MP4 Report

Introduction

This machine problem covers the machine problem 4. With this program we improve the page table management to be able to accommodate large numbers and sizes of address space. This is done by moving pages to a virtual memory pool. This allows the user to utilize large address spaces. The report will explain each addition to the source files that was necessary to accomplish this task.

Machine Problem 3 File Modifications

```
PageTable::PageTable()
```

The constructor needed modifications to create the implementation of virtual memory. We set the last entry of the page directory to itself to enable page recursion. Additionally, all virtual memory pools are set to none since this is the creation where nothing should be allocated here yet.

```
void PageTable::handle_fault(REGS * _r)
```

In the handle_fault function, we iterate through the list of VMPools to find an acceptable frame pool. This is verified by using the is_legitimate helper function. The fault is then handled by modifying the necessary Page Table pages and directories.

```
void PageTable::register_pool(VMPool * _vm_pool)
```

This function completes its task of registering virtual memory pools by iterating through the data structure used in this implementation. It goes through the list and checks for the index where it is null. Then it uses this index to register it in the list of virtual memory pools.

Machine Problem 4 File Additions

```
void VMPool::VMPool(...)
```

The constructor simply assigns all required data members for the class VMPool. This is the class which is helping us extend MP3 page manager to handle large address spaces. The fields include base_address, _size, _frame_pool, _page_table, totalRegions, totalRegionSize, and listOfRegionDescriptors. The listOfRegionDescriptors is a pointer to an instance of a struct used to help in the logic used in VMPool. This struct contains members which provide the address of a region and the length in bytes of the corresponding region represented by the struct.

```
void VMPool::allocate(unsigned long _size)
```

This function handles the allocation of VMPools. It does so by first checking if there is space for this input in the total region of memory reserved for this purpose. If it fits then it places it in the

correct location by using the data members which provide addresses. Related members which track number of regions and size are updates in this process.

```
void VMPool::release(unsigned long _start_address)
```

The release function is needed for when a program that was allocated in virtual memory needs to have its memory freed. This might be because the process has terminated or whatever other reason its memory is needed. It begins by iterating through the data structure that tracks all memory regions and checks for the index where the address data member matches with the input address. Once the program finds the index, it can then release its memory by modifying the data structures which allow this storage to be possible.

```
void VMPool::is_legitimate(unsigned long _address)
```

This is a helper function which the page manager uses to determine if the address provided as input is acceptable within our implementation of virtual memory. It is used within the handleFault function.