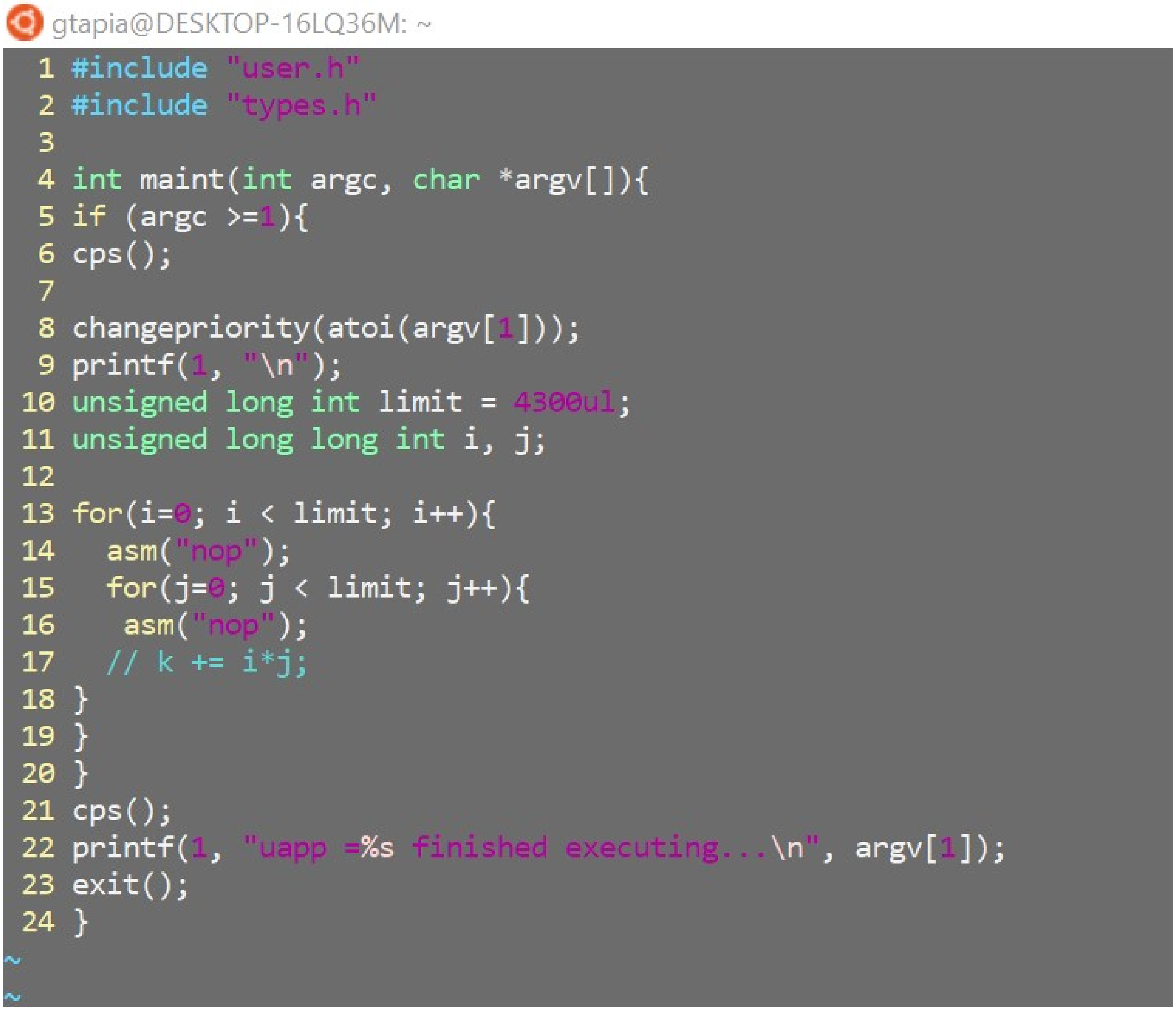
