Brad Slater
Ashley Smith
Dr. Mikler
CSCE 4600 - Operating Systems
November 3, 2014

Project 1 - Scheduler Simulation

For this project, we developed a program that simulates a scheduler by assigning processes to a set of processes. The program is written entirely in C and can be compiled using the GNU compiler - the gcc command. Each of the four problems for this project will be assessed and shown screenshots of our code with details of the implementation. Screenshots of the results for each problem will also be given. At the very end of the report, a screenshot of the entire output of the program (executed once) is shown.

Given in the assignment instructions, we are simulating five processors and fifty processes. Each process has different runtime requirements - specifically, a burst time for the process to complete and an amount of memory that the process contains. We assigned the burst times at random between the values of $10 * 10^6$ cycles and $50 * 10^{12}$ cycles. The memory for each process was also randomly assigned values between .25 megabytes and 8 gigabytes.

Seen below is the struct in which we used to hold the data for each process:

```
5  typedef struct {
6      unsigned long burst;
7      unsigned int memory;
8  } process;
```

The struct is type defined with the name “process” and contains two properties: an unsigned long integer “burst” which contains the burst time and an unsigned integer “memory” which contains the memory.

Seen below is the struct in which we used to hold the data for each central processing unit (CPU):

```
10 typedef struct {
11     unsigned long speed;
12     unsigned int memory;
13     process queue[50];
14     int procCount;
15 } CPU;
```

This struct is type defined with the name “CPU” and contains four properties: an unsigned long integer “speed” which contains the cpu speed in hertz, an unsigned integer “memory” which simulates the RAM of the machine, an array of fifty processes called “queue” which is used to store the processes in simulation, and an integer “procCount” which is used to determine how many processes are currently in a cpu’s “queue”.

Since the scheduling algorithm known as “shortest job first” (SJF) is provably optimal - as seen in class - we used this as the solution for each problem. The implementation is modified slightly to cater to each limitation and situation, but the basis of the algorithm remains unchanged. It should be noted that before each problem set, the simulated CPU’s are cleared of all their properties (speed, memory, queue, and processor count). Also, the turnaround time was calculated by adding up the burst times of each processor, taking the largest sum, and dividing by the speed of that processor. The function that returns the turnaround time can be seen on page 4. That returned value gets divided by the speed property of the corresponding cpu.

Problem 1:

For the first problem, it is a given limitation that the processors are identical. This means that the processors contain the same amount of memory and have the same speed. As stated, we implemented a SJF algorithm as the best solution. This is implemented in a function called “prob1” which also contains the assignment of the CPUs and getting the turnaround time for the entire set of processes.

```
173         clearCpus();
174         cpus[0].speed = 3 * (long)(pow(10,6));
175         cpus[0].memory = 8388608; // 8GB
176         cpus[0].procCount = 0;
177         cpus[1].speed = 3 * (long)(pow(10,6));
178         cpus[1].memory = 8388608; // 8GB
179         cpus[1].procCount = 0;
180         cpus[2].speed = 3 * (long)(pow(10,6));
181         cpus[2].memory = 8388608; // 8GB
182         cpus[2].procCount = 0;
183         cpus[3].speed = 3 * (long)(pow(10,6));
184         cpus[3].memory = 8388608; // 8GB
185         cpus[3].procCount = 0;
186         cpus[4].speed = 3 * (long)(pow(10,6));
187         cpus[4].memory = 8388608; // 8GB
188         cpus[4].procCount = 0;
```

Seen above, each of the five cpu’s are assigned the same speed of 3 gigahertz and 8 gigabytes of memory. The processor count is also assigned to zero for each cpu.

