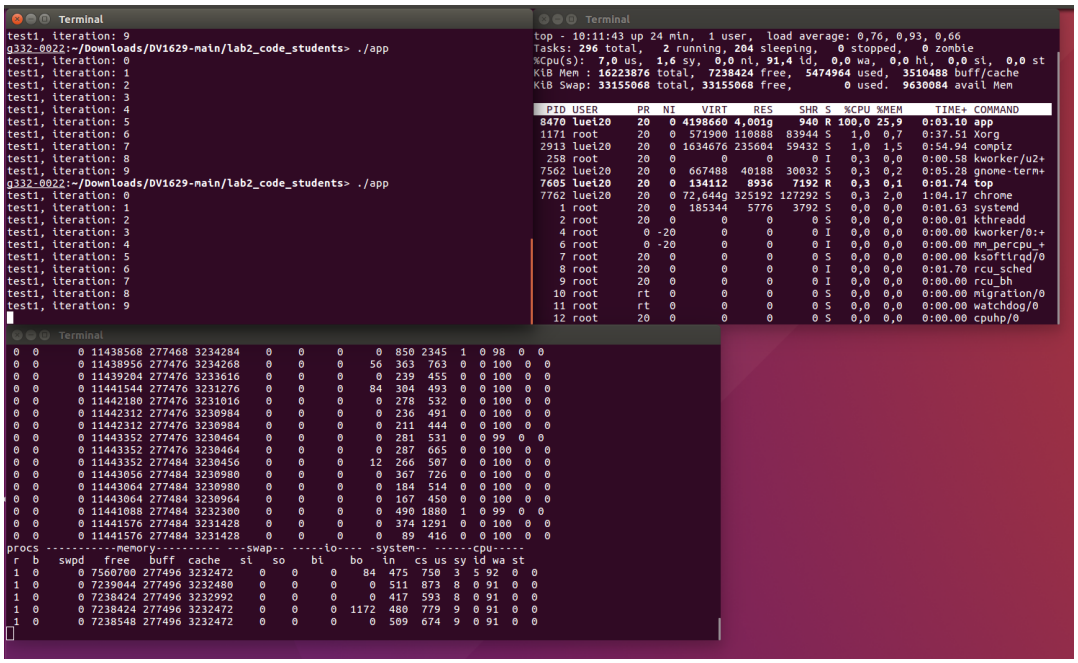
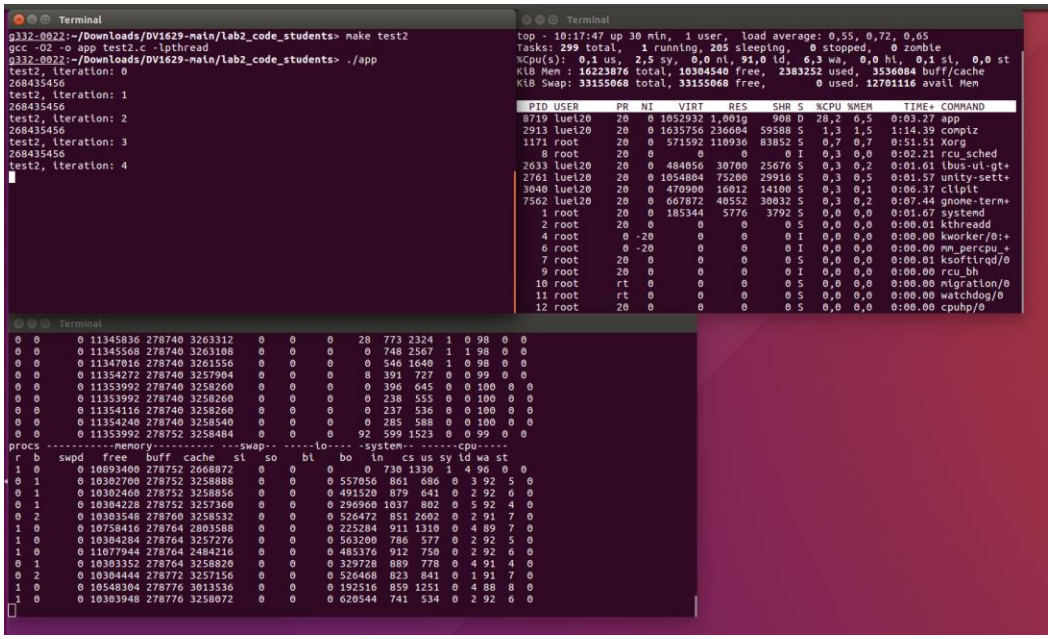
Tasks

