Operating Systems Laboratory (CS39002)

Assignment 5: Hands-on experience for creating better memory management systems

1. **Structure of internal page table**
    - The page table is represented as an array of UDT page table entries
    - The page table consists of various page table entries bounded by a specified size
    - A particular page table entry has the following fields
        - **Offset** : the location in physical memory where the variable is kept
        - **sz:** size of the variable (used when the variable is of type array)
        - **is_valid** : valid/invalid bit to know whether the current page table entry is relevant or not
        - **type :** type of variable. Eg int, char ,etc
    - We have used the page table as an array of objects of type page_table_entries because this makes it easy to allocate page_table_entries which have same size.
    - **Macros**
        - **MEM_SIZE :** max physical memory size (hypothetical physical memory)
        - **PT_SIZE :** max page table size
        - **STK_SIZE :** max stack size
    - Functions
        - **init()** : initializes the page table
        - **shift_from()** : It shifts all the element of the page table by 1 to add a new element
        - **Search_insert()** : it first searches for an interval and then adds a new page table entry
        - **shiftLeftKtimes()** : it shifts the page table entry and reduces its size by k times when the garbage collection is done

2. **Additional data structures**
    - **Stack**

    - This is used to keep track of the scope the variables i.e. to push and pop the local variables into stack when entering and exiting the functions
    - It has some standard boiler plate code like
        - **Top**
        - **Peak**
        - **Pop**
        - **Push, etc**

- **Data:**
    - This is a class used to implement the various type of data types given in the assignment like medium int, char, etc. It also makes it easy to keep track of the page table entity of variable itself
    - It makes the overall tracking of variables easy

- **dataTable**
    - it is a simple array which is considered as the hypothetical physical memory and where the values of the declared variables are kept
    - The array is of type int such that the memory stays word aligned
    - We can store int type variables easily but what about other types such as char and medium int.
    - For char type data we've used bitmask to to store each char in a in a block of 4 bytes by assign each char to each respective byte to reduce internal fragmentation
    - For medium int we were unable to reduce the internal fragmentation as 3 bytes were occupied in a 4 byte block

3. **Impact of mark and sweep for demo1 and demo2**
    - If we go on creating objects we might get Out Of Memory error since it is not possible to allocate heap memory to objects.
    - So we need to clear heap memory by releasing memory for all those objects which are no longer referenced by the program so that the space is made available for subsequent new objects

- Any garbage collection algorithm must perform 2 basic operations. One, it should be able to detect all the unreachable objects and secondly, it must reclaim the heap space used by the garbage objects and make the space available again to the program.
- Mark and Sweep Algorithm in two phases as listed and described further as follows:
    - **Mark phase**
    - **Sweep phase**

4. **Logic for running compact in Garbage collection**
    - Compaction helps in reducing external fragmentation which makes it possible for large blocks of memory to be allocated whereas if the holes in memory were small and distributed then despite memory being available we won't be able to allocate the required amount of memory.
    - We have divided the algorithm in two parts
        - Compaction
        - Defragmentation
    - In compaction its is merging the page table entries with is_valid bits 0 i.e not being used page table entries into one big chunk
    - Then defragmentation is shifting the page table entries with is_valid bit 1 so that we are able to avoid memory holes .

5. **Use of Locks**
    - Yes we are using locks. It is possible for library to deal with the data table or the page table simultaneously when the garbage collector is compacting and the user is also doing some task like assigning or creating some variable in the data table (hypothetical physical memory).
    - Simple single lock for whole memory would make the execution serial as the main process won't be able to access the memory(any part) when the garbage collector is shifting blocks of memory.
    - This has a major drawback as when the garbage collector is accessing only a small portion of memory but whole memory is locked and the main process can't even access the parts not being used by the garbage collector.
    - Therefore, it's necessary to use locks to avoid data races.

Group - 20
Parth Tusham 19CS30034
Shashwat Shukla 19CS10056