On the next page is the implementation of the SJF algorithm. For each process, a call to “getShortestQueueByTime” is called (shown below) which returns the processor that contains the shortest amount of execution time for all of its processes. From here, it adds the current process to

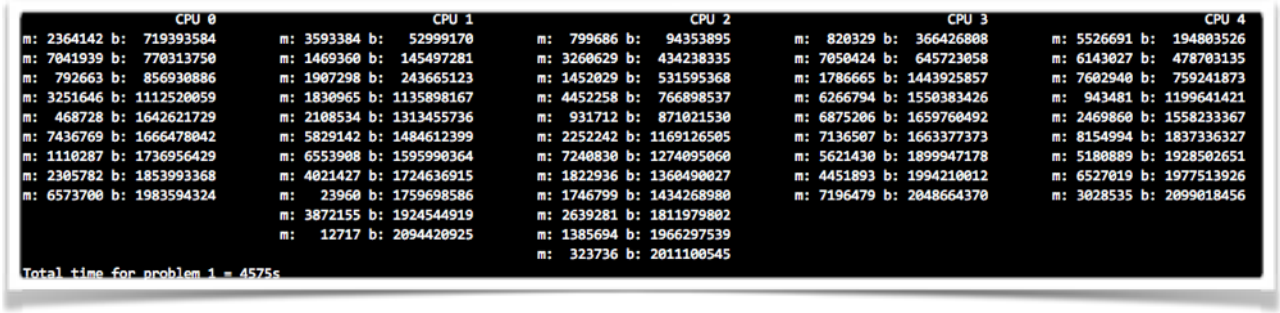
that cpu's queue, increments the processor count for that cpu, and then reorders the queue by shortest job first using a simple selection sort algorithm.

```
190     // SJF
191     int i, queuePos = 0;
192     for (i = 0; i < 50; i++) {
193         // Get the shortest queue
194         int min = getShortestQueueByTime();
195
196         // add process to queue
197         cpus[min].queue[cpus[min].procCount] = processes[i];
198         cpus[min].procCount++;
199         reorderQueue(cpus[min].queue, cpus[min].procCount);
200
201     }
```

In more depth of this SJF algorithm, we can see below how the shortest queue is returned by the functions "getShortestQueueByTime" and "turnaroundTimeForCpu". The total execution time of all the processes for a processor is computed, and then compared to the previous iteration that was calculated. Once the smallest is found, it is returned and stored in the variable "min".

```
149 unsigned long long turnaroundTimeForCpu(int cpu) {
150     unsigned long long sum = 0;
151     int i;
152     for (i = 0; i < cpus[cpu].procCount; i++) {
153         sum += cpus[cpu].queue[i].burst;
154     }
155
156     return sum;
157 }
158
159 // Gets the CPU with the shortest turnaround time by cycles
160 // Only useful if all CPU's are the same speed
161 int getShortestQueueByTime() {
162     unsigned long long sum = 0;
163     int i, min = 0;
164     for (i = 0; i < 5; i++) {
165         if (turnaroundTimeForCpu(i) < turnaroundTimeForCpu(min))
166             min = i;
167     }
168
169     return min;
170 }
```

A final call to get the turnaround time, print the processes of each cpu, and printing the turnaround time results in below:



```
                CPU 0                    CPU 1                    CPU 2                    CPU 3                    CPU 4
m: 2364142 b:  719393584 m: 3593384 b:   52999170 m:  799686 b:   94353895 m:  820329 b:  366426808 m: 5526691 b:  194803526
m: 7041939 b:  770313750 m: 1469360 b:  145497281 m: 3260629 b:  434238335 m: 7050424 b:  645723058 m: 6143027 b:  478703135
m:  792663 b:  856930886 m: 1907298 b:  243665123 m: 1452029 b:  531595368 m: 1786665 b: 1443925857 m: 7602940 b:  759241873
m: 3251646 b: 1112520059 m: 1830965 b: 1135898167 m: 4452258 b:  766898537 m: 6266794 b: 1550383426 m:  943481 b: 1199641421
m:  468728 b: 1642621729 m: 2108534 b: 1313455736 m:  931712 b:  871021530 m: 6875206 b: 1659760492 m: 2469860 b: 1558233367
m: 7436769 b: 1666478042 m: 5829142 b: 1484612399 m: 2252242 b: 1169126505 m: 7136507 b: 1663377373 m: 8154994 b: 1837336327
m: 1110287 b: 1736956429 m: 6553908 b: 1595990364 m: 7240830 b: 1274095060 m: 5621430 b: 1899947178 m: 5180889 b: 1928502651
m: 2305782 b: 1853993368 m: 4021427 b: 1724636915 m: 1822936 b: 1360490027 m: 4451893 b: 1994210012 m: 6527019 b: 1977513926
m: 6573700 b: 1983594324 m:   23960 b: 1759698586 m: 1746799 b: 1434268980 m: 7196479 b: 2048664370 m: 3028535 b: 2099018456
                         m: 3872155 b: 1924544919 m: 2639281 b: 1811979802
                         m:   12717 b: 2094420925 m: 1385694 b: 1966297539
                                                  m:  323736 b: 2011100545

Total time for problem 1 = 4575s
```

