Operating Systems Project 1 Report: Jayce Houghton, Mark Yo

Goal:

The goal of this project was to measure the time of running a system call and to measure the time of running a context switch. We created two separate c programs to accomplish this goal.

Methods for system_call.c:

For the program system_call.c we need to measure the cost in execution time of running a system call. To do this we decided to read from a file using the read() system call and measure the execution time of that call. To do this we used the clock_gettime() function of the <time.h> library. We decided on clock_gettime() over gettimeofday() because clock_gettime() returns the time in nano seconds, and also has more customization options, while gettimeofday() only returns the time in microseconds, thus clock_gettime() returns a much more accurate timing, especially given how fast system calls execute. We opened and close the file to be read before we started timing to ensure that only the read() system call was being timed. In our main program we loop this test 1000 times and ended up with an average time of 0.003075 milliseconds to run the read() system call. This time may be slightly longer then expected because we are no doing a NULL read but instead reading from an external file.

Challenges for system_call.c:

The biggest challenge for system_call.c was figuring out the best function for timing the system call. After testing gettimeofday() enough we finally settled on clock_gettime(). The other challenge was simply figuring out how to run system calls in c since this was our first experience with this.

What we learned from system_call.c

The biggest take away we got from working on system_call.c was just how fast a system call executes to the point where we need to measure the time elapsed in nanoseconds. The other take away was that we should not use gettimeofday() when measuring execution times for exceptionally fast processes.

Execution Output for system_call.c:

Prints out the total time in microseconds, milliseconds, and prints out the average time to run the system call in milliseconds. This output shows that the average time for a read() system call, which takes input from a text file, is 0.003 ms.

```
Total Time in Microseconds: 3074.599000
Total Time in Millseconds: 3.074599
Average Time in Milliseconds: 0.003075
```

Methods in context_switch.c:

In context_switch.c we had to measure the cost in time of a context switch. In order to do this we first had to force the program to run on a single CPU using sched_setaffinity(). The C4 lab

computers are multi-core, so without the processes being forced to run on a single CPU that not need to context switch. To force a context switch we used fork() and pipe(). We used fork() to create a parent and a child process, and we used pipe() to send information between them. We have the parent read from a pipe that get information from the child, forcing a context switch as the parent needs to wait for the child to send that information first. We make sure this happens using the wait(NULL) system call. We check that sched_setaffinity() is resolved in both the parent and child processes to ensure that the processes remain on a single CPU.

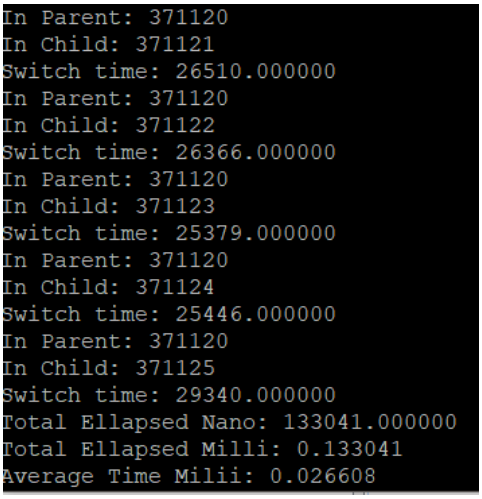
To measure the time it takes to execute this context switch we again used clock_gettime(). We first start the timer, and we then need to perform a write to special timer pipe to pipe the clock start time to the child processes. The context switch then happens, and we immediately call clock_gettime() in the child process to get the ending time. We then read from the timer pipe to get the start time, and subtract it from the ending time to find the elapsed time(the time it took to execute the context switch). We then use another pipe, which we call the main pipe, to pipe the elapsed time back to the parent processes using write so that we can maintain a running total of elapsed time for use in calculating the average time. The parent process then using a read system call to read this value and add it to the running total. We also measure the time of the write instructions that happens immediately after the timer starts and subtract that from the elapsed time of the context switch in order to ensure we are only measuring the context switch.

Challenges for context_switch.c:

The first challenge was forcing the two processes to run on a single processor. We had to spend a lot of time researching how sched_setaffinity() works. The second challenge after this was figuring out how pipes work. For a while we had issue sending information back and forth through our pipes because we had forgotten to properly initialize them. The final challenge was figuring out how to actually time the context switch after we confirmed that one was being force to happen. We did this by piping the start time to the child process and doing the math for elapsed time in the child process, and then sending that back in a pipe to the parent process.

Execution Output of context_switch.c:

Outputs the parent PID, the child's parent PID and the switch time, then outputs the total elapsed time in nanoseconds, elapsed time in milliseconds, and then the average elapsed time in milliseconds. This output shows that the average time in milliseconds for a context switch is 0.026 ms.



