```
AddrSpace::AddrSpace(char * filename)
{
   NoffHeader noffH;
   unsigned int i, size, j;
   unsigned int numCodePage, numDataPage; // số trang cho phần code và phần initData
   int lastCodePageSize, lastDataPageSize, firstDataPageSize,tempDataSize; // kích
//thước ghi vào trang cuối Code, initData, và trang đầu của initData
   OpenFile* executable = fileSystem->Open(filename);

   if (executable == NULL){
     printf("\nAddrspace::Error opening file:  %s",filename);
     DEBUG(dbgFile,"\n Error opening file.");
     return;
   }

//đọc header của file
   executable->ReadAt((char *)&noffH, sizeof(noffH), 0);
   if ((noffH.noffMagic != NOFFMAGIC) &&
         (WordToHost(noffH.noffMagic) == NOFFMAGIC))
     SwapHeader(&noffH);
   ASSERT(noffH.noffMagic == NOFFMAGIC);

   addrLock->Acquire();

//  how big is address space?
   size = noffH.code.size + noffH.initData.size + noffH.uninitData.size
                              + UserStackSize;       // we need to increase the size
                                                     // to leave room for the stack
   numPages = divRoundUp(size, PageSize);
   size = numPages * PageSize;

   // Check the available memory enough to load new process
   //debug

   if (numPages > số trang còn trống){
     printf("\nAddrSpace:Load: not enough memory for new process..!");
     numPages = 0;
     delete executable;
     addrLock->Release();
     return ;
   }

   // first, set up the translation

   pageTable = new TranslationEntry[numPages];
   for (i = 0; i < numPages; i++) {
```

```
      pageTable[i].virtualPage = i;   // for now, virtual page # = phys page #
      pageTable[i].physicalPage = tìm 1 trang trống và đánh dấu đã sử dụng;
      pageTable[i].valid = TRUE;
      pageTable[i].use = FALSE;
      pageTable[i].dirty = FALSE;
      pageTable[i].readOnly = FALSE;  // if the code segment was entirely on
                                      // a separate page, we could set its
                                      // pages to be read-only
     // xóa các trang này trên memory
      bzero(&(machine->mainMemory[pageTable[i].physicalPage*PageSize]), PageSize);
printf("phyPage %d \n",pageTable[i].physicalPage);
    }

    addrLock->Release();

    // Calculate numCodePage and numDataPage
    numCodePage = divRoundUp(noffH.code.size, PageSize);

    // Calculate lastCodePageSize
    lastCodePageSize = noffH.code.size - (numCodePage-1)*PageSize;
    tempDataSize = noffH.initData.size - (PageSize - lastCodePageSize);

    if (tempDataSize < 0){
          numDataPage = 0;
          firstDataPageSize = noffH.initData.size;
    }
    else{
          numDataPage = divRoundUp(tempDataSize, PageSize);
          lastDataPageSize = tempDataSize - (numDataPage-1)*PageSize;
          firstDataPageSize = PageSize - lastCodePageSize;
    }

// Copy the Code segment into memory
    for (i = 0; i < numCodePage; i++) {
             //      if(noffH.code.size > 0)
          executable->ReadAt(&(machine->mainMemory[noffH.code.virtualAddr]) +
pageTable[i].physicalPage*PageSize, i<(numCodePage-1)?PageSize:lastCodePageSize,
noffH.code.inFileAddr + i*PageSize);
    }

          //Check whether last page of code segment is full and copy the first part of
//initData segment into this page
          if (lastCodePageSize < PageSize){
             // Copy initData into the remain part of lastCodePage
             if (firstDataPageSize > 0)
```

```
            executable->ReadAt(&(machine-
>mainMemory[noffH.code.virtualAddr])+(pageTable[i-1].physicalPage*PageSize +
lastCodePageSize), firstDataPageSize, noffH.initData.inFileAddr);
        }

// Copy the remain of initData segment into memory
    for (j = 0; j< numDataPage; j++) {
//          if(noffH.initData.size > 0)
            executable->ReadAt(&(machine-
>mainMemory[noffH.code.virtualAddr])+pageTable[i].physicalPage*PageSize,
j<(numDataPage-1)?PageSize:lastDataPageSize,
                        noffH.initData.inFileAddr + j*PageSize + firstDataPageSize);
        i++;
        }

    delete executable;
    return ;
}
```