The program is split into 4 main parts:

1. Initializing Memory Map
2. Handling Process Arrivals
3. Handling Process Completions
4. Moving Processes into Memory

Initializing Memory Map

The way we did did the memory map is through a vector of objects, with each object representing being a block of space. Each object holds the first and last frame, the size of the block, whether the block should be deleted and whether the block is free. It also holds the associated process id and page number. Storing it this way makes it easy to use the memory map. Checking if there is any memory available is as easy as checking the size variable. Deallocating is as easy as setting the free variable to true. And outputting the memory map is as easy as outputting each variable.

The memory map is initialized from information in the input text file. The text file is read word by word. The first word, being the number of processes, is stored into the processcount variable. The loop will then start, going processcount times. In the loop, we go to the second word, which is the first process's id, so that is stored into a temporary object which will hold the processes information. The third word is the arrival time, then lifetime, then memorypieces, and finally the address space which is added by looping how ever many memorypieces there are times and adding each number along the way. All of this, of course, gets stored in the object which at the end of the first pass

will get send to a vector of processes. Once it is done, a temporary object will hold the first block the memory map. Being that the map is empty it will be free and the first frame is set to 0, the end will size - 1 of the total memory available and the size is the size of the total memory available. This object will then be sent to the vector holding memory blocks.

Handling Process Arrivals

To explain how the current time in the program is calculated and thus how we can detect when and have a process arrive, the timer of this program is set through a do while loop, incrementing the t (timer) variable once at the end each and every pass through. A process arrives by looping through the process list and simply checking if the current time is equal to the process arrival time. It it is and it is the first event of the current time, it will output the current time (by time we refer to program time), then it will state what process has arrived, will add to and show the input queue and finally the memory map. If another process arrives at the same time it will output everything above except the current time.

Handling Process Completions

After handling process arrivals the priority is handling processes that complete and thus freeing up memory for other processes. First, the process list is sorted by smallest life time. It will then loop through the process list, first making sure the process is in memory and then checking if the current time minus the process arrival time in

memory is equal to the process lifetime. If not, it continues looping trying to find a process to handle. If there is a process to handle and it is the first event of the current time, it will output the current time and say which process has completed. Otherwise it will just say which process has completed. It will then calculate the turnaround time which is the current time minus the process's own set arrival time. Next it will remove the process from memory. It does this by looping through the memory map and finding any blocks associated with the process and setting it to free. After that it will send the process id to the deletion queue. Finally, it will show the updated memory map and increment the process completions count. After looping another 2 loops will begin to delete the process(es) from the process list. It will loop through deletion queue, and in each pass through will do another loop through the process list finding the matching process id of the current element of the deletion queue, if it finds it, it will be removed from the process list.

Moving Processes to Memory

After any processes are freed from memory, the program will try and move the process(es) in the input queue to memory. It will loop through the input queue and for each process will try and store the entire process in memory. It does this first by getting the size of free space by looping through the memory map and checking if a block is free, it will add to the free space size. It will then check if the free space is greater than the process address space so that it can fit the entire process in. If so it will loop through

the memory map again checking again if a block is free each pass through. If it is free, it will go down one of three pathways.

