Homework 5

Procedures, Stack & Heap in MIPS

Lecturer: Dr. Yisroel Mirsky

Objective:

Implement a Customer Database Management System in MIPS Assembly to handle customer records efficiently. Your implementation should demonstrate mastery of procedure calls, control flow, syscalls, and appropriate usage of the **heap** and **stack**, with an emphasis on spilling and restoring registers.

Requirements:

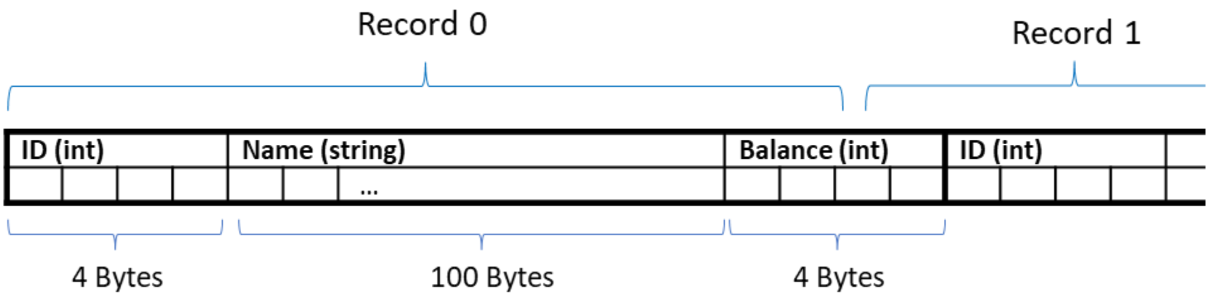
1) **Main Menu**

When the program starts, your Customer Database Management System program should display the following options to the user:

- `add_customer`
- `display_customer`
- `update_balance`
- `delete_record`
- `exit_program`
    - You must make sure that the user's selection is legal
      I.e. the input is a number between 1 to 5.
      If not, print: "Invalid choice. Please enter a number between 1 and 5".

2) **Customer Record Structure (struct):**
   Each customer record contains the following
   - Customer ID (integer)
   - Customer Name (string)
   - Customer Balance (integer)

Tip: Use a fixed amount of space for the string so that each record will use the exact same amount of space in the heap. This way you will be able to store the records in an array.
Example:



3) Functional Requirements:
The program should support the following operations:

- **add_customer(ID, name, balance)**: Adds a new customer record to the database.
    - Input: First enter ID, then name, and then enter the balance
      (refer to the workingExample.exe file for syntax)
    - You can assume that the ID and balance given are valid values.
    - You must check that a customer with the given ID doesn't already exist (IDs are unique in the database).
        - If the ID already exists:
          Output: "Error: Customer <ID> already exists"
          Else:
          Output: "Success: Customer <ID> was added"

- **display_customer(ID)**: Displays a specific customer record by their ID
    - Input: ID
    - If the customer exists:
        - Output: "Success: <ID>, <Name>, <Balance>"
          Else (Invalid id):
          Output: "Error: Customer <ID> doesn't exist"

- **update_balance(ID, new_balance)**: Updates the balance of a specific customer
    - Input: enter <ID> and then enter <New Balance>
    - If the customer exists
        - If balance is valid (balance >=0 and balance<=99999)
            - Output: "Success: <ID>, <Name>, <Balance>"
              Else:
              Output: "Error: The inputted balance isn't valid"
    - If the customer does not exist (invalid id):
        - Output: "Error: Customer <ID> doesn't exist"

- **delete_customer(ID):** Deletes a specific customer record by their ID.
  This feature is bonus (10 points)
  If you do not implement it, then
  simply return to main menu if "`4) delete_record`" is selected
    - Input: <ID>
    - If the customer exists:
        - Output: "Success: Customer <ID> deleted"
          Else (Invalid id):
          Output: "Error: Customer <ID> doesn't exist"

- **exit_program()**: Exits the program gracefully
    - Output: "Exiting program"...

4) Procedure Calls:

- Implement the functions **add_customer, display_customer, update_balance**, and **delete_customer** as separate procedures.

- Ensure you demonstrate appropriate spilling and restoring of registers when calling procedures. Using frame pointers is optional but recommended.
- The function must not get any input directly from the user. Your menu must receive the input from the user and then call the corresponding function with the arguments.
  Tip: declare empty words and string arrays in the .text section as buffers. Use them to collect data from the user, and then pass their addresses to the functions.

5) Control Flow:
   - Incorporate decision structures and loops where necessary for navigation and user input validation.
   - **Syscalls**: Utilize syscalls for reading input, writing output, and memory allocation.
   - **Heap** and **Stack Usage**: Manage customer records on the heap. Allocate space on the heap as needed with syscall 9 (sbrk)
   - Demonstrate proper usage of the stack for procedure calls and returns, and ensure appropriate register spilling and restoration.

Important Notes:

You are provided with the file `workingExample.exe`. This file is a complete example of what you need to implement. Use it to check yourself and compare your code's behavior. Your implementation should behave the exact same as the one in the example file.

- If you find some discrepancy between the example and the submission instructions, please let us know.
- To run the file, just click the workingExample.exe file
- The format of strings you print are very important since the automatic grading systemis based on them. Please ensure you use them correctly
- You can assume that inputs will always be with the correct type.
  If an integer is expected, the input will be an integer. Etc.

Grading Criteria:

- **Functionality (50 points):**
    - The code should function as instructed.
  - **Procedure Calls (10 Points):**
    - Proper implementation and usage of procedure calls.
    - Demonstrated spilling and restoring of registers during procedure calls.
  - **Control Flow (10 Points):**
    - Correct implementation of decision structures and loops.
  - **Syscalls Practice (10 Points):**
    - Accurate and efficient use of syscalls for various operations.
  - **Heap Usage (10 Points):**
    - Effective and error-free dynamic allocation and deallocation of memory on the heap.
  - **Stack Usage (10 Points):**
    - Proper usage and management of the stack for procedure call handling.

Submission:

Submit your MIPS assembly code with the filename
<ID>.asm if you are submitting alone
or as <ID1>_<ID2>.asm if submitting as a pair.

Your code will be tested for cheating (copied code) using anti-plagiarism software.

Happy coding!