Mike Ames
4/5/2018
CIS 452

Program 3

In completing this assignment, I tried to adhere to the techniques that real operating systems use to manage memory.

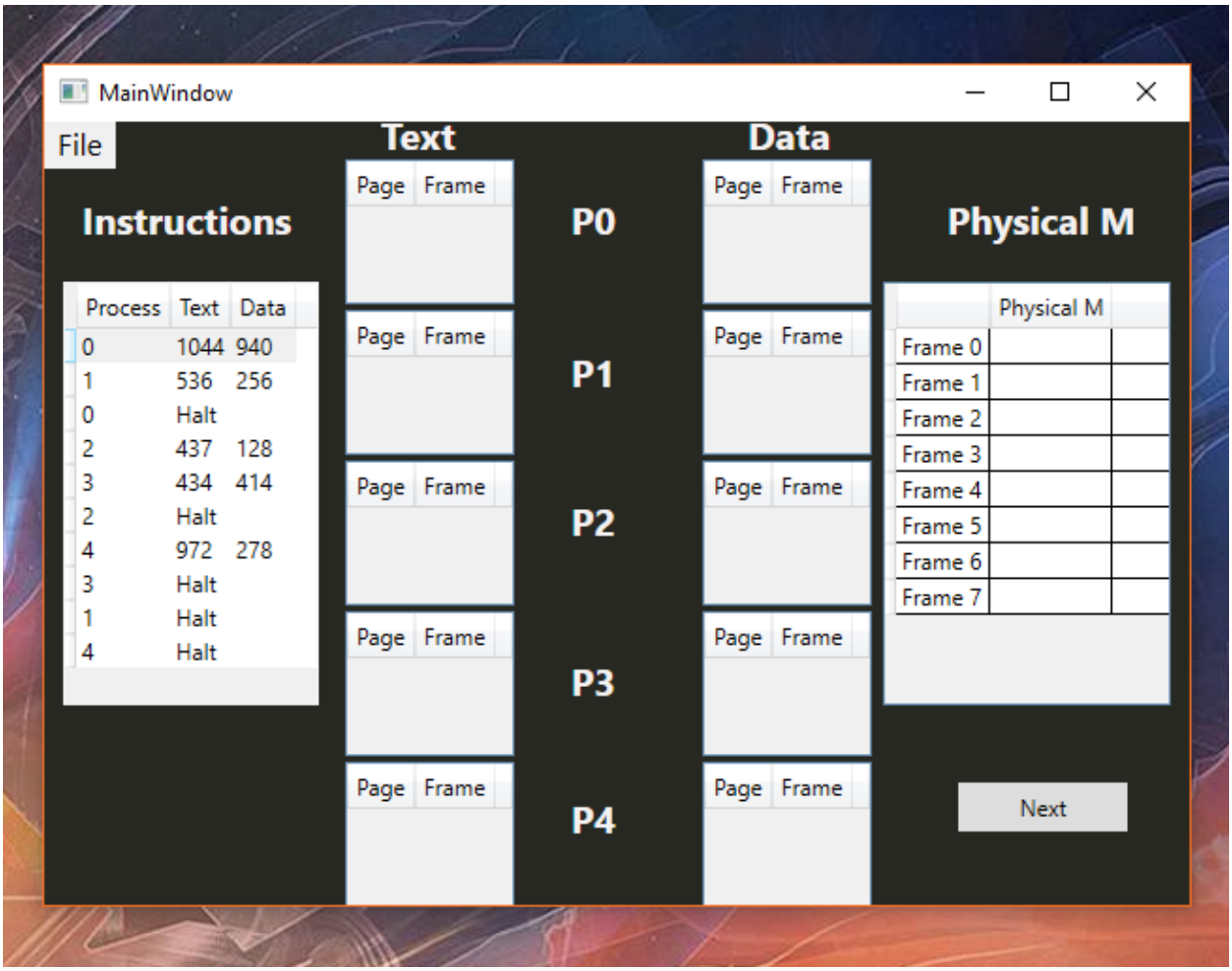
My program represents PCBs as classes. In Linux these are represented as structs, so my program is very similar to Linux in that regard. A little research showed that operating systems tend to organize in doubly linked lists. I tried to implement this at first, but I had a lot of problems getting it to work correctly, so I switched to a list. Each PCB holds information about the process, such as the number of text pages, the number of data pages, and pointers to the page table.

