

Tasks

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

```
Terminal
test1, iteration: 9
g332-0822:~/Downloads/DV1629-main/lab2_code_students> ./app
test1, iteration: 0
test1, iteration: 1
test1, iteration: 2
test1, iteration: 3
test1, iteration: 4
test1, iteration: 5
test1, iteration: 6
test1, iteration: 7
test1, iteration: 8
test1, iteration: 9
g332-0822:~/Downloads/DV1629-main/lab2_code_students> ./app
test1, iteration: 0
test1, iteration: 1
test1, iteration: 2
test1, iteration: 3
test1, iteration: 4
test1, iteration: 5
test1, iteration: 6
test1, iteration: 7
test1, iteration: 8
test1, iteration: 9

Terminal
top - 18:11:43 up 24 min, 1 user, load average: 0.76, 0.93, 0.66
Tasks: 296 total, 2 running, 204 sleeping, 0 stopped, 0 zombie
%Cpu(s): 7.0 us, 1.6 sy, 0.0 ni, 91.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 16223876 total, 7238424 free, 5474964 used, 3510488 buff/cache
KiB Swap: 33155068 total, 33155068 free, 0 used, 9630084 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR   S  %CPU  %MEM    TIME+  COMMAND
 8470 luel20    20   0 4198660 4,001g  940 R 100.0 25.9  0:03.10 app
1171 root        20   0 571900 110888 83944 S   1.0  0.7  0:17.51 Xorg
2913 luel20    20   0 1634076 235604 59432 S   1.0  1.5  0:54.94 compiz
258 root        20   0 0 0 0 I   0.3  0.0  0:00.58 kworker/u2+
7562 luel20    20   0 667488 40188 30032 S   0.3  0.2  0:05.28 gnome-term+
7605 luel20    20   0 134112 8936 7192 R   0.3  0.1  0:01.74 top
7762 luel20    20   0 72,644g 325192 127292 S   0.3  2.0  1:04.17 chrome
1 root        20   0 185344 5776 3792 S   0.0  0.0  0:01.63 systemd
2 root        20   0 0 0 0 S   0.0  0.0  0:00.01 kthreadd
4 root        20   0 0 0 0 S   0.0  0.0  0:00.00 kworker/0:++
6 root        20   0 0 0 0 S   0.0  0.0  0:00.00 mm_percpu_+
7 root        20   0 0 0 0 S   0.0  0.0  0:00.00 ksoftirqd/0
8 root        20   0 0 0 0 S   0.0  0.0  0:01.70 rcu_sched
9 root        20   0 0 0 0 S   0.0  0.0  0:00.00 rcu_bh
10 root       rt    0 0 0 0 S   0.0  0.0  0:00.00 migration/0
11 root       rt    0 0 0 0 S   0.0  0.0  0:00.00 watchdog/0
12 root       20   0 0 0 0 S   0.0  0.0  0:00.00 cpuhp/0

Terminal
0 0 0 11438568 277468 3234284 0 0 0 850 2345 1 0 98 0 0
0 0 0 11438956 277476 3234208 0 0 0 56 363 763 0 0 100 0 0
0 0 0 11439204 277476 3233616 0 0 0 0 239 455 0 0 100 0 0
0 0 0 11441544 277476 3231276 0 0 0 84 304 493 0 0 100 0 0
0 0 0 11442180 277476 3231010 0 0 0 0 278 532 0 0 100 0 0
0 0 0 11442312 277476 3230984 0 0 0 0 236 491 0 0 100 0 0
0 0 0 11442312 277476 3230984 0 0 0 0 211 444 0 0 100 0 0
0 0 0 11443352 277476 3230464 0 0 0 0 281 531 0 0 99 0 0
0 0 0 11443352 277476 3230464 0 0 0 0 287 605 0 0 100 0 0
0 0 0 11443352 277484 3230456 0 0 0 12 266 507 0 0 100 0 0
0 0 0 11443656 277484 3230980 0 0 0 0 367 726 0 0 100 0 0
0 0 0 11443664 277484 3230980 0 0 0 0 184 514 0 0 100 0 0
0 0 0 11443664 277484 3230964 0 0 0 0 167 450 0 0 100 0 0
0 0 0 11441088 277484 3232300 0 0 0 0 490 1880 1 0 99 0 0
0 0 0 11441576 277484 3231428 0 0 0 0 374 1291 0 0 100 0 0
0 0 0 11441576 277484 3231428 0 0 0 0 89 416 0 0 100 0 0

procs-----memory-----swap-----lo-----system-----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
1 0 0 7560700 277496 3232472 0 0 0 84 475 750 3 5 92 0 0
1 0 0 7239044 277496 3232480 0 0 0 0 511 873 8 0 91 0 0
1 0 0 7239044 277496 3232982 0 0 0 0 417 593 8 0 91 0 0
1 0 0 7238424 277496 3232472 0 0 0 1172 480 779 9 0 91 0 0
1 0 0 7238548 277496 3232472 0 0 0 0 509 674 9 0 91 0 0
```

