size + 4 as it also stores the size as mentioned in the pdf. The free list being maintained can also be seen in the output. We then created buf2. And then deleted buf1. The outputs are coming correctly as we can see that buf1 has been cleared and extra space that was created is added to the free list. Input - Condition to print doubles in the memory

```
Enter command: CF main 1010
Stack Size Before creating the Stack-Frame --> 105
Frame created successfully
Frame 0 - Created frame main at address 385
Stack Size After creating the Stack-Frame --> 115

Enter command: CD pi 3.14
Frame 0 - Before creating double pi, is at address 385 in the Stack
Stack Size Before creating the double --> 115
Created double pi at address 387
Frame 0 - After creating integer pi, is now at address 385 in the Stack

Enter command: SM

Memory Printing format -->
Enter I to print values in the form of Integers (The character and doubles will also be represented in Integers):
Enter C to print values in the form of Chars (The Integers and Doubles will also be represented in Integers):
 Enter H to print values in the form of Hexadecimal (The Integers, Doubles Chars all also be represented in Hexadecimal):
Enter D to print values in the form of Doubles (The Integers and Chars will also be represented in Doubles)
Input -->D
```

OUTPUT -

```
Memory Contents:
395 --> 3.140000
394 --> 0.000000
393 --> 0.000000
392 --> -0.000000
391 --> 42769804971223216000000000000000000000000000000000000000000000000000000000000000000000.000000
390 --> -86796737120581224000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000.000000
```

To print doubles that have been created through CD, Input D in Show Memory. Similarly to print char values created through Create Char func, input C in the show memory.