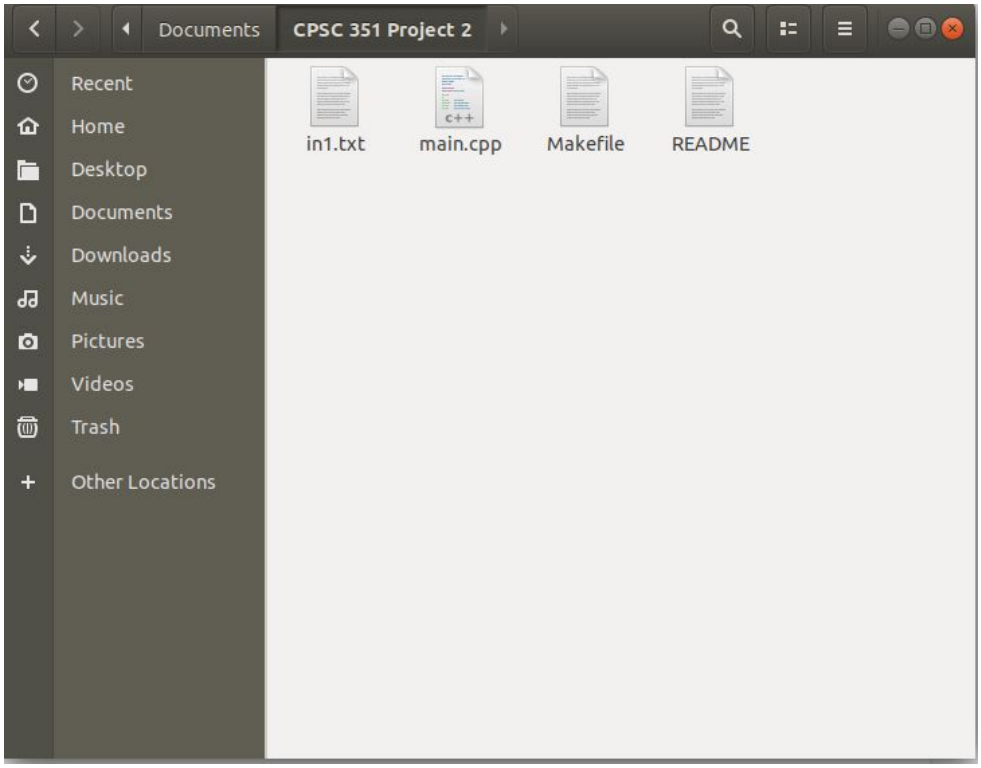
If the free block can fit the entire process, it will split the block into the required number of blocks. It does this by first calculating the number blocks are needed for the process to fit. It does this by taking the address space and dividing by the page size and taking the ceiling of it. Since each block size cannot be less than the page size, this will calculate how many page sized blocks are needed. It will then loop through the result of that times. A temporary object will be used to store each block. On first pass it will set the temporary block's start equal to the free block's start. On subsequent passes, the temporary block's start will be set to the end of the previous block plus 1. Then it will set the block's end to the block's start plus page size minus 1. It will then set the process id to the id of the current process needed to be allocated, the page number equal to the current value of loop plus 1 and finally setting the block to not be free. For the free block, it will first decrement the size by page size. If the size of the free block is not 0, it will move the start of the block to the end of the object's start plus 1. If it is 0, it will be set to be deleted.

If the free block can partially fit the process, it will split the block similarly as above but instead of looping the number of blocks the process needs, it will loop the number of page sized blocks in the free block which is calculated as the size of the block divided by page size. It will also break if the current page number is equal to the number of blocks the process needs.

