INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY ASSIGNMENT 6

(Operating Systems Laboratory)

GROUP 31

Likhith Reddy Morreddigari 20CS10037

Shivansh Shukla 20CS10057

Venkata Sai Suvvari 20CS10067

Shashank Goud Boorgu 20CS30013

Data Structures

Page Table Entry

Each page table entry is of the following type:

```
struct pageTableEntry
{
    uint32_t headOffset;                    // stores offset of head of list
   corresponding to this entry if isValid=1
    u_int nextFreeIdx : 15;                 // stores index of next entry in
   PageTable having isValid=0
    u_int isValid : 1;                      // isValid = 0 => chunk is free
};
```

Each page entry is of size of 48 bits.

Data Members

- **headOffset:** The offset of header of memory chunk corresponding to this page table entry.
- **nextFreeIdx:** The pointer to the next free entry in the page table. We're basically maintaining a linked list of free entries.
- **isValid:** Marks whether the chunk is free (isValid = 0) or occupied(isValid = 1).

Functions

- **pageTableEntry()** : The constructor for the structure when it is declared. This initializes the headOffset to 0, nextFreeIdx to MAX_PAGE_TABLE_ENTRIES which is 16384 and the isValid bit to 0.

Page Table

The page table structure:

```
class PageTable
{
    pageTableEntry ptearr[MAX_PAGE_TABLE_ENTRIES];
    uint16_t firstFreeIdx;
    uint16_t lastFreeIdx;
    uint16_t currSize;
    public:
    PageTable() ;
    void insert(const pageTableEntry&);
    uint16_t findFreeBlockIdx(uint32_t);
    uint16_t allocateList(uint32_t);
    void assignValUtil(uint16_t, uint32_t, uint32_t);
    uint32_t getValUtil(uint16_t, uint32_t);
    uint16_t findPTidxByHeadOffset(uint32_t);
    void freeList(uint16_t);
};
```

The page table is the array of the page table entries. This has a dedicated space at the beginning of the memory allocated at the beginning. This is an explicit list storing the free entries where end of each free entry points to the next free entry, in the allotted memory which is an implicit list in which each chunk stores its size and given information about whether it is valid or not.

Data Members

- **ptearr:** The array of page table entries.
- **firstFreeIdx and lastFreeIdx:** The free entries are stored as the linked list. The firstFreeIdx points to the beginning of the list and the lastFreeIdx points to the end of the list. When an element is freed, its valid bit is changed to 0, and it made as the next to the lastFreeIdx and the lastFreeIdx is update to the current index.
- **currSize:** The present number of valid entries in the page table.

Functions

- **PageTable()**: This initializes the firstFreeIdx, lastFreeIdx and currSize to 0. It also the sets the respective attributes of the page table entries.
- **insert(const pageTableEntry& pte):** Inserts a new entry in the page table.
- **findFreeBlockIdx(uint32_t bytes):** Returns the index of the first chunk that has the **bytes** number of bytes available using **FIRST FIT**.
- **freeList(uint16_t idx):** Frees the entry of the given index.
- **findPTidxByHeadOffset(uint32_t headoffset):** Returns the pointer index that matches given headoffset.

- **allocateList(uint32_t bytes):** Allocates teh required memory using the findFreeBlockIdx function.
- **assignValUtil(uint16_t idx, uint32_t listIndex, uint32_t val):** Assigns the given value at the given index.
- **getValUtil(uint16_t idx, uint32_t listIndex):** Gives the value at the given position.

FrameStack

The Frame Stack structure:

```
class FrameStack
{
    uint32_t fptrs[MAX_FRAME_CNT];           // array implementation of frame
   pointer stack
    uint32_t currIdx;                        // current index in fptrs[]
    public:
    FrameStack();
    ~FrameStack() {}
    void push(const uint32_t&);
    void pop();
    uint32_t top();
    friend class Stack;                     // so that Stack class can access
   its private data members
};
```

For storing the frame pointers.

Data Members

- **fptrs:** The arrays for storing the frame pointers.
- **currIdx:** The position of current frame pointer.

Functions

- **FrameStack():** Initializes the currIdx to 0.
- **push(const uint32_t&):** Push the new frame pointer to fptrs array.
- **pop():** Removes the recent frame pointer.
- **top():** Returns the recent frame pointer.

Stack Entry

The Stack Entry structure:

```
struct stackEntry
{
    uint16_t pageTableIdx;              // stores index of page table entry
   corresponding to this
    char* listName;                     // Name of list
    uint8_t listNameLen;                // assuming name of list can fit in 8
   bits
    stackEntry()
    {
        pageTableIdx = MAX_PAGE_TABLE_ENTRIES;
        listName = NULL;
        listNameLen = 0;
    }
};
```

The entry in the stack.

Data Members

- **pageTableIdx:** Stores index of page table entry corresponding to this.
- **listName:** The name of the list.
- **listNameLen:** The length of the list name.

Functions

- **stackEntry():** Initializes the pageTableIdx to MAX_PAGE_TABLE_ENTRIES, listName to NULL, listNamelen to 0.