Task 1. CPU-bound vs. I/O-bound processes



The program uses 4GB of physical memory – It corresponds to the Home assignment



The program use 1GB of physical memory with no swap outs. It correspond to the home assignment if we sum up the iterations (256MB * 5 ~= 1 GB).
It swaps out no memory, the whole program is in the physical memory.

Task 2. Overlapping CPU and I/O execution

What is the difference between them in terms of how they execute the test programs?

It appears that the run1 script is being executed sequentially, whereas the run2 script is running the test programs concurrently.

Execute the script run1 and measure the execution time. Study the cpu utilization using top during the execution.
How long time did it take to execute the script and how did the cpu utilization vary?

The program took around 91.5 seconds, we could see that it is running test1 first and that the CPU utilization was at 98%, when test2 starts the CPU utilization was around 1%.

Execute the script run2 and measure the execution time. Study the cpu utilization using top during the execution.
How long time did it take to execute the script and how did the cpu utilization vary?

Both tests are running concurrently. The CPU utilization was at 96% for test1 and at 11% for test2.

Which of the two cases executed fastest?
Test2 executed a bit faster.

In both cases, the same mount of work was done. I which case was the system best utilized and why?

The second test run was more effeciently utilized as it used more of the CPU to simultaneously run both programs, resulting in a faster execution time.

Task 3. Implementation of FIFO page replacement

See source code.

Task 4. Page fault measurements for the FIFO page replacement policy

Mp3d.mem:

	Number of physical pages							
Page size	1	2	4	8	16	32	64	128
128	55421	22741	13606	6810	3121	1503	1097	877
256	54357	20395	11940	4845	1645	939	669	478
512	52577	16188	9458	2372	999	629	417	239
1024	51804	15393	8362	1330	687	409	193	99

mult.mem:

	Number of physical pages							
Page size	1	2	4	8	16	32	64	128
128	45790	22303	18034	1603	970	249	67	67
256	45725	22260	18012	1529	900	223	61	61
512	38246	16858	2900	1130	489	210	59	59
1024	38245	16855	2890	1124	479	204	57	57

Task 5. Evaluation of the FIFO page replacement policy

When the number of physical pages is constant but we increase the page size; the amount of page faults decrease. The reason is because the memory addresses referenced in the program exists in less pages if the page size is increased (the page ranges are larger and the memory of the program is broken down less), thus less page faults. The time complexity of swapping in/out large pages is less efficient though.

When the number of physical pages is increased but keep the page size constant instead, there is a decrease in the number of page faults again. But this time because the physical memory is bigger and can hold more pages and therefore also more memory references and thus less page faults appear.

If there is an increase in page faults the program references memory beyond the page sizes in physical memory and must add the new memory reference along with its page into physical memory. If there is a decrease in the number of page faults the program references memory in the range of the page sizes that is already in physical memory. It depends on the program and the user/computer using the program.

Big changes happens when the page size go from 256 to 512 on multiple different number of pages with the memory references found in the “multi.mem” program. This happens because the program mainly references memory in intervals of 512 addresses.
When the number of page faults doesn’t decrease much even though there is an increase in page size; the program rarely references memory across intervals of 1024. It is running almost equally well with 512 in page size.

