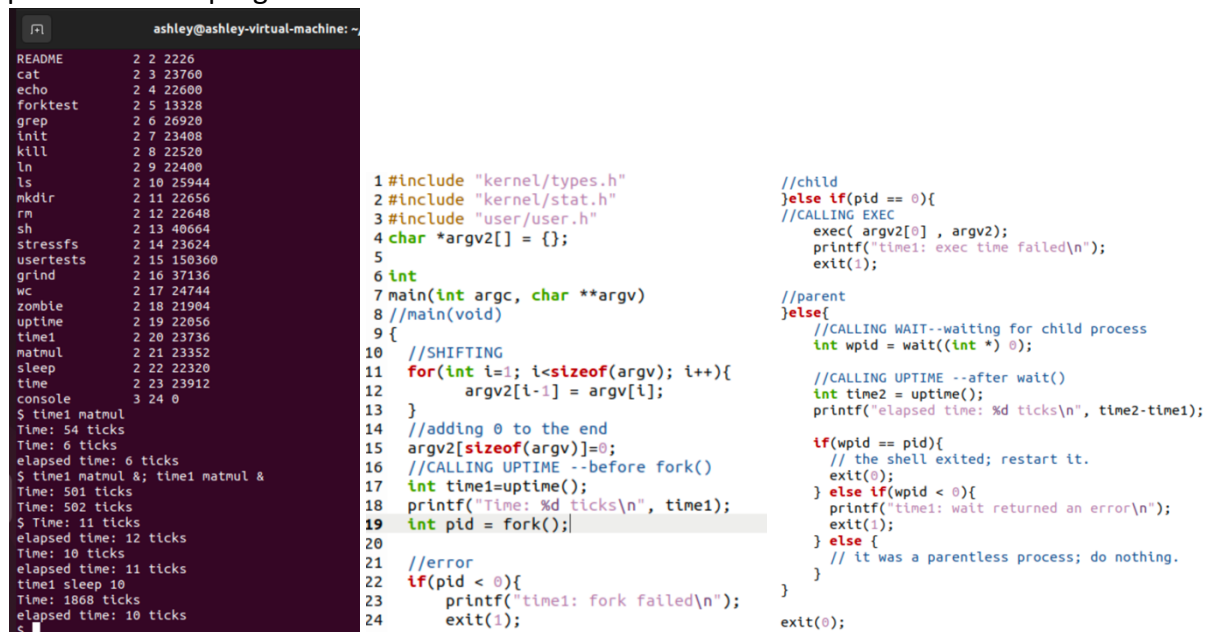


HW 2: Time Command

Task 1. Implement a time1 command that reports elapsed time.

Implementing the time1 command involved making a new c file in the user directory called 'time1' and as mentioned in the instructions also add the command to the Makefile. This file needs to be able to have access to types.h, stat.h and of course user.h files, therefore I included these on the top of my file. Then I approached the task by utilizing an empty array which I called 'argv2' this array was going to allow me to 'read' the arguments passed in by the user. When running the command time1, other arguments are passed as well, like matmul or sleep, therefore we need to have access to those as well. That is why I 'read' the array 'argv' which holds those other arguments but since 'time1' is held in index 0 and it was already run that arguments needs to be taken out. That is why array argv2 is used to 'shift' the array argv left and 'remove' time1 from the arguments needed. This is done with a for-loop that traverses argv and adds to argv2. Lastly, in order to reach the end of the arguments a '0' needs to be placed at the end. I did this just by setting the index corresponding to the argv2 array length to a zero. Then I followed the instructions to get the time before forking and then call exec() to run the rest processes. Then I used wait() for the child process and use that time now to get the elapsed time of the process. The elapsed time is the current time minus the past time that was gotten before forking. The times were gotten using the already give uptime() method. Then the rest are just some 'error' messages I included using if statements to make sure nothing failed in the procedure.

The following image showcases the result of running 'time1' command along with some portions of the program.



```
ashley@ashley-virtual-machine: ~
$ time1 matmul
Time: 54 ticks
Time: 6 ticks
elapsed time: 6 ticks
$ time1 matmul 8; time1 matmul 8
Time: 501 ticks
Time: 502 ticks
elapsed time: 12 ticks
Time: 10 ticks
elapsed time: 11 ticks
time1 sleep 10
Time: 1868 ticks
elapsed time: 10 ticks
$
```

```
1 #include "kernel/types.h"
2 #include "kernel/stat.h"
3 #include "user/user.h"
4 char *argv2[] = {};
5
6 int
7 main(int argc, char **argv)
8 //main(void)
9 {
10     //SHIFTING
11     for(int i=1; i<sizeof(argv); i++){
12         argv2[i-1] = argv[i];
13     }
14     //adding 0 to the end
15     argv2[sizeof(argv)] = 0;
16     //CALLING UPTIME --before fork()
17     int time1=uptime();
18     printf("Time: %d ticks\n", time1);
19     int pid = fork();
20
21     //error
22     if(pid < 0){
23         printf("time1: fork failed\n");
24         exit(1);
25     }
26
27     //child
28     }else if(pid == 0){
29         //CALLING EXEC
30         exec( argv2[0] , argv2);
31         printf("time1: exec time failed\n");
32         exit(1);
33     }
34
35     //parent
36     }else{
37         //CALLING WAIT--waiting for child process
38         int wpid = wait((int *) 0);
39
40         //CALLING UPTIME --after wait()
41         int time2 = uptime();
42         printf("elapsed time: %d ticks\n", time2-time1);
43
44         if(wpid == pid){
45             // the shell exited; restart it.
46             exit(0);
47         } else if(wpid < 0){
48             printf("time1: wait returned an error\n");
49             exit(1);
50         } else {
51             // it was a parentless process; do nothing.
52         }
53     }
54
55     exit(0);
56 }
```