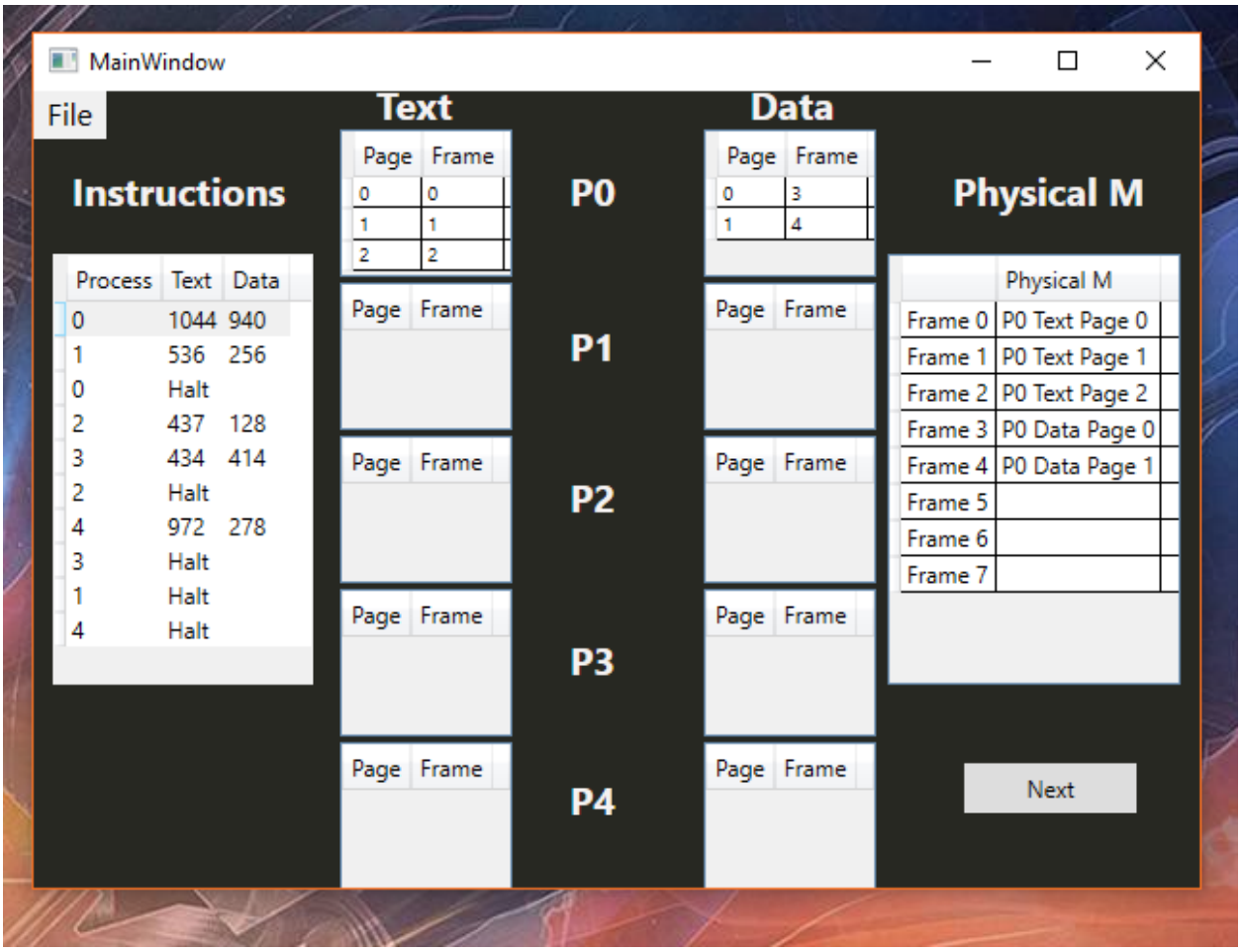
In implementing the page table, I tried to emulate a multi-level page table system. There is an outer page table, which is represented as a dictionary. The key to each entry is the process ID. Each key holds one inner page which is represented as a class and holds references to two page tables: the text table and the data table. I chose to use a dictionary for the outer table because it though it held some resemblance to how things work in practice. In a real operating system when we want to access a page table we give the OS a logical memory address and are presented with a physical address. In the same way we give the dictionary a key and are presented with a route to a physical address.

