Report

1. kernel.h 和 kernel.cc

我們首先用 lib 中的 list.h，在 kernel.h 中新增一個 List<int>* 型別的freeFrameList，用來讓 kernel 管理所有的 free frame，如下：

```
/* MP2 */
List<int> *freeFrameList;
```

並在kernel.cc的Initialize()中初始 freeframeList，如下：

```
freeFrameList = new List<int>;
for(int i=0 ; i<NumPhysPages ; i++) freeFrameList->Append(i);
```

2. addrspace.cc

這是此次作業主要更改的地方，我們先將原本 Addrspace 的建構式刪掉，因為在原本的建構式裡是直接 new 一個大小是包含整個 physical page 的數目的 page table，但在 multiprogramming 裡頭我們在 context switch 之後會根據 load 進來的 process 而用符合那 process 大小的 page table，因此我們先將原本的建構式刪掉變成：

```
AddrSpace::AddrSpace()
{
    // pageTable = new TranslationEntry[NumPhysPages];
    // for (int i = 0; i < NumPhysPages; i++) {
  // pageTable[i].virtualPage = i;  // for now, virt page # = phys page #
  // pageTable[i].physicalPage = i;
  // pageTable[i].valid = FALSE;
  // pageTable[i].use = FALSE;
  // pageTable[i].dirty = FALSE;
  // pageTable[i].readOnly = FALSE;
    // }
    //
    // // zero out the entire address space
    // bzero(kernel->machine->mainMemory, MemorySize);
}
```

而我們在下面的 Load(char *fileName) 裡頭進行修改，我們能看到原先打好有算 process 會佔多少個 pages，也就是 numPages，以及 page table size：

```
numPages = divRoundUp(size, PageSize);
size = numPages * PageSize;
```

然後將檢查的函式 Assert() 裡頭改成檢查 process 所需要的 numPages 是否小於或等於 kernel所管理的 freeFrameList 中所含的 frame 數目，也就是目前能被使用的閒置 frames 數目：

```
ASSERT(numPages <= kernel->freeFrameList->NumInList());
        // check we're not trying
        // to run anything too big --
        // at least until we have
        // virtual memory
```

當以上的檢查完畢後，便能根據前面所算的所需 page 數來 new page table 了，並用一個迴圈將這個 page table 初始完畢。而每次迴圈裡頭做的是先用list.h所提供的函式取出 free frame number，並將這個frame從freeFrameList中 remove 掉。再以迴圈的 index 來依序給定 virtual page number，接著將先前從 freeFrameList 中拿到的 frame number 放入目前 page table entry 裡對應的 physicalPage，而因為是要被 load 進 memory 中準備執行，會將 valid 值設為 true，而其他 use, dirty, readOnly 都先設為 false，如下：

```
pageTable = new TranslationEntry[numPages];
for (int i = 0; i < numPages ; i++) {
int freeFrame = kernel->freeFrameList->Front();
kernel->freeFrameList->RemoveFront();
pageTable[i].virtualPage = i;
pageTable[i].physicalPage = freeFrame;
pageTable[i].valid = TRUE;
pageTable[i].use = FALSE;
pageTable[i].dirty = FALSE;
pageTable[i].readOnly = FALSE;
```

而在建好 page table 後，在下方把 code 以及 data segments 放進 memory 時，我們改成使用原本他有幫我們寫好的 Addrspace::translate() 來將virtual address 轉換成 physical address 並用此實體位址去 mainMemory 讀取，所修改部分變成如下：

```
unsigned int paddr;
AddrSpace::Translate( noffH.code.virtualAddr, &paddr, 0);
```

而最後當 process 要被 swap out 出 memory 時會呼叫 Addrspace 的解構式，因此我們在解構式裡頭多加了一個 for 迴圈跑整個 page table 來將原本所佔用的 frame 放回 freeFrameList 後才 delete 掉 page table，如下：

```
AddrSpace::~AddrSpace()
{
    for(int i=0 ; i<numPages ; i++)
        if(pageTable[i].valid)
            kernel->freeFrameList->Append(pageTable[i].physicalPage);
    delete pageTable;
}
```

3. 與 spec 結果圖不同的原因

由於 cpu 的資源是由 OS 分配的，因此在multiprogramming下若沒有特地處理 synchronization 的話，process是有可能會在我們期待的一連串動作還沒做完就被 context switch 的，而我們在上次 MP1 所做的 PrintInt() 裡是分兩行 PutChar() 數字及換行字元的，上次的實作方法如下：

```
while(index >=0 ) synchConsoleOut->PutChar(num[index--]);
synchConsoleOut->PutChar('\n');
```

因此是有可能印完數字就被 context switch 並換另一個 process 執行而後才又 switch 回來繼續做完 PutChar('\n')。

4. Group contribution

103062121 劉亮廷　trace code　50%
103062238 林子淵　implement code 50%