*** Professor Moore added some additional Time printing statements on the 'Matmul' file therefore there are some additional prints, however I let Dr.Moore know and she said it was okay to leave them there

This task helped me gain a better understanding on how commands are programed and how there are different factors to take into consideration. Not only do we have to program the specific purpose for the command, in this case 'counting time' but also, we need to handle and manage the 'errors' that child processes can have if something unusual happens whit the other processes ran like forking and exec.

This task wasn't that complicated, however starting and understanding specifically what the command has to do is what is difficult. In this case I knew that I had to find a way to process the rest of the arguments other than just time1 but I also had to take care of the child process. This was mainly the issue I ran into but after further explanation and clarifications from Dr.Moore I was able of completing it.

Task 2. Keep track of how much cputime a process has used.

To track the cpu time I had to some modifications on various files. First, I added the 'cputime' field to the structure proc in the proc.c file allocated in the kernel directory. This filed will hold the cputime value. Then I initialized this filed inside the allocproc() function inside the proc.c file found in the kernel directory. This value is initialized to zero. Then this field was actually put to use on the trap.c file. In the methods usertrap() and kerneltrap() everytime the process uses up its time slice the cputime is incremented. This is done after the yield method call.

The following are some snippets of the code that I ended up with after completing task two of this homework assignment.

```

104 // If there are no free procs, or a memory trap
105 static struct proc*
106 allocproc(void)
107 {
108     struct proc *p;
109     for(p = proc; p < &proc[NPROC]; p++) {
110         acquire(&p->lock);
111         if(p->state == UNUSED) {
112             goto found;
113         } else {
114             release(&p->lock);
115         }
116     }
117     return 0;
118 }
119 found:
120 p->pid = allocpid();
121 p->state = USED;
122 //initializing cputime
123 p->cputime = 0;
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

This task helped me understand how file directories work and how pointers are a good way of being able to have access to fields and methods. All files being handled were located in the kernel directory but they were all in different files therefore I had to use the '->' symbol to be able to modify the cputime field. Therefore, I got a better overall understanding in c programming language as well.

This task was pretty easy, the only hard thing was basically finding the different files. In addition I did struggle with the incrementation of the cputime because I wasn't sure of where to place it but then Dr.Moore explained that it had to go before the yield method call.

Task 3. Implement a `wait2()` system call that waits for a child to exit and returns the child's status and rusage.

To implement the `wait2()` method I had to make some modifications on various files as well, but I also had to make a new file. I made file `pstat.h` inside the kernel directory in where I declared the new structure `'rusage'` that contains the `cputime` field. In order to use the struct `'rusage'` I had to add it into the `user.h` file first. Then I started with the `wait2` method, I first made its argument declaration also in the `user.h` file. The method declaration relied on one integer pointer and a struct pointer. The `int` handles the address of the process being held by the wait two, while the struct pointer handles the `'rusage'`. In order to avoid any future errors when trying to access the new wait method being created I added its corresponding information to the files, `syscall.h` and `syscall.c` both located in the kernel directory. This was just a static `uint64` system call and declaring the `sys_wait` call. Then in the `syscall.h` we define the system call number. In `sysproc`, I implemented the `sys_wait2()` function, this is the function that calls the actual `wait2` function. I did this by copying and pasting the already given `wait` function and then modified it for the additional argument taken into by `wait2`. I did this by also guiding myself from the already built `if` statement that checks if the argument is correctly located in memory. Instead of just having one statement we need two places in memory. `Index 0` and `index 1`, where `index 1` holds the `rusage`. Finally, in `proc.c` we implement the `wait2()` function. For this function I also copied and pasted the method for `wait` since I now only had to add the implementation to utilize the `'rusage'`. I had to return not only the child process state/status but also the `cputime` which as mentioned earlier is held in the `rusage` structure. I first added the new `uint64` `addr2` parameter, since that's where `rusage` will get passed. Then I created a new `rusage` structure called `'cru'`, which is the child `rusage`. Then I added the line `cru.cputime=np->cputime`. This was done in order to have access using the new `'cru'` field created. Then I used the `if` statement that already copied from the kernel into the user the status of the child and modified it to copy the `rusage cputime`. Instead of copying address one I copied address two and passed `cru` for the other statements.

The following are some portions of code from this task.