Task 6. Implementation of LRU page replacement

See source code.

Task 7. Page fault measurements for the LRU page replacement policy

mp3d.mem:

	Number of physical pages							
Page size	1	2	4	8	16	32	64	128
128	55421	16973	11000	6536	1907	995	905	796
256	54357	14947	9218	3811	794	684	557	417
512	52577	11432	6828	1617	603	503	362	206
1024	51804	10448	5605	758	472	351	167	99

Task 8. Comparison of the FIFO and LRU page replacement policies

LRU seems to give the lower amount of page faults in general because it takes into account how much each page is used in the program. FIFO doesn't. When a program is running and the user/computer is using some page of the program alot and then wants to use another page not found in physical memory. FIFO might replace the much used page in the physical memory because it was the next in the queue to get rid of. But LRU checks how old the page is, and if the page has been used recently it is young and not replaced. The oldest page in terms of recent memory references is gotten rid of.

FIFO and LRU have the same amount of page faults in some cases. This happens when the page size is 1. If the page size is one there is only one spot for a physical page and the algorithm that chooses that one page doesn't mattter. When the number of physical pages is 2 or more the algorithms affects the number of page faults.
The same amount of page faults also happens when the page size is 1024 and number of pages is 128. That happens because all the memory references in the program are not found in memory the first time they are referenced but then, when all memory references has been made, the physical memory is big enough for the whole program to exist there and no more page faults occurs.

Task 9. Implementation of Optimal page replacement (Bélády's algorithm)

See source code.

Task 10. Page fault measurements for the Optimal (Bélády's) page replacement policy

mp3d.mem:

Number of physical pages

Page size	1	2	4	8	16	32	64	128
128	55421	15856	8417	3656	1092	824	692	558
256	54357	14168	6431	1919	652	517	395	295
512	52577	11322	4191	920	470	340	228	173
1024	51804	10389	3367	496	339	213	107	99

Task 11. Comparison of the FIFO, LRU, and Optimal page replacement policies

The optimal algorithm gives the least amount of page faults simply because it knows which page will be used the furthest into the future and can get rid of that page when a page fault occurs. Programs cannot predict the future in practice. Even though the optimal algorithm cannot be used in practice it still holds value since it can be used as a base to compare how well other page fault algorithm works.

LRU and FIFO has the same number of page faults as the optimal one when the number of physical pages is 1 and also when the number of physical pages is 128 and the page size is 1024. That happens for the same reasons as mentioned in task 8.

Home Assignments

Home Assignment 3.

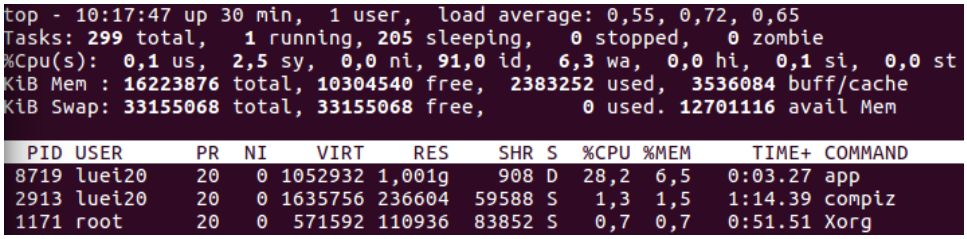
The program ***test1.c*** allocates 4294967296 bytes of memory ~= 4 GB of data

The amount of data written in each iteration is 268435456 bytes of data ~= 256 MB of data.

Home Assignment 4.

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 10893400 278752 2668872    0    0     0     0  730 1330  1  4 96  0  0
```

- The memory column shows the amount of memory a process is using
- The number of swap ins and outs can be found under "swap" → "si" and "so"
- The number of I/O blocks written can be found under "io" → "bi" and "bo"



- TIME+ shows the total cpu time used in hundredths of a second
- %CPU shows the CPU utilization a process is using