Report of Assignment 3 - Operating Systems

Problem 1(Reader Writer)

1. The question stated that the program should take buffer size and number of readers as input.
2. There would be one writer which write to the buffers.
3. A reader should not be able to access the buffer without the writer writing anything to them.
4. A reader should not be able to access the buffer once it already ready it until it is overwritten by the writer.
5. A writer should be able to write to the buffer only if all the readers accessed it.
6. I used locks to lock the buffers until a reader finishes reading it to prevent it from others reading the buffer and causing conflicts.
7. I used semaphores to implement the locks which are initialised to 1 using the sem_init() function.
8. I created n threads using the pthread_create() function.
9. I created 1 writer thread as stated.
10. I used a 1D mutex array for implementing the locks.
11. I used 1 separate 1D arrays to keep the count of number of reads and writes.

Problem 3(Merge Sort)

1. The question stated that we need to modify the normal recursive merge sort into a merge sort using fork() calls.
2. i.e we need to call the recursive merge sort using 2 separate fork() calls i.e
3. pid_t pid1,pid2;
   if(pid1==0)
   mergesort(low to mid);

```
else
        waitpid(pid1,&status,0);
if(pid2==0)
        mergesort(mid+1 to high)
else
        waitpid(pid2,&status,0);
```

4. This kind of implementation did not go well as many forks are created.
5. Forking is a very slow process as it needs to copy all the registers and variables and then continue the process.
6. But as we go in increasing the input size this type of merge sort will become worse as it would create forks recursively and becomes a lot slower compared to normal merge sort.
7. This did not work for 100000,1000000 as it was not able to fork() these many times.

for n=1000

	Iterative Merge	Recursive Merge	Concurrent Merge
CPU Utilised	0.800	0.700	0.829
Context Switches	7	6	2010
Time Taken(sec)	0.0014	0.0013	0.429

for n=100000

	Iterative Merge	Recursive Merge	Concurrent Merge
CPU Utilised	0.400	0.401	0.841
Context Switches	595	579	201167
Time Taken(sec)	0.158	0.151	48.1

Problem 4a(Running perf on dtella)

1. By running the “p4a.py” file you will get the x coordinate as time and y coordinate as the cpu usage into the file named “fun1” and x coordinate as time and y coordinate as

context-switches into the file named “fun2” and finally two graphs(graph1.png and graph2.png) are generated.
2. I used “GNUPLOT” for plotting the graphs.
3. I used “PYTHON” as the scripting language.
4. During the idle time i.e without exhaustive using of dtella the cpu usage varied from 0 to 0.004 at max.
5. But during the exhaustive usage time i.e during downloading of approximately 50-60GB of data the cpu usage went up to 0.194 at max.

Problem 4B(Split and Merge file of size > 1GB)

1. I generated a file of size 1.1GB by using the seq command.
2. seq 120000000 -1 1 > filename
3. This prints the numbers in reverse order into the file named filename.
4. I wrote the program using python scripting.
5. I ran the program for the threshold 10MB,200MB,500MB.
6. The screenshots are in the folder.
7. I ran perf on all the 3 cases and put the output in files.
8. I observed that as the threshold increases the time take to merge decreases.