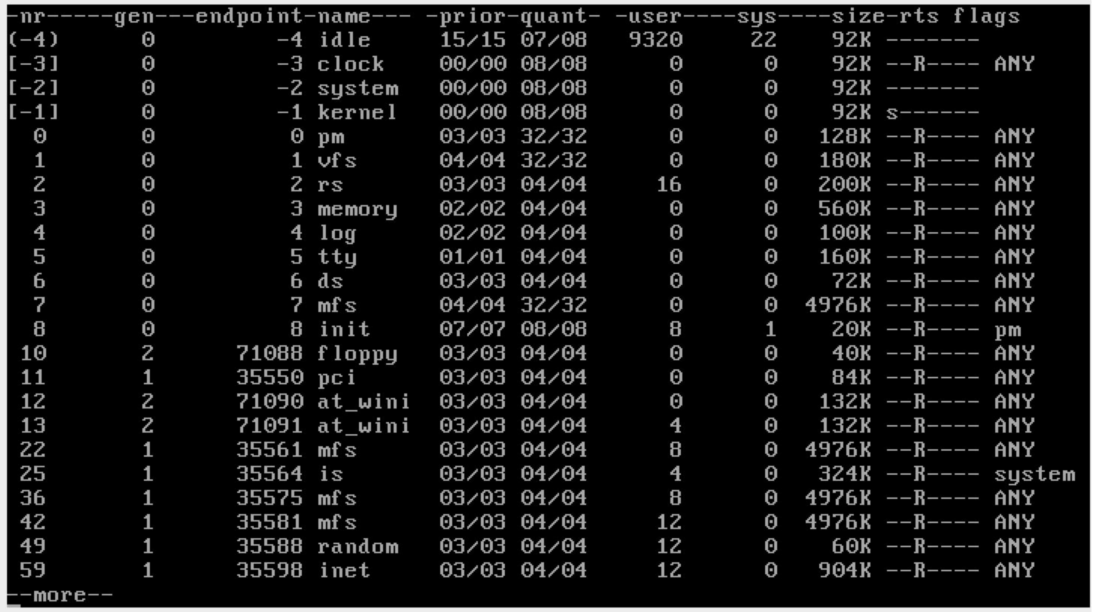
Ryan Citron

This homework took a very long time. I was very confused at first on how everything needed to work together. Once I started to understand it got a lot easier. I added a new variable called in proc.h p_cpu_time to keep track of each processes cpu time. To actually keep track of the variable, in proc.c I used the sched function. Everytime a quantum was fully used, I added the size of that quantum to the p_cpu_time. After a count of 50000 occured, I took the values of each processes p_cpu_time and cut it in half. This allowed me to see which processes were recently used easier. In the pick_proc function, I added the way it schedules the lowest cpu time for a user process. I have a base number of 5000 which the way I keep track of numbers that is the largest the “smallest” amount of cpu time a process could ever reach. I check if the current process, rp, is a user process. If it is a user process I change the quan_min_time to that cpu time rather than the base of 5000. After that I set my next_ptr to be rp which is the user process. If it is not a user process it runs like normal.

To test my code, I modified dmp_kernel.c. The function I modified within that file was proctab_dmp. The reason why I chose this file is because it was already all set up for me. All I had to do was change the total ticks for the users into my p_cpu_time. When F1 is hit you are able to see the proctab_dmp. This shows the name, priority, proc number, total quantums per process, and my time which is under the User section. I can watch the numbers go up and after refreshing with hitting F1 again, it is noticeable the less to no use processes were at 0 or approaching 0. The ones that were being used the most always had a value within it. To see if the numbers were keeping up, if I hit F1 I can see idle rising and falling, but the user processes are keeping their time. This shows that the processes are getting routed to the front of the queue and keep getting their quantum time refreshed. The image below shows a high process time for idle while the user times are keep a constant quantum time. This is what I was testing for and this is how the code is written. The way I interpret the question I have this completely working.



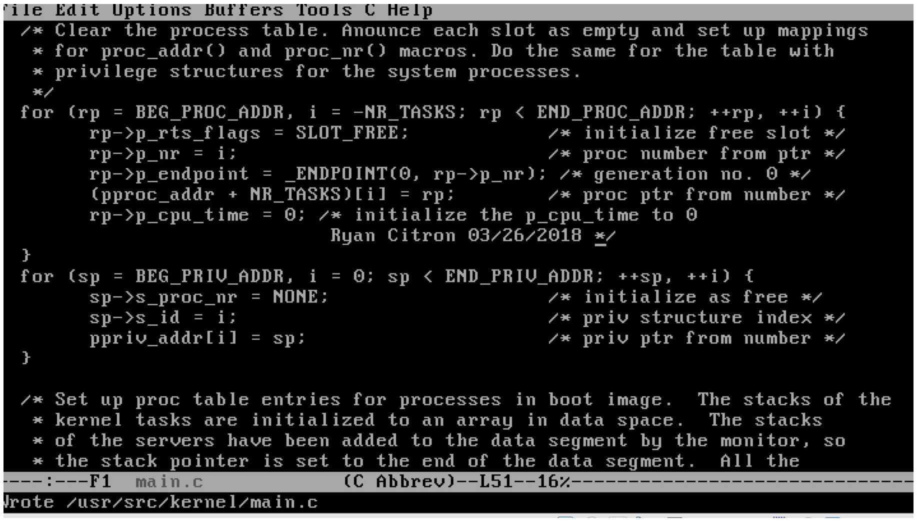
```
-nr-----gen---endpoint-name--- -prior-quant- -user----sys----size-rts flags
(-4)      0        -4 idle     15/15 07/08   9320     22     92K -------
[-3]      0        -3 clock    00/00 08/08      0      0     92K --R---- ANY
[-2]      0        -2 system   00/00 08/08      0      0     92K -------
[-1]      0        -1 kernel   00/00 08/08      0      0     92K s------
  0       0         0 pm       03/03 32/32      0      0    128K --R---- ANY
  1       0         1 vfs      04/04 32/32      0      0    180K --R---- ANY
  2       0         2 rs       03/03 04/04     16      0    200K --R---- ANY
  3       0         3 memory   02/02 04/04      0      0    560K --R---- ANY
  4       0         4 log      02/02 04/04      0      0    100K --R---- ANY
  5       0         5 tty      01/01 04/04      0      0    160K --R---- ANY
  6       0         6 ds       03/03 04/04      0      0     72K --R---- ANY
  7       0         7 mfs      04/04 32/32      0      0   4976K --R---- ANY
  8       0         8 init     07/07 08/08      8      1     20K --R---- pm
 10       2     71088 floppy   03/03 04/04      0      0     40K --R---- ANY
 11       1     35550 pci      03/03 04/04      0      0     84K --R---- ANY
 12       2     71090 at_wini  03/03 04/04      0      0    132K --R---- ANY
 13       2     71091 at_wini  03/03 04/04      4      0    132K --R---- ANY
 22       1     35561 mfs      03/03 04/04      8      0   4976K --R---- ANY
 25       1     35564 is       03/03 04/04      4      0    324K --R---- system
 36       1     35575 mfs      03/03 04/04      8      0   4976K --R---- ANY
 42       1     35581 mfs      03/03 04/04     12      0   4976K --R---- ANY
 49       1     35588 random   03/03 04/04     12      0     60K --R---- ANY
 59       1     35598 inet     03/03 04/04     12      0    904K --R---- ANY
--more--
```

Ryan Citron

Changelog

In Kernel:
Main.c:

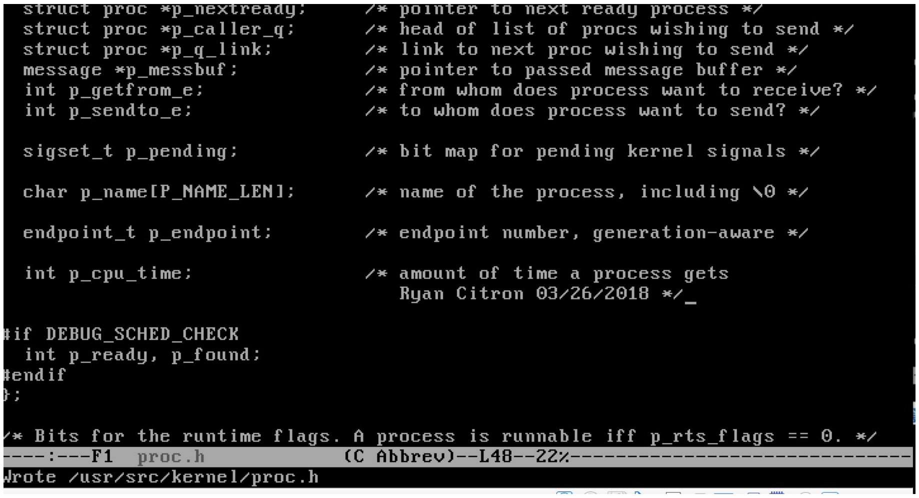
On line 50, added the like rp->p_cpu_time = 0 to initialize the p_cpu_time to 0.



```
File Edit Options Buffers Tools C Help
  /* Clear the process table. Anounce each slot as empty and set up mappings
   * for proc_addr() and proc_nr() macros. Do the same for the table with
   * privilege structures for the system processes.
   */
  for (rp = BEG_PROC_ADDR, i = -NR_TASKS; rp < END_PROC_ADDR; ++rp, ++i) {
        rp->p_rts_flags = SLOT_FREE;            /* initialize free slot */
        rp->p_nr = i;                           /* proc number from ptr */
        rp->p_endpoint = _ENDPOINT(0, rp->p_nr); /* generation no. 0 */
        (pproc_addr + NR_TASKS)[i] = rp;        /* proc ptr from number */
        rp->p_cpu_time = 0; /* initialize the p_cpu_time to 0
                             Ryan Citron 03/26/2018 */
  }
  for (sp = BEG_PRIV_ADDR, i = 0; sp < END_PRIV_ADDR; ++sp, ++i) {
        sp->s_proc_nr = NONE;                   /* initialize as free */
        sp->s_id = i;                           /* priv structure index */
        ppriv_addr[i] = sp;                     /* priv ptr from number */
  }

  /* Set up proc table entries for processes in boot image.  The stacks of the
   * kernel tasks are initialized to an array in data space.  The stacks
   * of the servers have been added to the data segment by the monitor, so
   * the stack pointer is set to the end of the data segment.  All the
----:---F1  main.c           (C Abbrev)--L51--16%----------------------------
Wrote /usr/src/kernel/main.c
```

Proc.h:

On line 47, added a counter p_cpu_time to count up the amount of time each process gets.



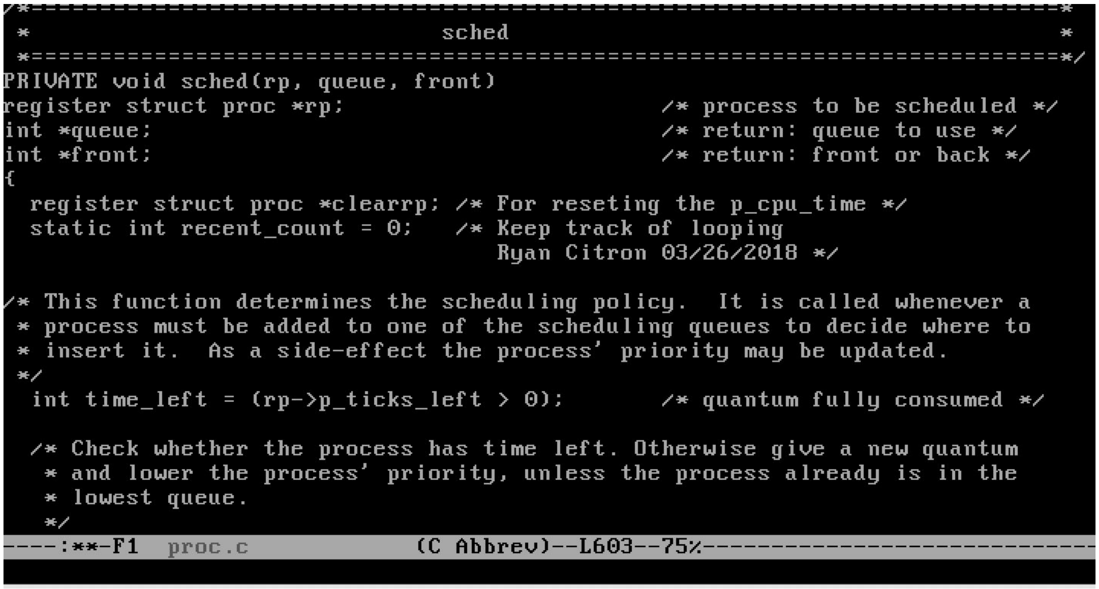
```
  struct proc *p_nextready;      /* pointer to next ready process */
  struct proc *p_caller_q;       /* head of list of procs wishing to send */
  struct proc *p_q_link;         /* link to next proc wishing to send */
  message *p_messbuf;            /* pointer to passed message buffer */
  int p_getfrom_e;               /* from whom does process want to receive? */
  int p_sendto_e;                /* to whom does process want to send? */

  sigset_t p_pending;            /* bit map for pending kernel signals */

  char p_name[P_NAME_LEN];       /* name of the process, including \0 */

  endpoint_t p_endpoint;         /* endpoint number, generation-aware */

  int p_cpu_time;                /* amount of time a process gets
                                    Ryan Citron 03/26/2018 */

#if DEBUG_SCHED_CHECK
  int p_ready, p_found;
#endif
};

/* Bits for the runtime flags. A process is runnable iff p_rts_flags == 0. */
----:---F1  proc.h           (C Abbrev)--L48--22%----------------------------
Wrote /usr/src/kernel/proc.h
```

Proc.c:

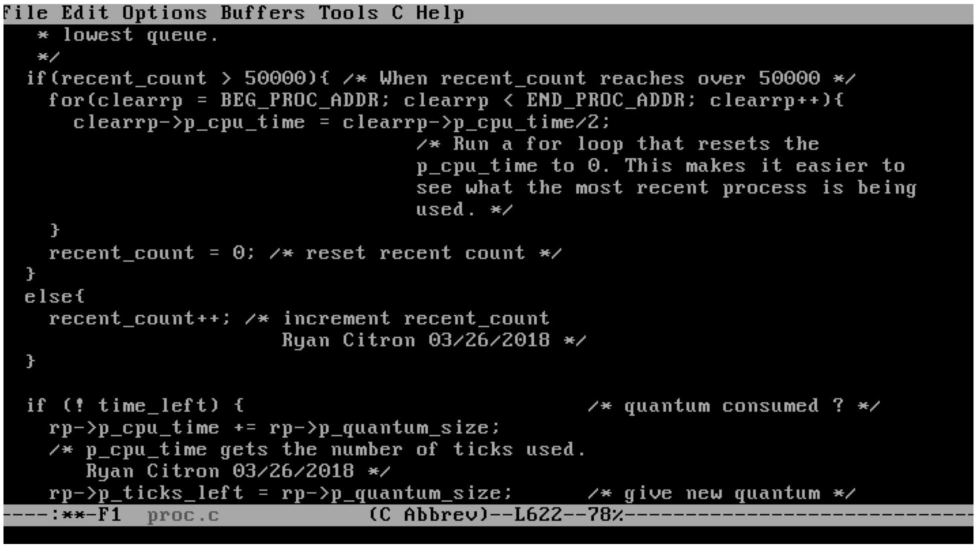
In the sched function, I added a lot of code. Near the beginning I added a proc *clearrp to reset the counters for the amount of time each process was getting. This way I am able to see what recently is. This is done by making a for loop and comparing a count to 50000. Once that count has been reached I set *clearrp to the beginning address and set the new p_cpu_time to the current time divided by 2. This allows me to be able to see what process was used recently without completely wiping out the number. The way I am incrementing the p_cpu_time is everytime a quantum is fully used, the p_cpu_time gets incremented by the size of that quantum.

In the pick_proc function, I added the way it schedules the lowest cpu time for a user process. I have a base number of 5000 which the way I keep track of numbers that is the largest the “smallest” amount of cpu time a process could ever reach. I check if the current process, rp, is a user process. If it is a user process I change the quan_min_time to that cpu time rather than the base of 5000. After that I set my next_ptr to be rp which is the user process. If it is not a user process it runs like normal.

In Sched:

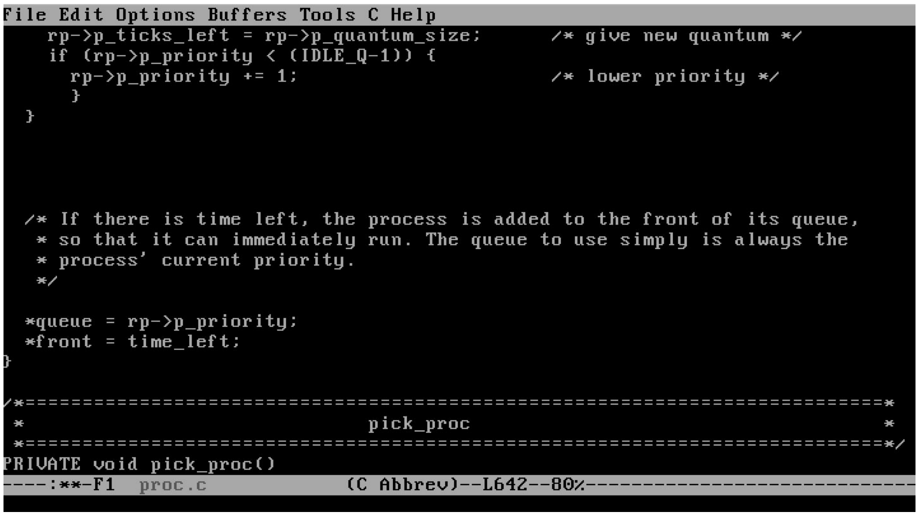


```
/*===========================================================================*
 *                              sched                                        *
 *===========================================================================*/
PRIVATE void sched(rp, queue, front)
register struct proc *rp;                       /* process to be scheduled */
int *queue;                                     /* return: queue to use */
int *front;                                     /* return: front or back */
{
  register struct proc *clearrp; /* For reseting the p_cpu_time */
  static int recent_count = 0;   /* Keep track of looping
                                    Ryan Citron 03/26/2018 */

/* This function determines the scheduling policy.  It is called whenever a
 * process must be added to one of the scheduling queues to decide where to
 * insert it.  As a side-effect the process' priority may be updated.
 */
  int time_left = (rp->p_ticks_left > 0);       /* quantum fully consumed */

  /* Check whether the process has time left. Otherwise give a new quantum
   * and lower the process' priority, unless the process already is in the
   * lowest queue.
   */
----:**-F1  proc.c          (C Abbrev)--L603--75%----------------------------
```



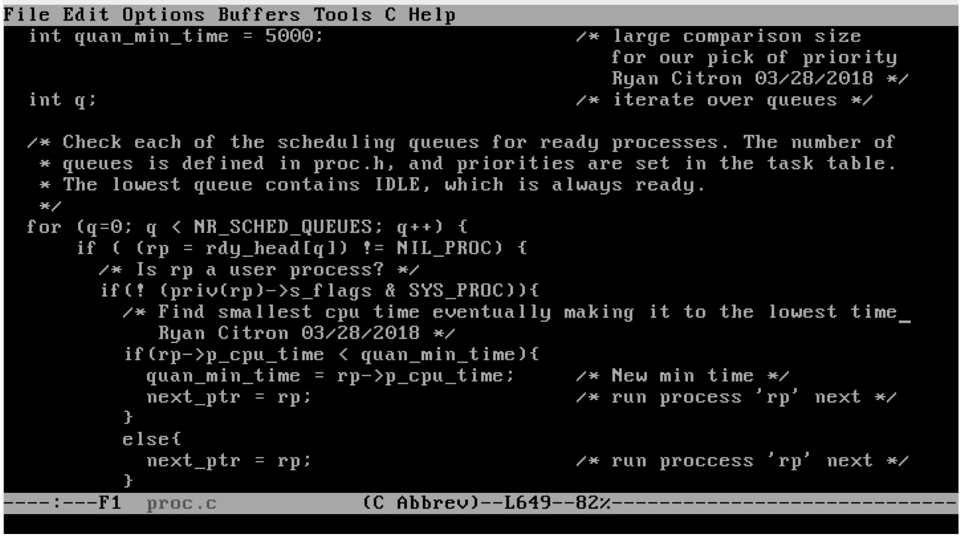
```
File Edit Options Buffers Tools C Help
   * lowest queue.
   */
  if(recent_count > 50000){ /* When recent_count reaches over 50000 */
    for(clearrp = BEG_PROC_ADDR; clearrp < END_PROC_ADDR; clearrp++){
      clearrp->p_cpu_time = clearrp->p_cpu_time/2;
                                /* Run a for loop that resets the
                                p_cpu_time to 0. This makes it easier to
                                see what the most recent process is being
                                used. */
    }
    recent_count = 0; /* reset recent count */
  }
  else{
    recent_count++; /* increment recent_count
                       Ryan Citron 03/26/2018 */
  }

  if (! time_left) {                            /* quantum consumed ? */
    rp->p_cpu_time += rp->p_quantum_size;
    /* p_cpu_time gets the number of ticks used.
       Ryan Citron 03/26/2018 */
    rp->p_ticks_left = rp->p_quantum_size;      /* give new quantum */
----:**-F1  proc.c          (C Abbrev)--L622--78%--------------------------
```

Ryan Citron



```
    rp->p_ticks_left = rp->p_quantum_size;      /* give new quantum */
    if (rp->p_priority < (IDLE_Q-1)) {
      rp->p_priority += 1;                      /* lower priority */
      }
  }




  /* If there is time left, the process is added to the front of its queue,
   * so that it can immediately run. The queue to use simply is always the
   * process' current priority.
   */

  *queue = rp->p_priority;
  *front = time_left;
}

/*===========================================================================*
 *                              pick_proc                                    *
 *===========================================================================*/
PRIVATE void pick_proc()
----:**-F1  proc.c            (C Abbrev)--L642--80%----------------------------
```

In Pick_Proc:

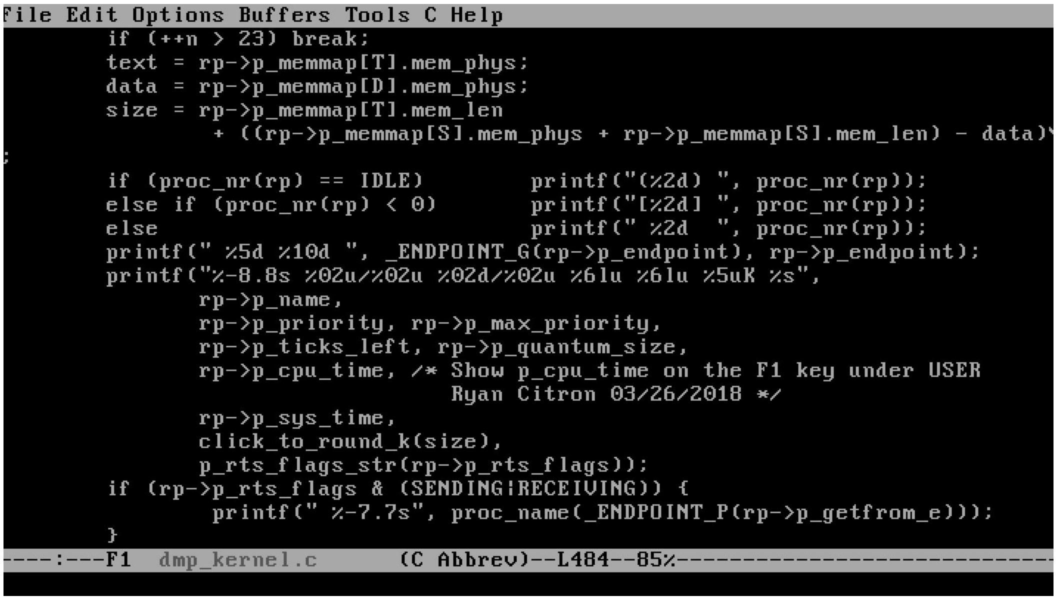


```
  int quan_min_time = 5000;                     /* large comparison size
                                                   for our pick of priority
                                                   Ryan Citron 03/28/2018 */
  int q;                                        /* iterate over queues */

  /* Check each of the scheduling queues for ready processes. The number of
   * queues is defined in proc.h, and priorities are set in the task table.
   * The lowest queue contains IDLE, which is always ready.
   */
  for (q=0; q < NR_SCHED_QUEUES; q++) {
      if ( (rp = rdy_head[q]) != NIL_PROC) {
        /* Is rp a user process? */
        if(! (priv(rp)->s_flags & SYS_PROC)){
          /* Find smallest cpu time eventually making it to the lowest time_
             Ryan Citron 03/28/2018 */
          if(rp->p_cpu_time < quan_min_time){
            quan_min_time = rp->p_cpu_time;     /* New min time */
            next_ptr = rp;                      /* run process 'rp' next */
          }
          else{
            next_ptr = rp;                      /* run proccess 'rp' next */
          }
----:---F1  proc.c            (C Abbrev)--L649--82%-----------------------------
```

In Servers/is:

dmp_kernel.c:

On line 484, under proctab_dmp I output my p_cpu_time under the USER section of the F1 key. That way everything is already structured for me.



```
        if (++n > 23) break;
        text = rp->p_memmap[T].mem_phys;
        data = rp->p_memmap[D].mem_phys;
        size = rp->p_memmap[T].mem_len
                + ((rp->p_memmap[S].mem_phys + rp->p_memmap[S].mem_len) - data)\
;
        if (proc_nr(rp) == IDLE)        printf("(%2d) ", proc_nr(rp));
        else if (proc_nr(rp) < 0)       printf("[%2d] ", proc_nr(rp));
        else                            printf(" %2d  ", proc_nr(rp));
        printf(" %5d %10d ", _ENDPOINT_G(rp->p_endpoint), rp->p_endpoint);
        printf("%-8.8s %02u/%02u %02d/%02u %6lu %6lu %5uK %s",
               rp->p_name,
               rp->p_priority, rp->p_max_priority,
               rp->p_ticks_left, rp->p_quantum_size,
               rp->p_cpu_time, /* Show p_cpu_time on the F1 key under USER
                                  Ryan Citron 03/26/2018 */
               rp->p_sys_time,
               click_to_round_k(size),
               p_rts_flags_str(rp->p_rts_flags));
        if (rp->p_rts_flags & (SENDING|RECEIVING)) {
                printf(" %-7.7s", proc_name(_ENDPOINT_P(rp->p_getfrom_e)));
        }
----:---F1  dmp_kernel.c      (C Abbrev)--L484--85%----------------------------
```