```

433 int
434 wait2(uint64 addr, uint64 addr2)
435 {
436
437     struct proc *np;
438     int havekids, pid;
439     struct proc *p = myproc();
440
441     struct rusage cru;
442
443     acquire(&wait_lock);
444
445     for(;;){
446         // Scan through table looking for exited children.
447         havekids = 0;
448         for(np = proc; np < &proc[NPROC]; np++){
449             if(np->state == p){
450                 // make sure the child isn't still in exit() or swtch().
451                 acquire(&np->lock);
452
453                 havekids = 1;
454                 if(np->xstate == ZOMBIE){
455                     // Found one.
456                     pid = np->pid;
457                     cru.cputime = np->cputime;
458                     //copying status
459                     if(addr != 0 && copyout(p->pagetable, addr, (char *)&np->xstate,
460                                             sizeof(np->xstate)) < 0) {
461                         release(&np->lock);
462                         release(&wait_lock);
463                         return -1;
464                     }
465                 }
466             }
467         }
468     }
469 }

```

This task allowed me to understand the already built in wait process because in order to implement wait2 I had to go through wait() and analyze its behavior. I also learned that pieces of code are sharable and many functions have similar characteristics, but there are also some code lines that are uniquely built for each function.

I had a hard time figuring out how to obtain the child process cputime based on what was given and from all the modifications that I had done on the other files. However, professor Moore helped me understand how the copyout() method worked and how this allowed me to use addresses in order to copy from memory in the kernel to memory in the user end files.

Task 4. Implement a time command that runs the command given to it as an argument and outputs elapsed time, CPU time, and %CPU used.

The time command was implemented by creating a new c file called 'time' in the user folder. This was easy since I had already done time1 in the first task of this homework so I just copied that into the new file. Instead of using wait() in this command I had to utilize wait2() which I had previously built. Therefore I just did minor modifications by adding a new structure rusage caller r. I used this structure to pass the address into the second parameter of the wait2 function. Then I added the print statement that showcased the elapsed time again, but also the cpu time and the CPU perctange. The cpu time was obtained through the 'r' rusage structure pointer, and the percentage was gotten by multiplying the cputime times a hundred and then dividing that by the elapsed time.

```

1 #include "kernel/types.h"
2 #include "kernel/stat.h"
3 #include "user/user.h"
4 #include "kernel/pstat.h"
5 char *argv1[] = {};
6 struct rusage r;
7
8 int
9 main(int argc, char **argv)
10 //main(void)
11 {
12     //SHIFTING
13     for(int i=1; i<sizeof(argv); i++){
14         argv1[i-1] = argv[i];
15     }
16     //adding 0 to the end
17     argv1[sizeof(argv)] = 0;
18     //CALLING UPTIME --before fork()
19     int time1=uptime();
20     int pid = fork();
21
22     if(pid == 0){
23         //CALLING EXEC
24         exec(argv1[0], argv1);
25         printf("time1: exec time failed\n");
26         exit(1);
27     }
28     //parent
29     else{
30         //CALLING WAIT--waiting for chld process
31         int wpid = wait2((int *) 0, &r);
32
33         //CALLING UPTIME --after wait()
34         int time2 = uptime();
35         int elapsedTime = time2-time1;
36         //Printing the cpu time and details
37         printf("elapsed time: %d ticks, cpu time: %d ticks, %d% CPU\n", elapsedTime,
38                r.cputime, (r.cputime*100)/elapsedTime);
39     }
40     if(wpid == pid){
41         // the shell exited; restart it.
42         exit(0);
43     } else if(wpid < 0){
44         printf("Error: wait returned an error!\n");
45     }
46 }

```

These are the results of the time command.

```

ashley@ashley-virtual-machine: ~/Documents/xv6-riscv-labs...
riscv64-linux-gnu-objdump -t user/_time | sed '1,/SYMBOL
/d' > user/time.sym
mkfs/mkfs fs.img README user/_cat user/_echo user/_forkte
t user/_kill user/_ln user/_ls user/_mkdir user/_rm user/
/_usertests user/_grind user/_wc user/_zombie user/_uptime
mul user/_sleep user/_time
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bit
4 total 1000
ballocc: first 708 blocks have been allocated
ballocc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kern
1 -nographic -drive file=fs.img,if=none,format=raw,id=x0
ce,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ time matmul
Time: 6 ticks
elapsed time: 6 ticks, cpu time: 6 ticks, 100% CPU
$ time matmul & ; time matmul &
$ Time: 11 ticks
elapsed time: 12 ticks, cpu time: 6 ticks, 50% CPU
Time: 10 ticks
elapsed time: 11 ticks, cpu time: 6 ticks, 54% CPU
time sleep 10
elapsed time: 10 ticks, cpu time: 0 ticks, 0% CPU
$

```

Extra Credit (5 points). Discuss limitations of our time command. (Hint: For one limitation, consider what would happen if the command that is being timed forks child processes).

As mentioned, one limitation in my time command is the issue that in the case that a child process is forked while being times this will impact the cpu time and elapsed time as a whole. This would happen because both the parent and the child process will continue running the same program because they both called it. The time is affected and therefore it is not fully accurate.