It shows the memory and burst time of each process in each cpu. The total simulated time is also shown (in seconds) underneath the printed output. This is actual output from a shell. In order to get the turnaround time of all the cpu's, the function "getTurnaroundTime" can be seen below. This function iterates through each cpu and then iterates through each process of the current cpu - adding up the burst times of each process along the way. By doing so, we calculate which cpu has the longest execution time. Since they are theoretically executing in parallel, we only need the cpu time that is running the longest. With this time, all the other processors have completed execution, meaning this is the execution time of all processes.

```
134 unsigned long long getTurnaroundTime() {
135     unsigned long long turnaround = 0, sum = 0;
136     int i, j;
137     for (i = 0; i < 5; i++) {
138         sum = 0;
139         for (j = 0; j < cpus[i].procCount; j++) { // 10 process per CPU
140             sum += cpus[i].queue[j].burst;
141         }
142         if (sum > turnaround)
143             turnaround = sum;
144     }
145
146     return turnaround;
147 }
```

Problem 2:

For this problem, the limitations on the processors are changed to having the same speed but different memory availabilities. The first two cpu's have two gigabytes of memory, the third and fourth cpu's have four gigabytes of memory, and the last cpu has eight gigabytes of memory. Since each cpu has the same speed, the only parts that change in the assignment statements from Problem 1 are the memory assignments. Two gigabytes equals 2,097,152 kilobytes, four gigabytes equals 4,194,304 kilobytes, and the eight gigabytes stays the same as Problem 1.

For the implementation of SJF in this problem, each process is assigned to a corresponding cpu based on the process's memory count. This can be seen below:

```
    int i, cpuI;
    for (i = 0; i < 50; i++) {
        if (processes[i].memory <= 2097152) {
            cpuI = (turnaroundTimeForCpu(0) < turnaroundTimeForCpu(1)) ? 0 : 1;
            cpus[cpuI].queue[cpus[cpuI].procCount] = processes[i];
            cpus[cpuI].procCount++;
            reorderQueue(cpus[cpuI].queue, cpus[cpuI].procCount);
        } else if (processes[i].memory <= 4194304) {
            cpuI = (turnaroundTimeForCpu(2) < turnaroundTimeForCpu(3)) ? 2 : 3;
            cpus[cpuI].queue[cpus[cpuI].procCount] = processes[i];
            cpus[cpuI].procCount++;
            reorderQueue(cpus[cpuI].queue, cpus[cpuI].procCount);
        } else if (processes[i].memory <= 8388608) {
            cpus[4].queue[cpus[4].procCount] = processes[i];
            cpus[4].procCount++;
            reorderQueue(cpus[4].queue, cpus[4].procCount);
        }
    }
```

Based on these new memory restrictions, the process is placed in the cpu queue that has the lower turnaround time - seen by the ternary operations below each if statement. Again, after the process is added to the correct cpu queue, that queue gets reordered to reflect the shortest job first.