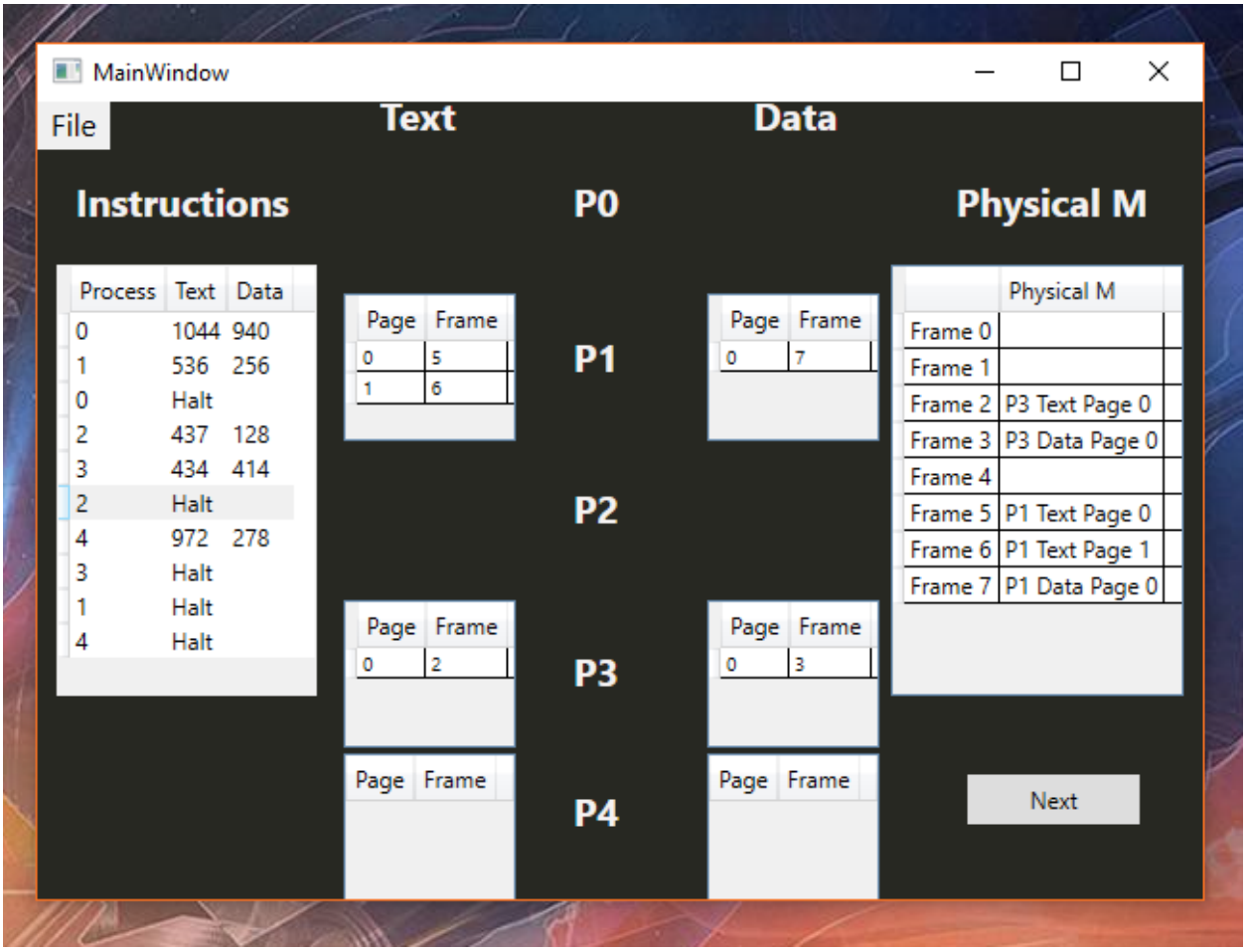
Regarding algorithms, all memory management is handled by the OS class. When an event occurs the OS first reads the process' "header" which is sent by the GUI and consists of a process ID, the size of the text segment, and the size of the data segment. Next the OS checks to see if the process already has a PCB entry. If it doesn't the OS goes to work loading the process into memory.

