Scott Wickline

C435 – Project 2 Report

3/29/2021

Approach:

This project took much more time than the previous one. I broke the OS several times and started completely fresh 3 times. The third time is final solution show in this report. I found the most difficult part was reading through the code and understanding how it was being utilized. I spent several hours tracing though figuring out how and when things were called and in what order. My first attempt at modifying the code I tried this.

This was added to proc.h

clock_t recent_time[NR_TASKS + NR_PROCS], stopwatch;

Then in proc.c at the start of pic_proc I added;

realtime = get_uptime();
recent_time[proc_ptr->p_nr + NR_TASKS] = realtime - stopwatch;
stopwatch = realtime;

This made the OS crash and not boot. I was attempting to make the processes be picked round robin wise, but this did not work.

My second attempt I tried adding helper function to pic_proc show below.

```
/*===========================================================================*
*                           pickLowest                                      *
*===========================================================================*/

/* This function iterates through current priority que and looks for
process with lowers cpu time and returns its address. */

PRIVATE void pickLowest(rp, q)
register struct proc *rp;
{
    char lowestTime = 1000;

    rp = rdy_head[q];
    rp->p_cpu_time = rp->p_quantum_size - rp->p_ticks_left;

    if(q==0)
    return;

    while(rp != NULL)
    {
    if(rp->p_cpu_time < lowestTime){
    lowestTime = rp->p_cpu_time;
    }
    else
:
```

```
  register struct proc *rp;                     /* process to run */
  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++) {

      pickLowest(rp, q);

      if ( (rp = rdy_head[q]) != NIL_PROC) {
          next_ptr = rp;                        /* run process 'rp' next */
          if (priv(rp)->s_flags & BILLABLE)
              bill_ptr = rp;                    /* bill for system time */


         if(q==0)
           return;

       }

  }
:_
```

This also crashed the OS. If q were == to zero, the function would just return as I did not want to alter que 0 or 15. Then it was supposed to iterate through the ques and store the process with lowers CPU time. I suspect the function was storing a nil value and just stopped the boot process.

Solution:

My third attempt was successful. I added a p_cpu_time to the proc.h to the process struct. Every time shed is called after a process has been picked this variable will increment CPU time used. By adding the tick left each time. Then a I added a bit of code to check if the pic proc selects the idle process. If so, that would indicate that no other process is wanting to run. So, if the p_name is idle the program executes and iterates through the ques. It finds the process with least amount of CPU time and gives it priority value of 1 so that way it is selected to run.

Testing:

I fist tested my code to see if it was being called by adding print function to the code. I added printf(called); and right after booting my screen filled up with called. I also checked the process table and dump and process que functions. The user time for idle seemed to go up less than before with additional code and the random process 49 seemed to be in the process ques more frequently. From my testing I discovered that a process is always wanting run. So that is why I checked for IDLE process being selected. When its selected

Proc.h

```
 char p_priority;                /* current scheduling priority */
 char p_max_priority;            /* maximum scheduling priority */
 char p_ticks_left;              /* number of scheduling ticks left */
 char p_quantum_size;            /* quantum size in ticks */
 char p_cpu_time;          /* cpu  time used */
```

proc.c – shed

```
/*===========================================================================*
 *                                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 */
{
/* 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 i;
 int time_left = (rp->p_ticks_left > 0);     /* quantum fully consumed */
 rp->p_cpu_time = rp->p_cpu_time + rp->p_ticks_left;


 /* 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.
  */
```

```
/* If IDLE prcocess is selected to run. The function will search the ques.
* and find the process that has had the lowest CPU time and give it a priority.
* of 1 and insert into that que and give it a new quantum */

 if(rp->p_name == "IDLE" ){

        for(i=0; i< NR_SCHED_QUEUES; i++){
        if((rp = rdy_head[i]) != NIL_PROC)
                next_ptr = rp;
        if(rdy_head[i]->p_cpu_time > next_ptr->p_cpu_time)
                next_ptr = rdy_head[i];
                }
                rp->p_priority = 1;
                rp->p_ticks_left = rp->p_quantum_size;
                }

 if (! time_left) {                                /* quantum consumed ? */

    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;
}
```