Shown below is a screenshot of the output of Problem 2. An issue we encountered with this method is that the C language has a limitation on its "random" implementation. Even seeding random with time as we did, it was consistently making processes with memory requirements closer to 8GB. This caused a shift in the data to be skewed towards the last cpu. Either way, the turnaround time for Problem 2 was about twice as long as Problem 1.

```
                CPU 0                       CPU 1                       CPU 2                       CPU 3                       CPU 4
m:  820329 b:  366426808    m:  799686 b:   94353895    m: 3593384 b:   52999170    m: 2469860 b: 1558233367    m: 5526691 b:  194803526
m: 1452029 b:  531595368    m: 1469360 b:  145497281    m: 3260629 b:  434238335    m: 4021427 b: 1724636915    m: 6143027 b:  478703135
m:  943481 b: 1199641421    m: 1907298 b:  243665123    m: 2364142 b:  719393584    m: 2639281 b: 1811979802    m: 7050424 b:  645723058
m: 1746799 b: 1434268980    m:  792663 b:  856930886    m: 3251646 b: 1112520059    m: 3872155 b: 1924544919    m: 7602940 b:  759241873
m: 1786665 b: 1443925857    m:  931712 b:  871021530    m: 2252242 b: 1169126505                                m: 4452258 b:  766898537
m:  468728 b: 1642621729    m: 1830965 b: 1135898167    m: 2108534 b: 1313455736                                m: 7041939 b:  770313750
m: 1110287 b: 1736956429    m: 1822936 b: 1360490027    m: 2305782 b: 1853993368                                m: 7240830 b: 1274095060
m:   23960 b: 1759698586    m: 1385694 b: 1966297539    m: 3028535 b: 2099018456                                m: 5829142 b: 1484612399
                            m:  323736 b: 2011100545                                                            m: 6266794 b: 1550383426
                            m:   12717 b: 2094420925                                                            m: 6553908 b: 1595990364
                                                                                                                m: 6875206 b: 1659760492
                                                                                                                m: 7136507 b: 1663377373
                                                                                                                m: 7436769 b: 1666478042
                                                                                                                m: 8154994 b: 1837336327
                                                                                                                m: 5621430 b: 1899947178
                                                                                                                m: 5180889 b: 1928502651
                                                                                                                m: 6527019 b: 1977513926
                                                                                                                m: 6573700 b: 1983594324
                                                                                                                m: 4451893 b: 1994210012
                                                                                                                m: 7196479 b: 2048664370
Total time for problem 2 = 9393s
```

Problem 3:

For this problem, the processors contain the same amount of memory, however, the speeds differ. The memory requirement is set at eight gigabytes for each cpu and the speed is set as follows: the first two have a speed of two gigahertz, the third and fourth at three gigahertz, and the last cpu at four

```
271     unsigned long thirdOfBurst = maxBurst/3; // 1/3 of longest runtime
272     int i, j, cpuI;
273     for (i = 0; i < 50; i++) {
274         if (processes[i].burst <= thirdOfBurst) {
275             cpuI = (turnaroundTimeForCpu(0) < turnaroundTimeForCpu(1)) ? 0 : 1;
276             cpus[cpuI].queue[cpus[cpuI].procCount] = processes[i];
277             cpus[cpuI].procCount++;
278             reorderQueue(cpus[cpuI].queue, cpus[cpuI].procCount);
279         } else if (processes[i].burst <= thirdOfBurst*2) {
280             cpuI = (turnaroundTimeForCpu(2) < turnaroundTimeForCpu(3)) ? 2 : 3;
281             cpus[cpuI].queue[cpus[cpuI].procCount] = processes[i];
282             cpus[cpuI].procCount++;
283             reorderQueue(cpus[cpuI].queue, cpus[cpuI].procCount);
284         } else if (processes[i].burst > thirdOfBurst*2) {
285             cpus[4].queue[cpus[4].procCount] = processes[i];
286             cpus[4].procCount++;
287             reorderQueue(cpus[4].queue, cpus[4].procCount);
288         } else {
289             printf("*******Someone is lost********\n");
290             printf("maxBurst: %lu p[%d].b: %lu\n", maxBurst, i, processes[i].burst);
291         }
292     }
```

gigahertz. Jumping right into the implementation, because the speed of the processors now differs, instead of splitting the processes up depending on their memory size, we now separate them by their burst times. Using a simple variable to distinguish 1/3 of the longest burst time, the processes can be split to their corresponding cpu's. This is done in the if statements just as in Problem 2. This also allows the processes with shorter burst times to be executed on the slower processors.