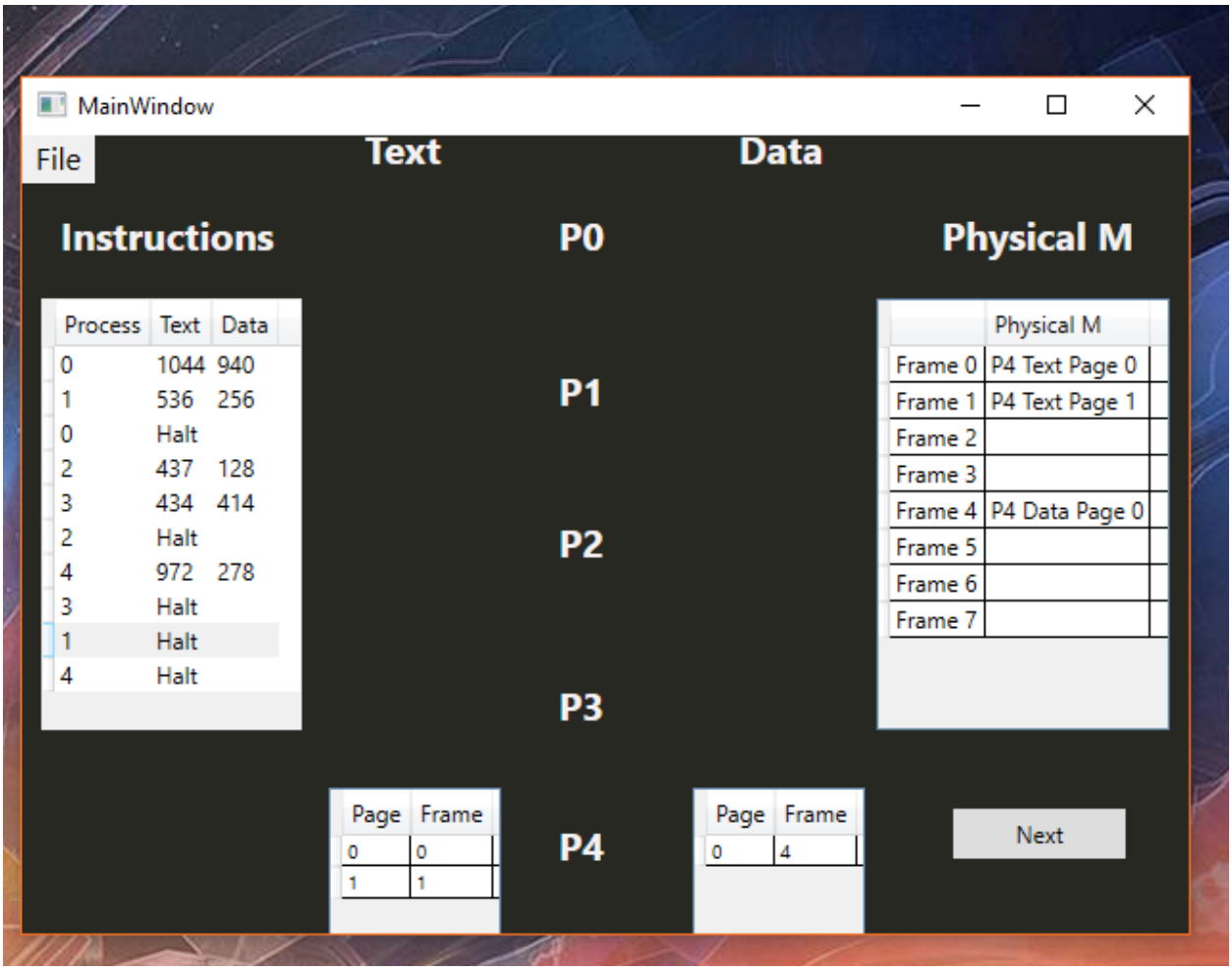
To accomplish loading a process into memory, the OS first calculates the number of pages needed for the text and data sections. It then loops through the frame table, which is represented by an array, and if it finds an empty frame it will mark the frame as used, and enter the frame number into the page table for that process. It repeats this process for the text segment, and then for the data segment. When the page tables are created it adds them to an inner page, then adds the inner page to the outer page.