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

```
Terminal
g332-0822:~/Downloads/DV1629-main/lab2_code_students> make test2
gcc -O2 -o app test2.c -lpthread
g332-0822:~/Downloads/DV1629-main/lab2_code_students> ./app
test2, iteration: 0
268435456
test2, iteration: 1
268435456
test2, iteration: 2
268435456
test2, iteration: 3
268435456
test2, iteration: 4

Terminal
top - 10:17:47 up 30 min, 1 user, load average: 0.55, 0.72, 0.65
Tasks: 299 total, 1 running, 205 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 2.5 sy, 0.0 ni, 91.0 id, 0.3 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem : 16223876 total, 10304540 free, 2383252 used, 3536084 buff/cache
KiB Swap: 33155068 total, 33155068 free, 0 used, 12701116 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR   S  %CPU  %MEM    TIME+  COMMAND
 8710 luel20    20   0 1052932 1,001g  908 D 28.2 6.5  0:03.27 app
2913 luel20    20   0 1635756 236604 59588 S   1.3  1.5  1:14.39 compiz
1171 root        20   0 571592 110936 83852 S   0.7  0.7  0:51.51 Xorg
8 root        20   0 0 0 0 I   0.3  0.0  0:02.21 rcu_sched
2633 luel20    20   0 484056 30700 25676 S   0.3  0.2  0:01.61 tbus-ut-gt+
2761 luel20    20   0 1054804 75200 29916 S   0.3  0.5  0:01.57 unity-sett+
3040 luel20    20   0 470900 16012 14100 S   0.3  0.1  0:06.37 clipit
7562 luel20    20   0 667672 40552 30032 S   0.3  0.2  0:07.44 gnome-term+
1 root        20   0 185344 5776 3792 S   0.0  0.0  0:01.67 systemd
2 root        20   0 0 0 0 S   0.0  0.0  0:00.01 kthreadd
4 root        20   0 0 0 0 S   0.0  0.0  0:00.00 kworker/0:++
6 root        20   0 0 0 0 S   0.0  0.0  0:00.00 mm_percpu_+
7 root        20   0 0 0 0 S   0.0  0.0  0:00.01 ksoftirqd/0
9 root        20   0 0 0 0 S   0.0  0.0  0:00.00 rcu_bh
10 root       rt    0 0 0 0 S   0.0  0.0  0:00.00 migration/0
11 root       rt    0 0 0 0 S   0.0  0.0  0:00.00 watchdog/0
12 root       20   0 0 0 0 S   0.0  0.0  0:00.00 cpuhp/0

Terminal
0 0 0 11345836 278740 3263312 0 0 0 28 773 2324 1 0 98 0 0
0 0 0 11345568 278740 3263108 0 0 0 0 748 2567 1 1 98 0 0
0 0 0 11347016 278740 3261556 0 0 0 0 546 1640 1 0 98 0 0
0 0 0 11354272 278740 3257904 0 0 0 0 391 727 0 0 99 0 0
0 0 0 11353992 278740 3258260 0 0 0 0 396 645 0 0 100 0 0
0 0 0 11353992 278740 3258260 0 0 0 0 238 555 0 0 100 0 0
0 0 0 11354116 278740 3258260 0 0 0 0 237 536 0 0 100 0 0
0 0 0 11354240 278740 3258540 0 0 0 0 285 588 0 0 100 0 0
0 0 0 11353992 278752 3258484 0 0 0 92 599 1523 0 0 99 0 0

procs-----memory-----swap-----lo-----system-----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
1 0 0 10093400 278752 3260872 0 0 0 730 1330 1 4 96 0 0
0 1 0 10302700 278752 3258888 0 0 0 557056 861 686 0 3 92 5 0
0 1 0 10302460 278752 3258856 0 0 0 491520 879 641 0 2 92 6 0
0 1 0 10304208 278752 3257360 0 0 0 296960 1017 802 0 5 92 4 0
0 2 0 10303548 278760 3258532 0 0 0 526472 851 2602 0 2 91 7 0
1 0 0 10758416 278764 2803588 0 0 0 225284 911 1310 0 4 89 7 0
1 0 0 10304284 278764 3257276 0 0 0 563200 786 577 0 2 92 5 0
1 0 0 11077944 278764 2484216 0 0 0 485376 912 750 0 2 92 6 0
0 1 0 10303352 278764 3258820 0 0 0 329728 889 778 0 4 91 4 0
0 2 0 10304444 278772 3257156 0 0 0 526468 823 841 0 1 91 7 0
1 0 0 10508304 278776 3013536 0 0 0 192516 859 1251 0 4 88 0 0
1 0 0 10303948 278776 3258072 0 0 0 620544 741 534 0 2 92 6 0
```

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:

Page size	Number of physical pages							
	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 a lot 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 matter. When the number of physical pages is 2 or more the algorithms affect 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 have been made, the physical memory is big enough for the whole program to exist there and no more page faults occur.

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:

Page size	Number of physical pages							
	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  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"

```

top - 10:17:47 up 30 min,  1 user,  load average: 0,55, 0,72, 0,65
Tasks: 299 total,   1 running, 205 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,1 us,  2,5 sy,   0,0 ni, 91,0 id,   6,3 wa,   0,0 hi,   0,1 si,   0,0 st
KiB Mem : 16223876 total, 10304540 free,  2383252 used,  3536084 buff/cache
KiB Swap: 33155068 total, 33155068 free,    0 used. 12701116 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
8719	luei20	20	0	1052932	1,001g	908	D	28,2	6,5	0:03.27	app
2913	luei20	20	0	1635756	236604	59588	S	1,3	1,5	1:14.39	compiz
1171	root	20	0	571592	110936	83852	S	0,7	0,7	0:51.51	Xorg

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