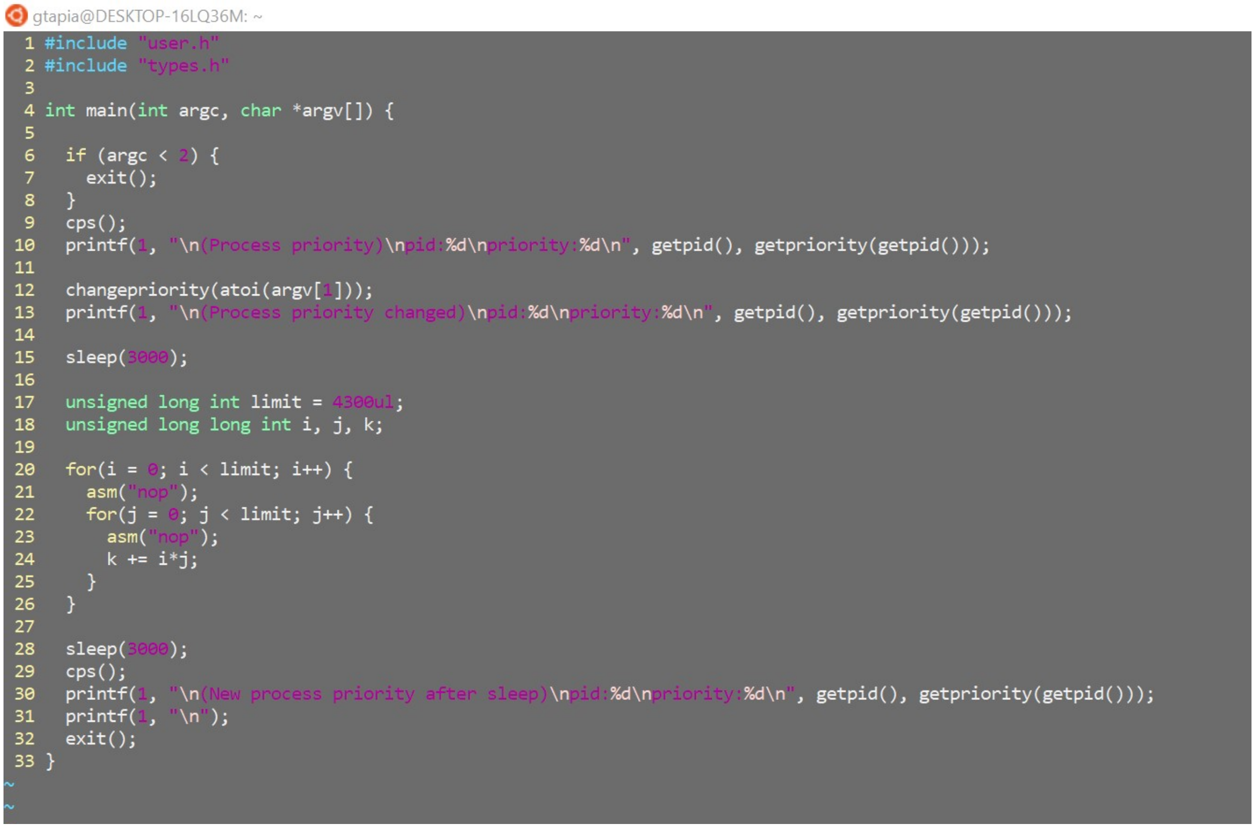