If, on the other hand the OS finds that there is already a PCB for the process, it knows that the command must be to halt the process. To accomplish this it first gets a reference to the PCB for the process. It uses the information in the PCB to loop through each of the process' page tables and free the frames the pages are assigned to. Finally it removes the inner page table for that process from the outer page table, and also removes the process' PCB.


MainWindow
File
Instructions
Process Text Data
0 1044 940
1 536 256
0 Halt
2 437 128
3 434 414
2 Halt
4 972 278
3 Halt
1 Halt
4 Halt
Text
Page Frame
P0
P1
P2
P3
P4
Data
Page Frame
Physical M
Physical M
Frame 0
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7
Next


MainWindow
File
Instructions
Process Text Data
0 1044 940
1 536 256
0 Halt
2 437 128
3 434 414
2 Halt
4 972 278
3 Halt
1 Halt
4 Halt
Text
Page Frame
0 0
1 1
2 2
P0
P1
P2
P3
P4
Data
Page Frame
0 3
1 4
Physical M
Physical M
Frame 0 P0 Text Page 0
Frame 1 P0 Text Page 1
Frame 2 P0 Text Page 2
Frame 3 P0 Data Page 0
Frame 4 P0 Data Page 1
Frame 5
Frame 6
Frame 7
Next


MainWindow
File
Text
Data
Instructions
P0
Physical M
Process Text Data
0 1044 940
1 536 256
0 Halt
2 437 128
3 434 414
2 Halt
4 972 278
3 Halt
1 Halt
4 Halt
Page Frame
0 5
1 6
P1
Page Frame
0 7
P2
Page Frame
0 2
P3
Page Frame
0 3
Page Frame
P4
Page Frame
Physical M
Frame 0
Frame 1
Frame 2 P3 Text Page 0
Frame 3 P3 Data Page 0
Frame 4
Frame 5 P1 Text Page 0
Frame 6 P1 Text Page 1
Frame 7 P1 Data Page 0
Next



Process	Text	Data
0	1044	940
1	536	256
0	Halt	
2	437	128
3	434	414
2	Halt	
4	972	278
3	Halt	
1	Halt	
4	Halt	

	Physical M
Frame 0	P4 Text Page 0
Frame 1	P4 Text Page 1
Frame 2	
Frame 3	
Frame 4	P4 Data Page 0
Frame 5	
Frame 6	
Frame 7	

Page	Frame
0	0
1	1

Page	Frame
0	4