Data for context_switch.c and system_call.c:

system_call.c			context_switch.c		
Test #	Time (ns)	Cumulative Total Time (ns)	Test #	Time (ns)	Cumulative Total Time (ns)
1	5542	5542	1	29184	29184
2	13095	18637	2	28938	58122
3	4169	22806	3	28532	86654
4	4200	27006	4	28607	115261
5	4192	31198	5	28065	143326
6	4146	35344	6	32008	175334
7	8299	43643	7	28351	203685
8	4207	47850	8	28174	231859
9	8286	56136	9	28064	259923
10	4196	60332	10	28097	288020
11	4194	64526	11	28255	316275
12	6033	70559	12	28220	344495
13	5357	75916	13	28367	372862
14	4183	80099	14	28367	401229
15	6024	86123	15	28153	429382
16	4086	90209	16	28106	457488
17	2863	93072	17	29269	486757
18	2639	95711	18	28123	514880
19	2712	98423	19	34676	549556
20	2589	101012	20	28419	577975
21	2711	103723	21	28232	606207
22	2743	106466	22	28097	634304
23	2727	109193	23	28138	662442
24	2738	111931	24	28293	690735
25	2752	114683	25	28238	718973

26	5778	120461
27	2735	123196
28	5821	129017
29	2760	131777
30	5393	137170
31	5139	142309
32	2746	145055
33	5457	150512
34	2697	153209
35	2698	155907
36	2748	158655
37	2697	161352
38	2730	164082
39	2773	166855
40	2752	169607
41	2717	172324
42	2596	174920
43	2731	177651
44	2755	180406
45	2757	183163
46	2747	185910
47	2751	188661
48	2744	191405
49	2779	194184
50	2720	196904
51	2719	199623
52	2707	202330
53	2747	205077
54	2809	207886
55	2729	210615
56	2730	213345
57	2724	216069
58	2738	218807
59	2730	221537
60	2740	224277
61	2751	227028
62	2720	229748
63	2733	232481
64	5494	237975
65	2721	240696
66	2763	243459
67	2794	246253
68	2746	248999
69	2766	251765
70	2741	254506

26	28086	747059
27	28432	775491
28	28309	803800
29	28312	832112
30	28273	860385
31	29395	889780
32	33861	923641
33	28815	952456
34	28249	980705
35	28139	1008844
36	28323	1037167
37	28226	1065393
38	28262	1093655
39	30385	1124040
40	28076	1152116
41	28006	1180122
42	28166	1208288
43	28262	1236550
44	28214	1264764
45	28401	1293165
46	31307	1324472
47	28294	1352766
48	28252	1381018
49	28066	1409084
50	27964	1437048
51	28121	1465169
52	30437	1495606
53	28181	1523787
54	28112	1551899
55	28188	1580087
56	28206	1608293
57	29382	1637675
58	28263	1665938
59	30071	1696009
60	28301	1724310
61	28195	1752505
62	28092	1780597
63	28183	1808780
64	28257	1837037
65	30812	1867849
66	28344	1896193
67	28234	1924427
68	28164	1952591
69	27933	1980524
70	28320	2008844

71	5896	260402	71	28178	2037022
72	2743	263145	72	28194	2065216
73	5838	268983	73	28342	2093558
74	2716	271699	74	28183	2121741
75	5823	277522	75	28105	2149846
76	2739	280261	76	28051	2177897
77	2726	282987	77	27985	2205882
78	2743	285730	78	33016	2238898
79	2720	288450	79	28411	2267309
80	5480	293930	80	28299	2295608
81	2779	296709	81	28195	2323803
82	2767	299476	82	28100	2351903
83	2701	302177	83	28135	2380038
84	5861	308038	84	28252	2408290
85	2714	310752	85	28276	2436566
86	2760	313512	86	28196	2464762
87	5468	318980	87	28488	2493250
88	2763	321743	88	28250	2521500
89	2751	324494	89	28252	2549752
90	2757	327251	90	28277	2578029
91	2723	329974	91	49255	2627284
92	2751	332725	92	28305	2655589
93	2743	335468	93	28194	2683783
94	2741	338209	94	28207	2711990
95	2724	340933	95	28086	2740076
96	2767	343700	96	28200	2768276
97	2760	346460	97	28234	2796510
98	5464	351924	98	28036	2824546
99	2717	354641	99	27971	2852517
100	2755	357396	100	27961	2880478

Estimate Time it took:

10 Hours

Conclusion:

This project allowed us to understand the intricacies of processes running on a CPU. Through analyzing and comparing the amount of time a system call and context switch takes, we are able to further our knowledge on the load a CPU has to take when running within a single process. Specifically, limiting the context switch to one CPU shows the limitations of a single CPU system. It was a bit time-consuming figuring out binding the two processes to a single CPU, but that challenge gave us extensive knowledge on manipulating CPU usage within our programs. Overall, we gained a large amount of skills to apply in future projects, and furthered our understanding of operating systems.