Stack

The stack structure:

```
class Stack
{
    uint32_t sp;                            // stack pointer
    FrameStack fstack;                      // stack of frame pointers
    public:
    Stack(uint32_t);
    ~Stack() {}
    void newFuncBeginUtil();                    // Save the current stack
   pointer as frame pointer in frame stack
    void FuncEndUtil();                         // Pop frame pointer entry
    void push(const stackEntry &);          // Push stack entry
    stackEntry top();                       // Returns top entry from stack
    void pop();                             // Pop stack entry
```

```
    uint16_t findPTidxByName(char*, uint8_t, uint32_t, uint32_t);
    uint16_t freeLastListCurrStack();
    void copyByValueUtil(char*, uint8_t, uint32_t);
    friend uint32_t getCurFramePointer();
    friend uint32_t getCurStackPointer();
};
```

The stack of the program. This has dedicated space allocated at the end of the memory that is created at the beginning.

Data Members

- **sp:** The stack pointer.
- **fstack:** The stack of frame pointers.

Functions

- **Stack():** Initializes the stack functions.
- **newFuncBeginUtil():** Save the current stack pointer as frame pointer in frame stack.
- **FuncEndUtil():** Pop frame pointer entry from the frame stack.
- **push(const stackEntry &):** Push stack entry into the stack.
- **top():** Returns the current value in the stack.
- **pop():** Pop stack entry from the stack.
- **findPTidxByName(char*listName, uint8_t listNameLen, uint32_t currsp, uint32_t currfp):** Returns the index of the list with given name and the stack pointer and frame pointer.
- **freeLastListCurrStack():** Frees the recent list that was allocated.

Functions

Main Functions

- **createMem(uint32_t bytes):** Creates a memory segment with given size and also allocates space for page table at the beginning of the segment and for stack at the end of the segment.
- **createList(uint32_t size, char* listName, uint8_t listNameLen):** Creates an implicit list of memory where in each chunk of memory, 4 bytes are reserved for header, which stores the length of chunk and using which, we can hop to the next chunk. Similarly, 4 bytes are reserved for tail, which stores same information as head and with this, previous chunk can be reached. This implicit list is made explicit using the page table. In page table, we maintain a linked list of free entries where each entry stores index of next free entry in page table. Using this, we can find free block much faster than simply iterating over each chunk and check whether it is free or not.
- **assignVal(char* listName, uint8_t listNameLen, uint32_t offset, uint32_t val, uint32_t currsp = 0, uint32_t currfp = 0):** Assigns the value to the list element at the given offset, stack pointer, frame pointer and list name. If stack pointer and frame pointer are not provided, then it perform lookup in current stack frame.
- **freeElem() and freeElem(char* listName, uint32_t listNameLen):** These are overloaded functions. Without any arguments, it will free the lists that are in the current stack and with arguments, it will free the list with the list name. When no arguments are provided, then it pops the stack entry and goes to corresponding page table entry to set it free.

Other Helper Functions

- **newFuncBegin():** This function should be called whenever a user enters a new function scope so that its stack scope allocated. Basically, it pushes the current stack pointer to frame stack.
- **FuncEnd():** This function is called whenever a user exits from the function scope so that its allocated stack is cleared by popping frame stack.
- **getVal(char* listName, uint8_t listNameLen, uint32_t offset, uint32_t currsp, uint32_t currfp):** Returns the value of the given list with given offset given the current

stack pointer and frame pointer. If stack pointer and frame pointer are not provided, then it perform lookup in current stack frame.

- **getCurFramePointer():** Gets the frame pointer of the current scope.
- **getCurStackPointer():** Gets the stack pointer of the current scope.
- **deleteMem():** Deleallocates all the memory allocated by the createMem.

Performance

Overhead memory statistics

- The initial portion of memory is reserved for storing PageTable, which is **131080 bytes**.
- The ending portion of memory is reserved for storing FrameStack and Stack, which is **40008 bytes**.
- Furthermore, Stack pointer is allowed to decrease upto **10000 bytes**.
- Overall overhead is nearly **181 KB = 0.0724%** of total memory occupied(250 MB).

Impact of freeElem()

- With freeElem(), the whole mergesort is running in **0.621 seconds**, averaged over 100 iterations.
- Maximum memory footprint averaged over 100 iterations : **4780096 bytes**.
- With freeElem(), the memory chunk assigned within a function is freed as soon as that function exits. Hence, our program can satisfy more memory requirements.
- Without the freeElem(), a heavy program can stop after sometime due to no free space available.
- Without freeElem(), Maximum memory footprint averaged over 100 iterations : **5376807 bytes**.
- Without freeElem(), it took **2.32 seconds**, as searching a free chunk of memory would be more expensive.

Performance Criteria

- **MAX Performance :** The memory utilisation would be maximum in case of a recursive function, in which we are not supposed to free the memory till it returns from recursion stack. Furthermore, memory utilisation would be maximum when lists of sufficient size are being allocated.

- **MIN Performance :** The memory utilisation would be minimum when the user is allocating quite a few lists, that too of smaller size. In this case, we are already reserving nearly 181 KB of memory for Pagetable and Stack, hence performance would be poorer.

Locks Usage

- The whole program runs as a single thread so there is no usage of locks.
- Also, there is no significant scope of concurrency, hence optimal performance can be achieved by just using single thread.