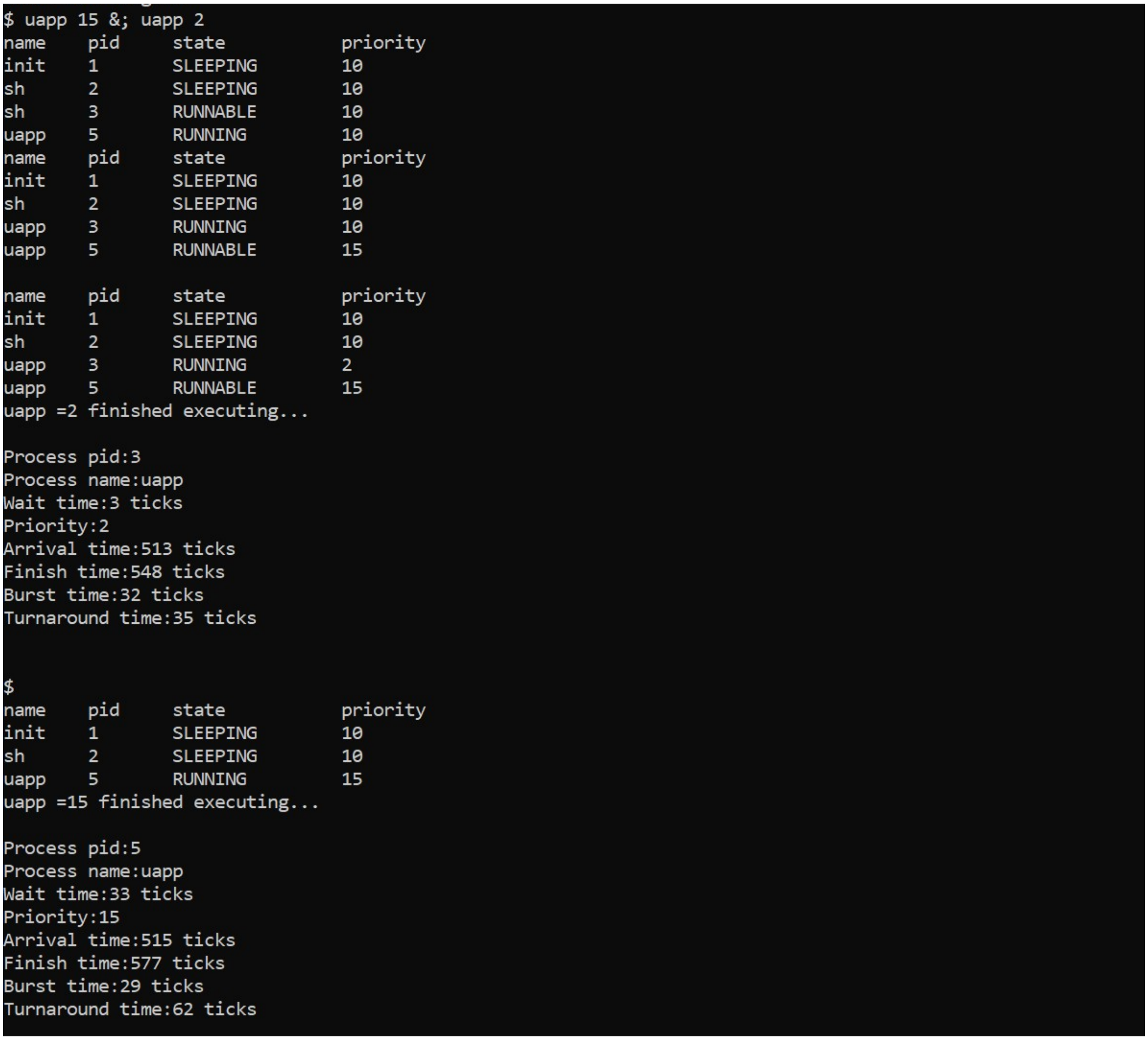
gtapia@DESKTOP-16LQ36M: ~

```
#include "user.h"
#include "types.h"

int maint(int argc, char *argv[]){
if (argc >=1){
cps();

changepriority(atoi(argv[1]));
printf(1, "\n");
unsigned long int limit = 4300ul;
unsigned long long int i, j;

for(i=0; i < limit; i++){
  asm("nop");
  for(j=0; j < limit; j++){
   asm("nop");
  // k += i*j;
}
}
}
cps();
printf(1, "uapp =%s finished executing...\n", argv[1]);
exit();
}
~
~
```



```
#include "user.h"
#include "types.h"

int main(int argc, char *argv[]) {

  if (argc < 2) {
    exit();
  }
  cps();
  printf(1, "\n(Process priority)\npid:%d\npriority:%d\n", getpid(), getpriority(getpid()));

  changepriority(atoi(argv[1]));
  printf(1, "\n(Process priority changed)\npid:%d\npriority:%d\n", getpid(), getpriority(getpid()));

  sleep(3000);

  unsigned long int limit = 4300ul;
  unsigned long long int i, j, k;

  for(i = 0; i < limit; i++) {
    asm("nop");
    for(j = 0; j < limit; j++) {
      asm("nop");
      k += i*j;
    }
  }

  sleep(3000);
  cps();
  printf(1, "\n(New process priority after sleep)\npid:%d\npriority:%d\n", getpid(), getpriority(getpid()));
  printf(1, "\n");
  exit();
}
~
~
```

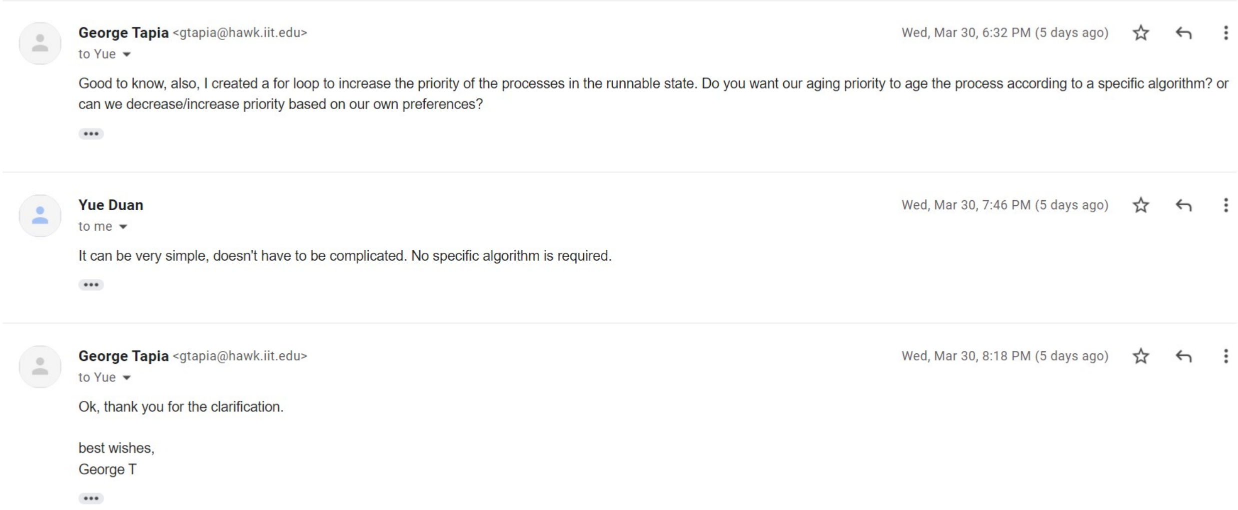
Test function that professor provided with added functionality. Including the printout of processes, and taking an argument to change the priority of the currently running process. Did not demonstrate getpriority because it is not specified to use in test program, but it does work. Code is in this document. Also, please email me if you prefer a different form of submission to make it easier on you, code explanations are below as well.



```
$ uapp 15 &; uapp 2
name    pid     state           priority
init    1       SLEEPING        10
sh      2       SLEEPING        10
sh      3       RUNNABLE        10
uapp    5       RUNNING         10
name    pid     state           priority
init    1       SLEEPING        10
sh      2       SLEEPING        10
uapp    3       RUNNING         10
uapp    5       RUNNABLE        15

name    pid     state           priority
init    1       SLEEPING        10
sh      2       SLEEPING        10
uapp    3       RUNNING         2
uapp    5       RUNNABLE        15
uapp =2 finished executing...

Process pid:3
Process name:uapp
Wait time:3 ticks
Priority:2
Arrival time:513 ticks
Finish time:548 ticks
Burst time:32 ticks
Turnaround time:35 ticks


$
name    pid     state           priority
init    1       SLEEPING        10
sh      2       SLEEPING        10
uapp    5       RUNNING         15
uapp =15 finished executing...

Process pid:5
Process name:uapp
Wait time:33 ticks
Priority:15
Arrival time:515 ticks
Finish time:577 ticks
Burst time:29 ticks
Turnaround time:62 ticks
```

As you can see, first process (uapp 15- pid 5 - priority 15) runs, but because (uapp 2- pid 3 - priority 2) is running too. The context switch occurs, (uapp 15-pid 5 -priority 15) switches from running to runnable, and (uapp 2-pid 3 -priority 2) switches from runnable to running, so uapp 2 finishes first because it has higher priority.

The next output below, demonstrates aging of processes. The professor mentioned that no specific algorithm is required that it can be very simple. I decreased the processes priority for processes running on the cpu. I also added functionality to decrease priority if the process is in runnable state. (no specific algorithm is required, as professor mentioned. Code is in the documentation.



George Tapia <gtapia@hawk.iit.edu>
to Yue
Wed, Mar 30, 6:32 PM (5 days ago)

Good to know, also, I created a for loop to increase the priority of the processes in the runnable state. Do you want our aging priority to age the process according to a specific algorithm? or can we decrease/increase priority based on our own preferences?

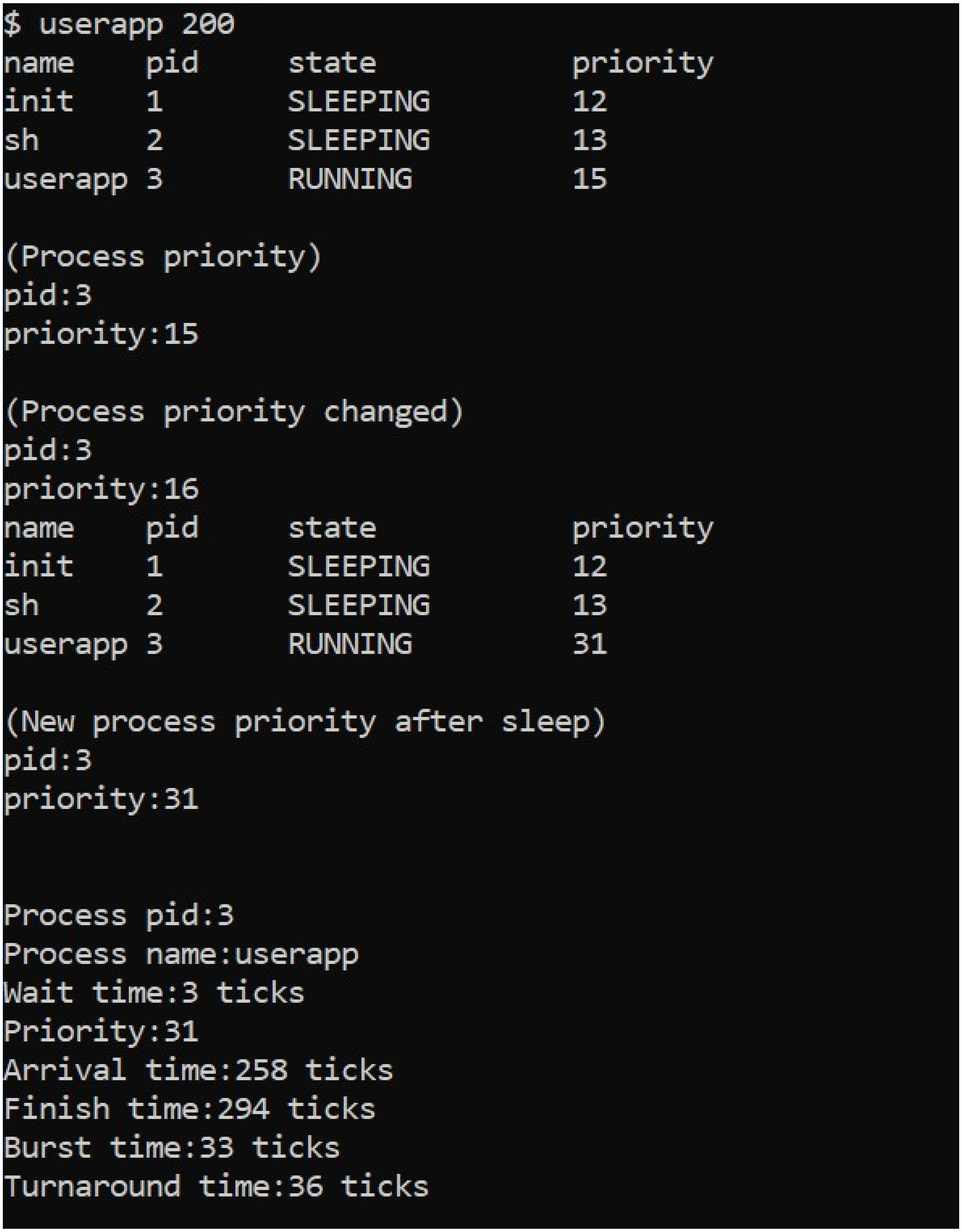
Yue Duan
to me
Wed, Mar 30, 7:46 PM (5 days ago)

It can be very simple, doesn't have to be complicated. No specific algorithm is required.

George Tapia <gtapia@hawk.iit.edu>
to Yue
Wed, Mar 30, 8:18 PM (5 days ago)

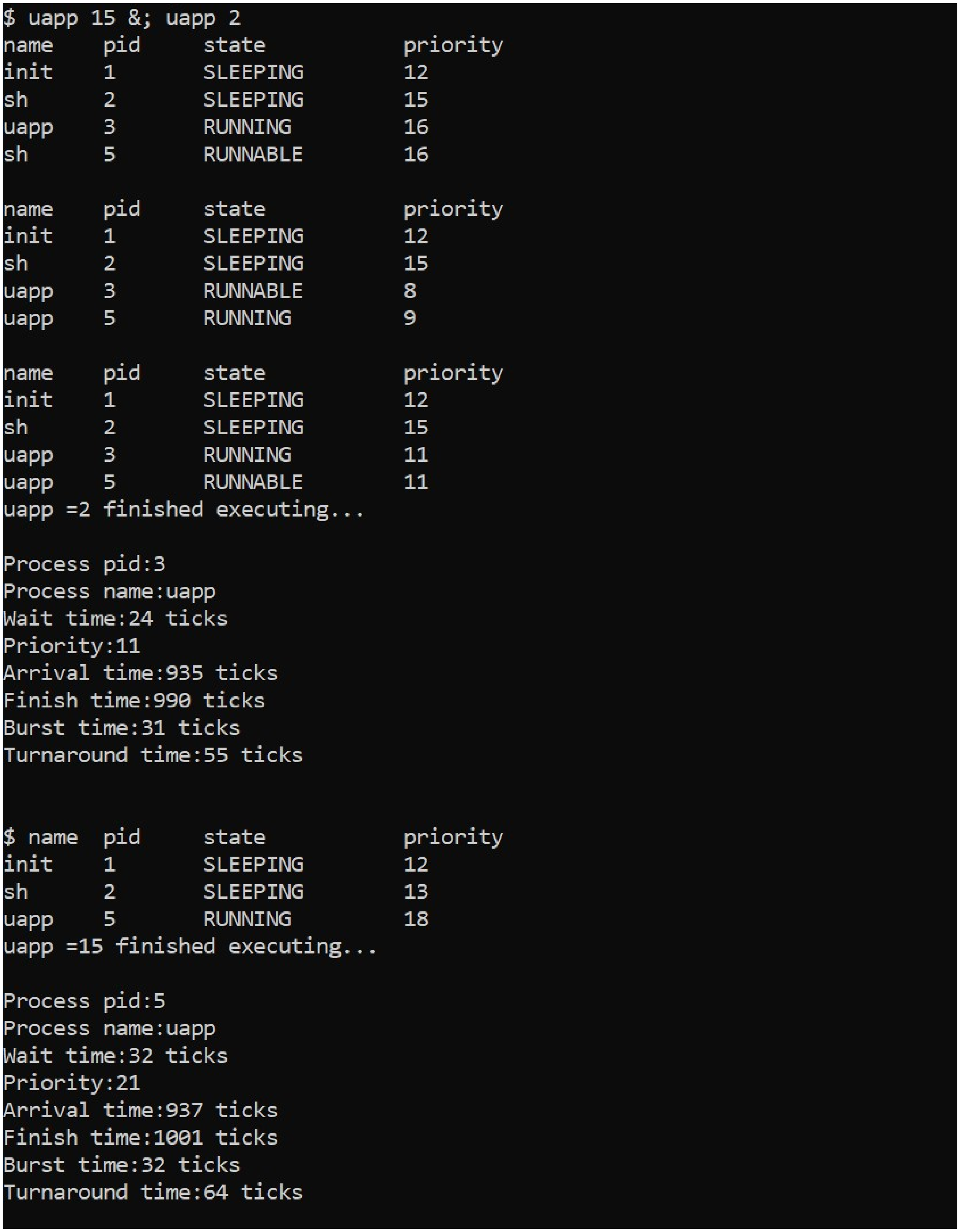
Ok, thank you for the clarification.

best wishes,
George T



```
$ userapp 200
name    pid     state           priority
init    1       SLEEPING        12
sh      2       SLEEPING        13
userapp 3       RUNNING         15

(Process priority)
pid:3
priority:15

(Process priority changed)
pid:3
priority:16
name    pid     state           priority
init    1       SLEEPING        12
sh      2       SLEEPING        13
userapp 3       RUNNING         31

(New process priority after sleep)
pid:3
priority:31


Process pid:3
Process name:userapp
Wait time:3 ticks
Priority:31
Arrival time:258 ticks
Finish time:294 ticks
Burst time:33 ticks
Turnaround time:36 ticks
```

As you can see, process priority is aging when it is running for a while on the cpu(decrease in priority).



```
$ uapp 15 &; uapp 2
name    pid     state           priority
init    1       SLEEPING        12
sh      2       SLEEPING        15
uapp    3       RUNNING         16
sh      5       RUNNABLE        16

name    pid     state           priority
init    1       SLEEPING        12
sh      2       SLEEPING        15
uapp    3       RUNNABLE        8
uapp    5       RUNNING         9

name    pid     state           priority
init    1       SLEEPING        12
sh      2       SLEEPING        15
uapp    3       RUNNING         11
uapp    5       RUNNABLE        11
uapp =2 finished executing...

Process pid:3
Process name:uapp
Wait time:24 ticks
Priority:11
Arrival time:935 ticks
Finish time:990 ticks
Burst time:31 ticks
Turnaround time:55 ticks


$ name  pid     state           priority
init    1       SLEEPING        12
sh      2       SLEEPING        13
uapp    5       RUNNING         18
uapp =15 finished executing...

Process pid:5
Process name:uapp
Wait time:32 ticks
Priority:21
Arrival time:937 ticks
Finish time:1001 ticks
Burst time:32 ticks
Turnaround time:64 ticks
```

Here, process in runnable state is aging as well, but in terms of decreasing and increasing. Increasing priority while in runnable state, decreasing while running on cpu.

```
diff -r xv6-original/Makefile.h xv6-scheduling/Makefile.h
---

>       _userapp\
>       _uapp\

diff -r xv6-original/defs.h xv6-scheduling/defs.h
123c123,125
<
---
> int           cps(void);
> int           changepriority(int);
> int           getpriority();


diff -r xv6-original/proc.c xv6-scheduling/proc.c
<
—
```

(added to allocproc to initialize allocated processes with these values)

```
>   p->priority = 10; //setting the priority of the process
>   p->arrival_time = ticks;
>   p->finish_time = 0;
>   p->running_time = 0; (running_time = burst time)
>   p->ticks0 = 0;
114c118
<
---
>
218c222
<
—
```

(updating exit function to print out scheduling performance as professor instructed)

(running_time is burst time (time taken for process execution on cpu))

```
>   np->priority = curproc->priority; //inheriting its parent's priority
262a267,281
>   curproc->finish_time = ticks;
>  //wait time = turnaround time (finish - start)  - burst time (burst time = running_time)
>   int waittime = curproc->finish_time - curproc->arrival_time - curproc->running_time;
>
>   //to avoid printing performance status of shell
>   if(strncmp(curproc->name, "sh",2) != 0){
>   cprintf("\nProcess pid:%d\nProcess name:%s\nWait time:%d ticks\n", curproc->pid, curproc->name, waittime);
>   cprintf("Priority:%d\n",curproc->priority);
>   cprintf("Arrival time:%d ticks\n",curproc->arrival_time);
>   cprintf("Finish time:%d ticks\n",curproc->finish_time);
>   cprintf("Burst time:%d ticks\n", curproc->running_time);
>   cprintf("Turnaround time:%d ticks\n\n\n",curproc->finish_time - curproc->arrival_time);
> }
>
>
325c344
```

(following are updates to scheduler(void) in proc.c)

```
<   struct proc *p;
---
>   struct proc *p, *p2; // proc pointers
326a346
>
332c352,353
<
---
>
>     struct proc *max_priority_process;
333a355
>     // finding the process with the highest priority in the ptable (the lower the value the higher the priority)
336c358
<       if(p->state != RUNNABLE)
---
>       if(p->state != RUNNABLE)// if it is not in runnable state, then we keep looking to the next processes
338c360,378
<
---
>       max_priority_process = p;
>       for(p2 = ptable.proc; p2 < &ptable.proc[NPROC]; p2++){
>         if (p2->state != RUNNABLE)
>            continue;
>         if(max_priority_process->priority > p2->priority)
>            max_priority_process = p2;
>        }
>
```

(**Implemented aging for processes and a safeguard to make sure priority doesn’t go over 31)(stays within range)**

```
>     // increasing priority of the other processes that are in the runnable state
>      for(p2= ptable.proc; p2 < &ptable.proc[NPROC]; p2++){
>         if(p2->priority > 31){
>         p2->priority = 31;
>             }
>
>         if(p2->priority > 0 && p2->state == RUNNABLE && p2 != max_priority_process){
>        p2->priority--;}
>     }
>
>
342,346c382,386
```

(switching to higher priority process)

```
<      c->proc = p;
<      switchuvm(p);
<      p->state = RUNNING;
<
<      swtch(&(c->scheduler), p->context);
---
>      p = max_priority_process;
>      c->proc = p;//assigning the highest priority process
>      switchuvm(p);
>      p->state = RUNNING;   //change the process state to running
>      swtch(&(c->scheduler), p->context);
354c394
<
---
>    }
356d395
< }
389,390c428,429
<   myproc()->state = RUNNABLE;
<   sched();
---
>   myproc()->state = RUNNABLE;//changes the process state from running to runnable
>   sched();//calls the scheduler to schedule the highest priority process
465d503
< }
466a505
> }
533a573,618
> }
```

```
>
```

(added changepriority, getpriority and cps to proc.c)

```
> int
> changepriority(int priority)
> {
>  struct proc *p = myproc();//setting pointer to current process running on the cpu
>  acquire(&ptable.lock);
>  if(priority >= 0 && priority <= 31)
>      p->priority = priority;//change the current process priority running on the cpu
>  release(&ptable.lock);
>  yield();//give up the cpu and schedule next highest priority process. Yield calls sched.
>  return 0;
> }
> int
> getpriority()
> {
>  struct proc *p = myproc();//getting the current running process on the cpu
>  int priority;
>  acquire(&ptable.lock);//aquire the ptable lock
>  priority = p->priority;//we get the process priority
>  release(&ptable.lock);//release the lock
>  return priority;//return the value
> }
>
```

[raj-maurya/xv6-public_modifiedOS: XV6-OS (github.com)](https://github.com/raj-maurya/xv6-public_modifiedOS) added cps function to print out processes pid, state and priority, had to use off this reference. For test function.

```
> int
> cps()
> {
> struct proc *p;
>
> sti();
> acquire(&ptable.lock);
> cprintf("name\tpid\tstate\t\tpriority\n");
> for(p = ptable.proc; p < &ptable.proc[NPROC];p++)
> {
> if(p->state == SLEEPING)
>    cprintf("%s\t%d\tSLEEPING\t%d\n",p->name,p->pid,p->priority);
> else if(p->state == RUNNING)
>    cprintf("%s\t%d\tRUNNING\t\t%d\n",p->name,p->pid,p->priority);
> else if(p->state == RUNNABLE)
>    cprintf("%s\t%d\tRUNNABLE\t%d\n",p->name,p->pid,p->priority);
>
>
> }
> release(&ptable.lock);
> return 24;
>
Only in xv6-scheduling: proc.d
diff -r xv6-original/proc.h xv6-scheduling/proc.h
```

(added these to the proc struct so that each process can have these assigned)

```
51a52,58
>
>   uint priority; //priority value ranges from 0-31
>   uint arrival_time;
>   uint finish_time;
>   uint running_time;
```

```
diff -r xv6-original/syscall.c xv6-scheduling/syscall.c
```

(**added function definitions to syscall.c and argument associaters with system calls)**

```
> extern int sys_changepriority(void);
> extern int sys_getpriority(void);
> extern int sys_cps(void);

> [SYS_changepriority] sys_changepriority,
> [SYS_getpriority] sys_getpriority,
> [SYS_cps] sys_cps,
```

```
diff -r xv6-original/syscall.h xv6-scheduling/syscall.h
```

(**numbers associated with system call functions added)**

```
> #define SYS_changepriority 22
> #define SYS_getpriority 23
> #define SYS_cps 24
Only in xv6-scheduling: syscall.o
Only in xv6-scheduling: sysfile.d
Only in xv6-scheduling: sysfile.o
```

```
diff -r xv6-original/sysproc.c xv6-scheduling/sysproc.c
68a69,72
>   myproc()->sleep_time=n;
>   myproc()->ticks0 = ticks0;
>
>
74a79
>     break;
```

```
90a96,114
> }
>
> int
> sys_changepriority(void){
>   int priority;
>   if(argint(0, &priority) < 0)
>     return -1;
>   if(priority < 0 || priority > 31)
>     return -1;
>   return changepriority(priority);
> }
> int
> sys_getpriority(void){
>   return getpriority();;
> }
>
> int
> sys_cps(void){
> return cps();
Only in xv6-scheduling: sysproc.d
Only in xv6-scheduling: sysproc.o
diff -r xv6-original/trap.c xv6-scheduling/trap.c
```

(**keeping track of running time in trap.c and aging of processes running in the cpu)**

```
106c106,111
<       tf->trapno == T_IRQ0+IRQ_TIMER)
---
>       tf->trapno == T_IRQ0+IRQ_TIMER){
>       myproc()->running_time++;
>  if(myproc()->priority >= 0 || myproc()->priority < 31){
>    myproc()->priority++;}
>

diff -r xv6-original/user.h xv6-scheduling/user.h
0a1
```

— **(function definitions)**

```
> int changepriority(int);
> int getpriority();
> int cps(void);
```

(adding calls)

```
diff -r xv6-original/usys.S xv6-scheduling/usys.S
31a32,34
```

```
> SYSCALL(changepriority)
> SYSCALL(getpriority)
> SYSCALL(cps)
```