From here, we implemented a rebalancing algorithm to keep the data from being skewed as it was in Problem 2. By taking the cpu's with the largest and smallest queue, we can balance the two by moving some of the processes from the larger queue to the smaller queue. This can be seen in the image to the right.

```
294     // balancing
295     // Find largest queue
296     int maxQueue = 0, minQueue = 0;
297     for (i = 0; i < 5; i++) {
298         if (cpus[i].procCount > cpus[maxQueue].procCount)
299             maxQueue = i;
300
301         if (cpus[i].procCount < cpus[minQueue].procCount)
302             minQueue = i;
303     }
304
305     // get # to re-schedule
306     int n = cpus[maxQueue].procCount - cpus[minQueue].procCount;
307
308     // reschedule
309     for (i = 0; i < n; i++) {
310         // get process
311         process p;
312         p.burst = cpus[maxQueue].queue[cpus[maxQueue].procCount-1].burst;
313         p.memory = cpus[maxQueue].queue[cpus[maxQueue].procCount-1].memory;
314         cpus[maxQueue].procCount--;
315
316         // place on shortest queue
317         cpus[minQueue].queue[cpus[minQueue].procCount] = p;
318         cpus[minQueue].procCount++;
319         reorderQueue(cpus[minQueue].queue, cpus[minQueue].procCount);
320
321         // Find new shortest queue
322         for (j = 0; j < 5; j++)
323             if (cpus[j].procCount < cpus[minQueue].procCount)
324                 minQueue = j;
325     }
```

After this, the turnaround time for each cpu is calculated the same as before but now with respect to each processor speed. Since they are not the same, it cannot be calculated one time as before. The results can be seen below. The time it took for all processes to complete was about 40% more than Problem 1 but about 1/3 less than Problem 2.

```
                CPU 0                       CPU 1                       CPU 2                       CPU 3                       CPU 4
m: 3593384 b:   52999170       m:  799686 b:   94353895       m: 2364142 b:  719393584       m: 7602940 b:  759241873       m: 1746799 b: 1434268980
m: 1469360 b:  145497281       m: 1907298 b:  243665123       m:  931712 b:  871021530       m: 4452258 b:  766898537       m: 1786665 b: 1443925857
m: 5526691 b:  194803526       m: 3260629 b:  434238335       m: 3251646 b: 1112520059       m: 7041939 b:  770313750       m: 5829142 b: 1484612399
m:  820329 b:  366426808       m: 7050424 b:  645723058       m: 1830965 b: 1135898167       m:  792663 b:  856930886       m: 6266794 b: 1550383426
m: 6143027 b:  478703135       m: 1110287 b: 1736956429       m: 2252242 b: 1169126505       m: 7240830 b: 1274095060       m: 2469860 b: 1558233367
m: 1452029 b:  531595368       m: 2305782 b: 1853993368       m:  943481 b: 1199641421       m: 2108534 b: 1313455736       m: 6553908 b: 1595990364
m:   23960 b: 1759698586       m: 1385694 b: 1966297539       m: 2639281 b: 1811979802       m: 1822936 b: 1360490027       m:  468728 b: 1642621729
m: 5621430 b: 1899947178       m: 7196479 b: 2048664370       m: 8154994 b: 1837336327       m: 7436769 b: 1666478042       m: 6875206 b: 1659760492
m: 6527019 b: 1977513926       m:   12717 b: 2094420925       m: 6573700 b: 1983594324       m: 4021427 b: 1724636915       m: 7136507 b: 1663377373
m:  323736 b: 2011100545       m: 3028535 b: 2099018456       m: 4451893 b: 1994210012       m: 3872155 b: 1924544919
                                                              m: 5180889 b: 1928502651

Total time for problem 3 = 6608s
```

Problem 4:

For the final problem, the new limitation is that we cannot compare the processes but must handle them as they are created. So, for this reason, instead of looping through the array of processes as we had previously done for Problems 1-3, we created a new process fifty times. The “getNewProcess” function can be seen below. However, this is the only change from Problem 1. Everything else is exactly the same. This resulted in a lower theoretical runtime, but again, each process was created in the loop, resulting in a completely different set of processes. The result of this Problem 4 solution can be seen on the next page, along with the other solutions previously posted.

```
43  process getNewProcess() {
44      process p;
45      p.memory = (random() % (8388608 - 250)) + 250; // .25MB - 8GB
46      p.burst = (random() % (long)((50*pow(10,12)) - (10*pow(10,6)))) + (10*pow(10,6));
47      return p;
48  }
```

```
363     while (i < 50) {
364         process p = getNewProcess();
365
366         // Get the shortest queue
367         int min = getShortestQueueByTime();
368
369         // add process to queue
370         cpus[min].queue[cpus[min].procCount] = p;
371         cpus[min].procCount++;
372         reorderQueue(cpus[min].queue, cpus[min].procCount);
373
374         i++;
375     }
```

```
                CPU 0                        CPU 1                        CPU 2                        CPU 3                        CPU 4
m: 2364142 b:  719393584       m: 3593384 b:   52999170       m:  799686 b:   94353895       m:  820329 b:  366426808       m: 5526691 b:  194803526
m: 7041939 b:  770313750       m: 1469360 b:  145497281       m: 3260629 b:  434238335       m: 7050424 b:  645723058       m: 6143027 b:  478703135
m:  792663 b:  856930886       m: 1907298 b:  243665123       m: 1452029 b:  531595368       m: 1786665 b: 1443925857       m: 7602940 b:  759241873
m: 3251646 b: 1112520059       m: 1830965 b: 1135898167       m: 4452258 b:  766898537       m: 6266794 b: 1550383426       m:  943481 b: 1199641421
m:  468728 b: 1642621729       m: 2108534 b: 1313455736       m:  931712 b:  871021530       m: 6875206 b: 1659760492       m: 2469860 b: 1558233367
m: 7436769 b: 1666478042       m: 5829142 b: 1484612399       m: 2252242 b: 1169126505       m: 7136507 b: 1663377373       m: 8154994 b: 1837336327
m: 1110287 b: 1736956429       m: 6553908 b: 1595990364       m: 7240830 b: 1274095060       m: 5621430 b: 1899947178       m: 5180889 b: 1928502651
m: 2305782 b: 1853993368       m: 4021427 b: 1724636915       m: 1822936 b: 1360490027       m: 4451893 b: 1994210012       m: 6527019 b: 1977513926
m: 6573700 b: 1983594324       m:   23960 b: 1759698586       m: 1746799 b: 1434268980       m: 7196479 b: 2048664370       m: 3028535 b: 2099018456
                               m: 3872155 b: 1924544919       m: 2639281 b: 1811979802
                               m:   12717 b: 2094420925       m: 1385694 b: 1966297539
                                                              m:  323736 b: 2011100545

Total time for problem 1 = 4575s
                CPU 0                        CPU 1                        CPU 2                        CPU 3                        CPU 4
m:  820329 b:  366426808       m:  799686 b:   94353895       m: 3593384 b:   52999170       m: 2469860 b: 1558233367       m: 5526691 b:  194803526
m: 1452029 b:  531595368       m: 1469360 b:  145497281       m: 3260629 b:  434238335       m: 4021427 b: 1724636915       m: 6143027 b:  478703135
m:  943481 b: 1199641421       m: 1907298 b:  243665123       m: 2364142 b:  719393584       m: 2639281 b: 1811979802       m: 7050424 b:  645723058
m: 1746799 b: 1434268980       m:  792663 b:  856930886       m: 3251646 b: 1112520059       m: 3872155 b: 1924544919       m: 7602940 b:  759241873
m: 1786665 b: 1443925857       m:  931712 b:  871021530       m: 2252242 b: 1169126505                                      m: 4452258 b:  766898537
m:  468728 b: 1642621729       m: 1830965 b: 1135898167       m: 2108534 b: 1313455736                                      m: 7041939 b:  770313750
m: 1110287 b: 1736956429       m: 1822936 b: 1360490027       m: 2305782 b: 1853993368                                      m: 7240830 b: 1274095060
m:   23960 b: 1759698586       m: 1385694 b: 1966297539       m: 3028535 b: 2099018456                                      m: 5829142 b: 1484612399
                               m:  323736 b: 2011100545                                                                     m: 6266794 b: 1550383426
                               m:   12717 b: 2094420925                                                                     m: 6553908 b: 1595990364
                                                                                                                            m: 6875206 b: 1659760492