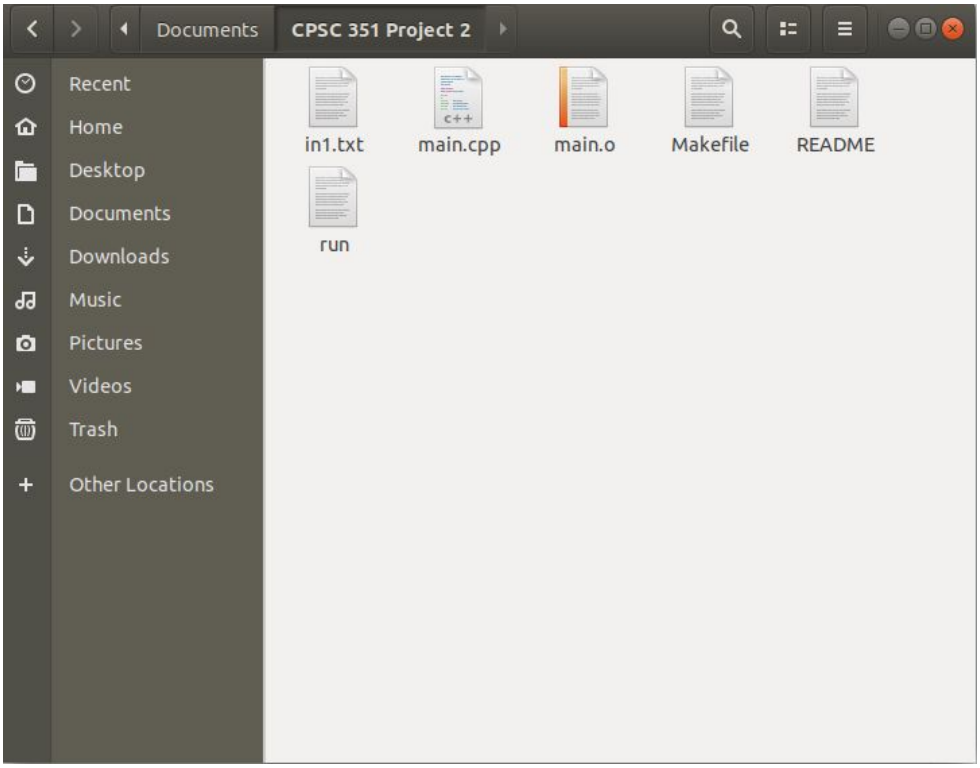
If the free block is only one page size, then the program basically replaces the block with the information of the process by setting the process id, the current page number and setting it to not be free. If the program was successfully able to allocate all memory, it will first delete any blocks similarly to how it deletes a process from the process list. It will then say that it has moved the process to memory, remove the process from the input queue, and will output the updated input queue. Finally it will record the time the process arrived in memory.

Handling process arrivals, process completions and moving processes to memory is all looped over and over until either the time reaches 100000 or all processes have completed. It will then update the average turnaround time.

Screenshot of initial folder:



Screenshot of folder after making:



Various screenshots of program running:

```
tavinc@indanailpail:~/Documents/CPSC 351 Project 2$ make
g++ -c main.cpp
g++ -o run main.o
tavinc@indanailpail:~/Documents/CPSC 351 Project 2$ ./run
Enter memory size: 2000
Enter page size (1:100, 2:200, 3:400): 1
Enter the name of the workload file (just the name): in1

t = 0: Process 1 arrives
       Input Queue: [ 1 ]
       Memory Map: 0-1999: Free frame(s)
       Process 2 arrives
       Input Queue: [ 1 2 ]
       Memory Map: 0-1999: Free frame(s)
       MM moved Process 1 to memory
       Input Queue: [ 2 ]
       Memory Map: 0-99: Process 1, Page 1
                100-199: Process 1, Page 2
                200-299: Process 1, Page 3
                300-399: Process 1, Page 4
                400-1999: Free frame(s)
       MM moved Process 2 to memory
       Input Queue: [ ]
       Memory Map: 0-99: Process 1, Page 1
```

```
       MM moved Process 2 to memory
       Input Queue: [ ]
       Memory Map: 0-99: Process 1, Page 1
                100-199: Process 1, Page 2
                200-299: Process 1, Page 3
                300-399: Process 1, Page 4
                400-499: Process 2, Page 1
                500-599: Process 2, Page 2
                600-699: Process 2, Page 3
                700-799: Process 2, Page 4
                800-899: Process 2, Page 5
                900-999: Process 2, Page 6
               1000-1999: Free frame(s)

t = 100: Process 3 arrives
       Input Queue: [ 3 ]
       Memory Map: 0-99: Process 1, Page 1
                100-199: Process 1, Page 2
                200-299: Process 1, Page 3
                300-399: Process 1, Page 4
                400-499: Process 2, Page 1
                500-599: Process 2, Page 2
                600-699: Process 2, Page 3
                700-799: Process 2, Page 4
```

```
      Process 4 arrives
      Input Queue: [ 3 4 ]
      Memory Map: 0-99: Process 1, Page 1
                  100-199: Process 1, Page 2
                  200-299: Process 1, Page 3
                  300-399: Process 1, Page 4
                  400-499: Process 2, Page 1
                  500-599: Process 2, Page 2
                  600-699: Process 2, Page 3
                  700-799: Process 2, Page 4
                  800-899: Process 2, Page 5
                  900-999: Process 2, Page 6
                 1000-1999: Free frame(s)
      MM moved Process 3 to memory
      Input Queue: [ 4 ]
      Memory Map: 0-99: Process 1, Page 1
                  100-199: Process 1, Page 2
                  200-299: Process 1, Page 3
                  300-399: Process 1, Page 4
                  400-499: Process 2, Page 1
                  500-599: Process 2, Page 2
                  600-699: Process 2, Page 3
                  700-799: Process 2, Page 4
                  800-899: Process 2, Page 5
```

```
      Input Queue: [ ]
      Memory Map: 0-99: Process 1, Page 1
                  100-199: Process 1, Page 2
                  200-299: Process 1, Page 3
                  300-399: Process 1, Page 4
                  400-499: Process 2, Page 1
                  500-599: Process 2, Page 2
                  600-699: Process 2, Page 3
                  700-799: Process 2, Page 4
                  800-899: Process 2, Page 5
                  900-999: Process 2, Page 6
                 1000-1099: Process 3, Page 1
                 1100-1199: Process 3, Page 2
                 1200-1299: Process 3, Page 3
                 1300-1399: Process 4, Page 1
                 1400-1499: Process 4, Page 2
                 1500-1999: Free frame(s)

t = 200: Process 5 arrives
      Input Queue: [ 5 ]
      Memory Map: 0-99: Process 1, Page 1
                  100-199: Process 1, Page 2
                  200-299: Process 1, Page 3
                  300-399: Process 1, Page 4
```

```
       MM moved Process 5 to memory
       Input Queue: [ ]
       Memory Map: 0-99: Process 1, Page 1
                 100-199: Process 1, Page 2
                 200-299: Process 1, Page 3
                 300-399: Process 1, Page 4
                 400-499: Process 2, Page 1
                 500-599: Process 2, Page 2
                 600-699: Process 2, Page 3
                 700-799: Process 2, Page 4
                 800-899: Process 2, Page 5
                 900-999: Process 2, Page 6
                1000-1099: Process 3, Page 1
                1100-1199: Process 3, Page 2
                1200-1299: Process 3, Page 3
                1300-1399: Process 4, Page 1
                1400-1499: Process 4, Page 2
                1500-1599: Process 5, Page 1
                1600-1699: Process 5, Page 2
                1700-1799: Process 5, Page 3
                1800-1899: Process 5, Page 4
                1900-1999: Process 5, Page 5

t = 1000: Process 5 completes
```

```
                1900-1999: Process 5, Page 5

t = 1000: Process 5 completes
       Memory Map: 0-99: Process 1, Page 1
                 100-199: Process 1, Page 2
                 200-299: Process 1, Page 3
                 300-399: Process 1, Page 4
                 400-499: Process 2, Page 1
                 500-599: Process 2, Page 2
                 600-699: Process 2, Page 3
                 700-799: Process 2, Page 4
                 800-899: Process 2, Page 5
                 900-999: Process 2, Page 6
                1000-1099: Process 3, Page 1
                1100-1199: Process 3, Page 2
                1200-1299: Process 3, Page 3
                1300-1399: Process 4, Page 1
                1400-1499: Process 4, Page 2
                1500-1599: Free frame(s)
                1600-1699: Free frame(s)
                1700-1799: Free frame(s)
                1800-1899: Free frame(s)
                1900-1999: Free frame(s)
       Process 3 completes
```

End of program screenshot: