Report 1 – Cis 415
By: Matt Almenshad

Project 1 Report

First, I would like to sincerely apologize for the late submission. I did not have an intention of creating extra work for the graders. So I decided to take an extra day to clean up the mess I made and settle for the 10% penalty rather than submit a cluster of maybe-working files and lose even more.

Overview

I started building this project looking ahead to the very end. At first, I decided to implement a linked list based structure to store my process struts. I thought that eventually I would end up using an AVL tree to balance the process based on their process. So I ended up implementing a single linked queue. However, upon discourse with professor Sventek, it became clear to me that a heap is a much better option. The realization lead me to restructure my project in favor of an array of node structures to be later replaced with a heap. Midway through my implementation of part 3, I decided it would be to have two queues. So that when I process a structure, once I am done with it, I would put in into the next queue. This way I reduce starvation tremendously.

As for my structure, I made a list of processes structure as well and a process node structure to hold all the relevant information. The list (pList) has two queues of process nodes (pNodes) as well as some indexing and size variables. The node has a parsed command for use with execvp, PID, number of arguments, and a status to indicate whether or not the process is terminated. It also holds proc information including the number read and write calls, the CPU time spent on user code as well as kernel code for the process. Later on I introduced a priority and a number of runs.

Report 1 – Cis 415
By: Matt Almenshad

USPSV1

This program is very simple. It starts by creating a list of nodes using the function pList_create(); The process then is populated with parsed commands from the workload provided through standard input using the function pList_insert_stdin(list). The a tracker created to keep track of all the allocated spaces using pList_create_tracker(list) following by holding the number of allocated nodes obtained by using pList_getQ1Size(list). Then I invoke the function pList_run_NOWAIT(list) which forks all the processes, runs execvp and updates the PIDs of their respective nodes. Then I call pList_wait(list) which invokes waitpid() on all inntiated process, forcing the main process to wait for all the children to be terminated.

afterwards I call queue_destroy(tracker, tracker_size); to free all the allocated node space using the tracker I created earlier, and finally pList_destroy(list) to free the space allocated for the list that holds all the nodes.

Compiling the program: done by using “make 1”.

Running the program: done by using “./1 < workload_path”.

Valgrind test result:

```
[cis415@cis415-arch project1.2]$ valgrind ./1 < workload/workload2
==27092== Memcheck, a memory error detector
==27092== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==27092== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==27092== Command: ./1
==27092==
Started all forked processes
Done
==27092==
==27092== HEAP SUMMARY:
==27092==     in use at exit: 0 bytes in 0 blocks
==27092==   total heap usage: 20 allocs, 20 frees, 3,684 bytes allocated
==27092==
==27092== All heap blocks were freed -- no leaks are possible
==27092==
==27092== For counts of detected and suppressed errors, rerun with: -v
==27092== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Report 1 – Cis 415
By: Matt Almenshad

USPSV2

for part two of the project, I choose to implement professor Sventek's approach rather than using sigwat(). The reason being that sigwait() can be issued after the main signal has been issued which is a very troublesome race condition. Professor Sventek's approach on the other hand, guarantees that the child will inherit the signal handlers and global variables. I start by initializing the global variables. Then create a list similar to the previous part. Then I call signal to set up the system-call hash-table. I then using pList_insert_stdin(list) and pList_create_tracker(list) as well pList_getQ1Size(list) similar to parse the workload and create the node structures as in part 1. Although this time I call pList_run(list) rather than pList_run(list) which puts the children in a sleep state awaiting for SIGUSR1 I then call pList_start(list) which starts all the nodes by sending the signal SIGUSR1 then I stop them using pList_stop(list) which sends the SIGSTOP signal to all the nodes. Then I call pList_cont(list) to resume all process followed by pList_wait(list) to invoke waitpid() on all the signals. then I do my heap clean-up using queue_destroy(tracker, tracker_size); as well as pList_destroy(list).

Compiling the program: done by using "make 2".
Running the program: done by using "./2 < workload_path".
Valgrind test result:

```
==27118== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==27118== Command: ./2
==27118==
Initialized all processes
Started all processes with SIGUSR1
Stopped all processes with SIGSTOP
Resumed all processes with SIGCONT
Done
==27118==
==27118== HEAP SUMMARY:
==27118==     in use at exit: 0 bytes in 0 blocks
==27118==   total heap usage: 20 allocs, 20 frees, 3,684 bytes allocated
==27118==
==27118== All heap blocks were freed -- no leaks are possible
==27118==
==27118== For counts of detected and suppressed errors, rerun with: -v
==27118== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[cis415@cis415-arch project1.2]$
```

Report 1 – Cis 415
By: *Matt Almenshad*

<u>USPSV3</u>

As with the previous part, I start by initializing global variables. After that is done I create the list using pList_create(); then I initialize the global pointer to hold a reference to my current node, as that is the only way for me to pass it to the signal handlers Then I use professor Sventek's code to use sigaction to associate SIGCHILD with my modified version of his child handler. I also issue the SIGUSR1 and SIGALRM signal() calls then I parse and initialize my list and tracker using pList_insert_stdin(list), pList_create_tracker(list) and pList_getQ1Size(list). This time I call pList_runAndStop(list) which sends SIGUSR1 then SIGSTOP to all my process. Then I resume the process using pList_start(list), 3 seconds later I call pList_stop(list) to stop all the processes putting in a state in which they are ready for the scheduler to execute. then I issue an alarm(1) and wait while dead process are less than created process using a loop and a sleep call. 1 second later, my alarm handler is calling, stopping the current running process if any, pulling the next one out of the queue and running it. if the process is done, the scheduler throws away the reference (no worries, I still have it in the tracker), otherwise I put it in the next queue. once a queue is fully iterated I move on to the next one. Once the child handler is invoked, it sets the status of the node to terminated so that the scheduler knows to throw it away, and increases the number of dead children. once all children are dead, I free the heap spaces using queue_destroy(tracker, tracker_size) as well as pList_destroy(list).

<u>Compiling the program:</u> done by using "make 3".

<u>Running the program:</u> done by using "./3 < workload_path".

Report 1 – Cis 415
By: *Matt Almenshad*

Valgrind test result:

```
[cis415@cis415-arch project1.2]$ valgrind ./3 < workload/workload2
==27200== Memcheck, a memory error detector
==27200== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==27200== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==27200== Command: ./3
==27200==
Initialized and stopped all procs with SIGUSR1 and SIGSTOP...
Stopped: 27201
Resumed: 27202
Stopped: 27202
Resumed: 27203
Stopped: 27203
Resumed: 27204
Stopped: 27204
Resumed: 27201
Stopped: 27201
Resumed: 27202
Stopped: 27202
Resumed: 27203
Stopped: 27203
Resumed: 27204
Stopped: 27204
Resumed: 27201
Stopped: 27201
Resumed: 27202
Stopped: 27202
Resumed: 27203
Stopped: 27203
Resumed: 27204
Stopped: 27204
Resumed: 27201
Stopped: 27201
Resumed: 27202
Stopped: 27202
Resumed: 27203
```

```
Stopped: 27202
Resumed: 27203
Stopped: 27203
Resumed: 27204
Stopped: 27204
Resumed: 27201
Stopped: 27201
Resumed: 27202
Terminated: 27202
Stopped: 27203
Resumed: 27204
Terminated: 27204
Stopped: 27201
Resumed: 27203
Stopped: 27203
Resumed: 27201
Stopped: 27201
Resumed: 27203
Terminated: 27203
Stopped: 27201
Resumed: 27201
Terminated: 27201
Done!
==27200==
==27200== HEAP SUMMARY:
==27200==     in use at exit: 0 bytes in 0 blocks
==27200==   total heap usage: 20 allocs, 20 frees, 3,684 bytes allocated
==27200==
==27200== All heap blocks were freed -- no leaks are possible
==27200==
==27200== For counts of detected and suppressed errors, rerun with: -v
==27200== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Report 1 – Cis 415
By: *Matt Almenshad*

USPSV4

Basically the same as part 3 except that I use functions to update and print the process information after stopping every process. The information includes:

- PID
- Size (in pages)
- number of system read calls during the last time quantum
- number of system write calls during the last time quantum
- total IO calls during the last time quantum
- total CPU time during the last time quantum
- time CPU spent on user code during the last time quantum (in HZ)
- time CPU spent on kernel code during the last time quantum (in HZ)
- name of the process.

The proc files I used were stat, statm and IO.

Compiling the program: done by using “make 4”.
Running the program: done by using “./4 < workload_path”.

Report 1 – Cis 415

By: *Matt Almenshad*

Valgrind test result:

```
==27219== Memcheck, a memory error detector
==27219== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==27219== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==27219== Command: ./4
==27219==
[PID]   Size (pages)    IO reads        IO writes       Total IO        CPU (User, Krnl) [HZ]           (Name)
[27220] 23918           2               2               4               0 (0, 0)                        (memcheck-amd64-)
[27221] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27222] 23918           3               3               6               0 (0, 0)                        (memcheck-amd64-)
[27223] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27224] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27225] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27226] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27227] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27228] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27229] 23918           2               3               5               1 (1, 0)                        (memcheck-amd64-)
[27230] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27231] 23918           2               3               5               1 (1, 0)                        (memcheck-amd64-)
[27232] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27233] 23918           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[27220] 1026            8               1               9               0 (0, 0)                        (iotest2)
[27221] 1009            8               0               8               45 (43, 2)                      (cputest2)
[27222] 1043            7               2373265         2373272         89 (64, 25)                     (iotest2)
[27223] 1009            8               0               8               48 (48, 0)                      (cputest2)
[27224] 1043            8               2151333         2151341         80 (48, 32)                     (iotest2)
[27225] 1009            8               0               8               47 (47, 0)                      (cputest2)
[PID]   Size (pages)    IO reads        IO writes       Total IO        CPU (User, Krnl) [HZ]           (Name)
[27226] 1043            8               2398662         2398670         89 (57, 32)                     (iotest2)
[27227] 1009            8               0               8               47 (47, 0)                      (cputest2)
[27228] 1043            8               2444292         2444300         91 (52, 39)                     (iotest2)
[27229] 1009            8               0               8               47 (43, 4)                      (cputest2)
[27230] 1043            8               2400631         2400639         91 (51, 40)                     (iotest2)
[27231] 1009            8               0               8               48 (48, 0)                      (cputest2)
[27232] 1043            8               2231014         2231022         84 (50, 34)                     (iotest2)
[27233] 1009            8               0               8               47 (47, 0)                      (cputest2)
[27220] 1043            0               2384900         2384900         89 (56, 33)                     (iotest2)
[27221] 1009            0               0               0               46 (46, 0)                      (cputest2)
[27222] 1043            0               2462994         2462994         92 (52, 40)                     (iotest2)
[27223] 1009            0               0               0               46 (46, 0)                      (cputest2)
```

```
[PID]   Size (pages)    IO reads        IO writes       Total IO        CPU (User, Krnl) [HZ]           (Name)
[27224] 1043            0               2278993         2278993         86 (61, 25)                     (iotest2)
[27226] 1043            0               2305495         2305495         87 (58, 29)                     (iotest2)
[27228] 1043            0               2330984         2330984         88 (57, 31)                     (iotest2)
[27230] 1043            0               2173661         2173661         83 (55, 28)                     (iotest2)
[27232] 1043            0               2298704         2298704         87 (44, 43)                     (iotest2)
[27220] 1043            0               2272064         2272064         86 (53, 33)                     (iotest2)
[27222] 1043            0               2369833         2369833         89 (68, 21)                     (iotest2)
[27224] 1043            0               2294205         2294205         85 (64, 21)                     (iotest2)
[27226] 1043            0               2360907         2360907         88 (61, 27)                     (iotest2)
[27228] 1043            0               2274095         2274095         86 (53, 33)                     (iotest2)
[27230] 1043            0               2362949         2362949         90 (64, 26)                     (iotest2)
[27232] 1043            0               2309237         2309237         87 (58, 29)                     (iotest2)
[27220] 1043            0               2260358         2260358         85 (49, 36)                     (iotest2)
[27222] 1043            0               2310942         2310942         88 (66, 22)                     (iotest2)
[27224] 1043            0               2323562         2323562         87 (60, 27)                     (iotest2)
[27226] 1043            0               2337310         2337310         88 (53, 35)                     (iotest2)
[27228] 1043            0               2274723         2274723         85 (65, 20)                     (iotest2)
[27230] 1043            0               2383212         2383212         91 (58, 33)                     (iotest2)
[27232] 1043            0               2248338         2248338         84 (64, 20)                     (iotest2)
[27220] 1043            0               2346876         2346876         89 (61, 28)                     (iotest2)
[PID]   Size (pages)    IO reads        IO writes       Total IO        CPU (User, Krnl) [HZ]           (Name)
[27224] 1043            0               0               0               0 (0, 0)                        (iotest2)
[27228] 1043            0               0               0               0 (0, 0)                        (iotest2)
[27230] 1043            0               2352201         2352201         89 (60, 29)                     (iotest2)
[27220] 1043            0               0               0               0 (0, 0)                        (iotest2)
[27228] 1043            0               0               0               0 (0, 0)                        (iotest2)
[27220] 1043            0               0               0               0 (0, 0)                        (iotest2)
[27220] 1043            0               0               0               0 (0, 0)                        (iotest2)
Done!
==27219==
==27219== HEAP SUMMARY:
==27219==     in use at exit: 0 bytes in 0 blocks
==27219==   total heap usage: 102 allocs, 102 frees, 5,354 bytes allocated
==27219==
==27219== All heap blocks were freed -- no leaks are possible
==27219==
==27219== For counts of detected and suppressed errors, rerun with: -v
==27219== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[cis415@cis415-arch project1.2]$
```

Report 1 – Cis 415
By: *Matt Almenshad*

USPSV5

For part 5 I changed my insert functionality to simulate a heap insertion. I used the methodology and some of the code described in this page:

http://robin-thomas.github.io/max-heap/

to achieve my heap insertion. For the key, I used a priority constructed by obtaining the log of the total number of reads and writes times (0.1)less the log number of the total CPU usage times (0.1) less the number of times the process has been ran times (0.1). This causes the IO bound processes to get a higher priority. Along with the aging and number of queues I use, all process get to run with as minimal starvation as possible. Eventually, more IO intensive program remain despite the optimization working perfectly.

<u>Compiling the program:</u> done by using “make 5”.

<u>Running the program:</u> done by using “./5 < workload_path”.

<u>Valgrind test result:</u>

```
[cis415@cis415-arch project1.2]$ valgrind --track-origins=yes ./5 < workload/workload2
==26778== Memcheck, a memory error detector
==26778== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==26778== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==26778== Command: ./5
==26778==
[PID]   Size (pages)    IO reads        IO writes       Total IO        CPU (User, Krnl) [HZ]           (Name)
[26779] 48494           2               2               4               0 (0, 0)                        (memcheck-amd64-)
[26780] 48494           2               3               5               0 (0, 0)                        (memcheck-amd64-)
[26781] 48494           3               3               6               0 (0, 0)                        (memcheck-amd64-)
[26782] 48494           3               3               6               0 (0, 0)                        (memcheck-amd64-)
[26779] 1009            8               1               9               0 (0, 0)                        (sPrinter1)
[26780] 1009            8               0               8               0 (0, 0)                        (sPrinter2)
[26781] 1009            7               0               7               0 (0, 0)                        (sPrinter3)
[26782] 1009            7               0               7               0 (0, 0)                        (sPrinter4)
[26779] 1009            0               0               0               0 (0, 0)                        (sPrinter1)
[26780] 1009            0               0               0               0 (0, 0)                        (sPrinter2)
[26781] 1009            0               0               0               0 (0, 0)                        (sPrinter3)
[26782] 1009            0               0               0               0 (0, 0)                        (sPrinter4)
[26779] 1009            0               0               0               0 (0, 0)                        (sPrinter1)
[26780] 1009            0               0               0               0 (0, 0)                        (sPrinter2)
[26781] 1009            0               0               0               0 (0, 0)                        (sPrinter3)
[26782] 1009            0               0               0               0 (0, 0)                        (sPrinter4)
[26780] 1009            0               0               0               0 (0, 0)                        (sPrinter2)
[26782] 1009            0               0               0               0 (0, 0)                        (sPrinter4)
[26782] 1009            0               0               0               0 (0, 0)                        (sPrinter4)
Done!
==26778==
==26778== HEAP SUMMARY:
==26778==     in use at exit: 0 bytes in 0 blocks
==26778==   total heap usage: 20 allocs, 20 frees, 3,723 bytes allocated
==26778==
==26778== All heap blocks were freed -- no leaks are possible
==26778==
==26778== For counts of detected and suppressed errors, rerun with: -v
==26778== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Report 1 – Cis 415

By: *Matt Almenshad*

<u>Priority in action:</u>

[PID]	Size (pages)	IO reads	IO writes	Total IO	CPU (User, Krnl) [HZ]	(Name)
[26360]	1043	0	2223430	2223430	82 (55, 27)	(iotest2)
[26362]	1043	0	2211587	2211587	83 (32, 51)	(iotest2)
[26364]	1043	0	2168689	2168689	82 (57, 25)	(iotest2)
[26366]	1043	0	2051594	2051594	77 (58, 19)	(iotest2)
[26356]	1043	0	1890028	1890028	74 (41, 33)	(iotest2)
[26355]	1009	0	0	0	0 (0, 0)	(cputest2)
[26354]	1043	0	0	0	0 (0, 0)	(iotest2)
[26358]	1043	0	2142068	2142068	80 (50, 30)	(iotest2)
[26360]	1043	0	2306589	2306589	87 (67, 20)	(iotest2)
[26362]	1043	0	2212520	2212520	84 (60, 24)	(iotest2)
[26364]	1043	0	2134664	2134664	81 (58, 23)	(iotest2)
[26366]	1043	0	2283795	2283795	86 (56, 30)	(iotest2)
[26356]	1043	0	2178350	2178350	85 (63, 22)	(iotest2)
[26354]	1043	0	0	0	0 (0, 0)	(iotest2)
[26358]	1043	0	2160374	2160374	81 (49, 32)	(iotest2)
[26360]	1043	0	1963652	1963652	75 (56, 19)	(iotest2)
[26362]	1043	0	2309255	2309255	86 (61, 25)	(iotest2)
[26364]	1043	0	1728298	1728298	65 (43, 22)	(iotest2)
[26366]	1043	0	2139277	2139277	80 (55, 25)	(iotest2)
[26356]	1043	0	2275256	2275256	87 (59, 28)	(iotest2)

[PID]	Size (pages)	IO reads	IO writes	Total IO	CPU (User, Krnl) [HZ]	(Name)
[26359]	1009	0	0	0	47 (46, 1)	(cputest2)
[26367]	1009	0	0	0	42 (42, 0)	(cputest2)
[26354]	1043	0	2473454	2473454	92 (36, 56)	(iotest2)
[26358]	1043	0	2444414	2444414	91 (40, 51)	(iotest2)
[26356]	1043	0	2383429	2383429	92 (52, 40)	(iotest2)
[26362]	1043	0	2396080	2396080	90 (56, 34)	(iotest2)
[26364]	1043	0	2462618	2462618	92 (46, 46)	(iotest2)
[26366]	1043	0	2458232	2458232	91 (36, 55)	(iotest2)
[26360]	1043	0	2336159	2336159	87 (65, 22)	(iotest2)
[26361]	1009	0	0	0	43 (43, 0)	(cputest2)
[26357]	1009	0	0	0	45 (45, 0)	(cputest2)
[26363]	1009	0	0	0	46 (46, 0)	(cputest2)
[26355]	1009	0	0	0	45 (45, 0)	(cputest2)
[26365]	1009	0	0	0	46 (46, 0)	(cputest2)
[26359]	1009	0	0	0	46 (46, 0)	(cputest2)
[26367]	1009	0	0	0	45 (45, 0)	(cputest2)
[26354]	1043	0	2294967	2294967	86 (67, 19)	(iotest2)
[26358]	1043	0	2365112	2365112	89 (65, 24)	(iotest2)
[26356]	1043	0	2287782	2287782	88 (58, 30)	(iotest2)
[26362]	1043	0	2292739	2292739	87 (65, 22)	(iotest2)

Report 1 – Cis 415
By: Matt Almenshad

Assistance

I discussed concepts and ideas of implementations with a number of students including Samuel Cordes, Emmalie Dion, Andrew Gao, Ian Garrett, Jennifer Horn, Robert Husbands, Matthew Jagielski, Jacob Lin, Timbwaoga Ouermi, Rui Tu. We STRICTLY discussed concepts and ideas, as well as helped each other get unstuck. We did not collude. Our implementations are completely different.
I would also like to give credit to Matthew Jagielski for giving me the idea of using logs to make the priority factor more effective. I struggled finding the right way of constructing a ratio and could not have done it without his mathematical fluency.

Workloads used

(*note: printer is merely a printing countdown basic loop that sleeps for a second between each count)
(*note2: sPrinter is a silent version of the same program)

```
./workload/printer1 5
./workload/printer2 5
./workload/printer3 5
./workload/printer4 5
```

```
./workload/sPrinter1 5
./workload/sPrinter2 5
./workload/sPrinter3 5
./workload/sPrinter4 5
```

```
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
./workload/iotest2 -b 1024 -n 1000000000
./workload/cputest2 -b 1000 -n 10000
```

```
./workload/iotest2
./workload/cputest2
./workload/iotest2
./workload/cputest2
./workload/iotest2
./workload/cputest2
./workload/iotest2
./workload/cputest2
./workload/iotest2
./workload/cputest2
./workload/iotest2
./workload/cputest2
./workload/iotest2
./workload/cputest2
```