The program works as follows: First two functions are used to remove the offset from the files. Then we use a for loop to create 3 processes. Each time fork is called to create a new process. If we are on the first loop we will do stuff for the first process, the second loop will do stuff for the second process and so on. PM1 process reads the address and R/W bit from bzip file and passes the information through a shared memory on to the third process. PM2 process does the same thing, but with the gcc file. The Memory Manager(MM) process read what a PM process wrote to the shared memory and then passes the information to the hash table.

The Hash Table class consists of an array of pointers to Page Table Entry objects. First, it takes the address and R/W bit and searches for them(through a hash function) in the hash array. If it couldn’t find them, then we have a page fault. When we reach k+1 page faults we delete every page(but before we delete the page we have to get whether it had an R or W bit). If we didn’t find the page inside the hash table then it is inserted in the hash array.
The Page Table Entry class simply stores the address, the R/W bit, the process id and a pointer to the next node.
After doing work for max traces, the program prints how many disk reads/writes we had, how many page faults occured, how many traces were examined and the number of covered frames.

A father process simply waits for his children to finish.

One shared memory was used. It stores the address, R/W bit and the process id.

Eleven semaphores were used. Two of them are used for the same memory. One Empty and one Full, based on the bounded buffer problem. A PM process may write only when the shared memory is empty. After it has finished writing it signals the reader process (MM) that may read the shared memory segment. After MM reads the segment and passes it to the hash table, it signals the shared memory is empty, so another process (or the same) may write again. Empty1 and Full1 were created so they could be used from the PM2 process.
Wait and Wait1 semaphores were used because the MM process has to wait for a PM process to write something to the shared memory before it can read again.
Proc2 semaphore is used in the beginning so the PM2 process is initially blocked. Only after PM1 process sends its first q traces, the PM2 begins its work.
After a PM process sends its q traces Q1 and Q2 semaphores are used in order to block the said process. For example, after PM1 has sent q traces it is blocked so PM2 may send its q traces, and so on. These semaphores are incremented by the MM process after it has sent q traces to the hash table.
Print1 and Print2 semaphores were used so the program will print the necessary information after all processes have finished their work.

To compile and run the program type:

```
make
./main W X Y Z (W = k, X = frames, Y = q, Z = max)
```

Here are different results for different values of k:

1.) Output for k between 1 and 9 (The other arguments are: frames=2048, q=10 and max=50000) is :

Total Disk Reads are: 13

Total Disk Writes are: 1

Total Page Faults are: 215

Max number of traces examined by each file: 50000

Total Covered Frames are: 201

2.) Output for k between 10 and 99 (The other arguments are: frames=2048, q=10 and max=50000) is :

Total Disk Reads are: 94

Total Disk Writes are: 44

Total Page Faults are: 330

Max number of traces examined by each file: 50000

Total Covered Frames are: 192

3.) Output for k over 100 (The other arguments are: frames=2048, q=10 and max=50000) is :

Total Disk Reads are: 49

Total Disk Writes are: 52

Total Page Faults are: 281

Max number of traces examined by each file: 50000

Total Covered Frames are: 180