                                                                                                                            m: 7136507 b: 1663377373
                                                                                                                            m: 7436769 b: 1666478042
                                                                                                                            m: 8154994 b: 1837336327
                                                                                                                            m: 5621430 b: 1899947178
                                                                                                                            m: 5180889 b: 1928502651
                                                                                                                            m: 6527019 b: 1977513926
                                                                                                                            m: 6573700 b: 1983594324
                                                                                                                            m: 4451893 b: 1994210012
                                                                                                                            m: 7196479 b: 2048664370
Total time for problem 2 = 9393s
                CPU 0                        CPU 1                        CPU 2                        CPU 3                        CPU 4
m: 3593384 b:   52999170       m:  799686 b:   94353895       m: 2364142 b:  719393584       m: 7602940 b:  759241873       m: 1746799 b: 1434268980
m: 1469360 b:  145497281       m: 1907298 b:  243665123       m:  931712 b:  871021530       m: 4452258 b:  766898537       m: 1786665 b: 1443925857
m: 5526691 b:  194803526       m: 3260629 b:  434238335       m: 3251646 b: 1112520059       m: 7041939 b:  770313750       m: 5829142 b: 1484612399
m:  820329 b:  366426808       m: 7050424 b:  645723058       m: 1830965 b: 1135898167       m:  792663 b:  856930886       m: 6266794 b: 1550383426
m: 6143027 b:  478703135       m: 1110287 b: 1736956429       m: 2252242 b: 1169126505       m: 7240830 b: 1274095060       m: 2469860 b: 1558233367
m: 1452029 b:  531595368       m: 2305782 b: 1853993368       m:  943481 b: 1199641421       m: 2108534 b: 1313455736       m: 6553908 b: 1595990364
m:   23960 b: 1759698586       m: 1385694 b: 1966297539       m: 2639281 b: 1811979802       m: 1822936 b: 1360490027       m:  468728 b: 1642621729
m: 5621430 b: 1899947178       m: 7196479 b: 2048664370       m: 8154994 b: 1837336327       m: 7436769 b: 1666478042       m: 6875206 b: 1659760492
m: 6527019 b: 1977513926       m:   12717 b: 2094420925       m: 6573700 b: 1983594324       m: 4021427 b: 1724636915       m: 7136507 b: 1663377373
m:  323736 b: 2011100545       m: 3028535 b: 2099018456       m: 4451893 b: 1994210012       m: 3872155 b: 1924544919
                                                              m: 5180889 b: 1928502651

Total time for problem 3 = 6608s
                CPU 0                        CPU 1                        CPU 2                        CPU 3                        CPU 4
m: 1491254 b:  178002245       m: 7008177 b:  279441500       m: 7254898 b:  165324914       m: 3461382 b:  280744729       m: 3725994 b:  126087764
m: 7815812 b:  299700723       m: 2489360 b:  348888228       m: 8190875 b:  223975407       m: 7521010 b:  449493451       m:  343577 b:  397346491
m: 4373011 b:  473480570       m: 5314405 b:  534872353       m:   50443 b:  448792350       m:   83352 b:  603209441       m: 3724739 b:  495560280
m:  933643 b:  525530019       m: 1872081 b:  832890675       m: 6399799 b:  851148365       m: 1019635 b:  937612902       m: 2577967 b:  508777856
m: 1866944 b:  670260756       m: 3732107 b: 1360573793       m: 3757328 b: 1130048829       m: 7811104 b: 1196452551       m: 5213003 b:  613570492
m:  108874 b: 1582276965       m: 2033488 b: 1583363368       m: 7473394 b: 1636276121       m: 6492361 b: 1632597488       m: 7944581 b:  801698927
m:  570846 b: 1615894428       m:   95433 b: 1610028624       m: 5924236 b: 1641518149       m: 7025173 b: 1643108117       m: 3807279 b:  904429689
m: 1066927 b: 1770243555       m: 4732344 b: 1794639529       m: 6728471 b: 1902066601       m: 5947653 b: 1894661237       m: 2839703 b:  981899228
m: 6222695 b: 1770281936       m: 2280690 b: 1985960378       m: 2646139 b: 1997231011       m:  672976 b: 2050332871       m: 1028100 b: 1713964683
m: 3529300 b: 1957346619       m:  548879 b: 1992275856       m:  872131 b: 2017905771                                      m:  715814 b: 2087486715
                                                                                                                            m: 7565542 b: 2140794395

Total time for problem 4 = 4107s
Kyles-Mac-Mini-2:4600Simulation Kyle$
```