Ittai Shay, 998099673
Abhishek Zaveri, 998855710

Programming Assignment 2

Changes to MINIX Source Code (additions bolded):

Note: We could not figure out how to pull the revised MINIX image or the modified source files. Copies of the test programs are included, and the MINIX changes are as follows:

```
~~~~~~~~~~ kernel/main.c ~~~~~~~~~~
  strncpy(rp->p_name, ip->proc_name, P_NAME_LErooN); /* set process name */

  /* initialize new proc arrays */
  for (j = 0; j < 91; j++) {
      rp->system_calls[j] = 0;
  }
  for (j = 0; j < 255; j++) {
      rp->num_messages[j] = 0;
  }
  rp->creation_time = get_uptime(); /* initialize creation_time */

  (void) get_priv(rp, (ip->flags & SYS_PROC));    /* assign structure */
```

```
~~~~~~~~~~ kernel/system/do_fork.c ~~~~~~~~~~
  rpc->p_ticks_left = (rpc->p_ticks_left + 1) / 2;
  rpp->p_ticks_left =  rpp->p_ticks_left / 2;

  /* initialize new proc arrays */
  for (i = 0; i < 91; i++) {
      rpc->system_calls[i] = 0;
  }
  for (i = 0; i < 255; i++) {
      rpc->num_messages[i] = 0;
  }
  rpc->creation_time = get_uptime(); /* initialize creation_time */
```

```
~~~~~~~~~~ kernel/proc.h ~~~~~~~~~~
   (added to the end of the proc struct)
  int system_calls[91];  /* keep track of number of calls made to each system call */
  int num_messages[255]; /* keep track of number of messages from proc i to proc j */
  clock_t creation_time; /* to help determine scheduling priority */
```

```
~~~~~~~~~~ kernel/proc.c ~~~~~~~~~~
sys_call function
  int result;                    /* the system call's result */
  vir_clicks vlo, vhi;        /* virtual clicks containing message to send */

  /* increment # of calls made to that system call */
  caller_ptr->system_calls[function]++;

mini_send function
    CopyMess(caller_ptr->p_nr, caller_ptr, m_ptr, dst_ptr,
      dst_ptr->p_messbuf);

    /* increment # msgs sent from caller to dst proc */
    caller_ptr->num_messages[proc_nr(dst_ptr)+NR_TASKS]++;

    if ((dst_ptr->p_rts_flags &= ~RECEIVING) == 0) enqueue(dst_ptr);
  } else if (

pick_proc function
  register struct proc *rp, *temprp;
  int q, t;                    /* 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 image 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) {
        if(q >= 7 && q <= 14) { /* start at the USER QUEUE, stop before IDLE */
          for(t = q; t <= 14; t++) {
            if( (temprp = rdy_tail[t]) != NIL_PROC) {
              if(rp->creation_time <= temprp->creation_time
                 && rp->p_user_time > temprp->p_user_time) {
                next_ptr = temprp;
                if(priv(temprp)->s_flags * BILLABLE)
                    bill_ptr = temprp;
                return;
              }
            }
          }
        }
        next_ptr = rp;            /* run process 'rp' next */
        if (priv(rp)->s_flags & BILLABLE)
            bill_ptr = rp;            /* bill for system time */
        return;
      }
  }
```

~~~~~~~~~~ servers/is/proto.h ~~~~~~~~~~

```
      _PROTOTYPE( void msg_matrix_dmp, (void));
      _PROTOTYPE( void sys_call_dmp, (void));
      _PROTOTYPE( void proc_queue_dmp, (void));
```

~~~~~~~~~~ servers/is/dmp.c ~~~~~~~~~~

```
      { F8,   msg_matrix_dmp, "Message matrix dump" },
      { F9,   sys_call_dmp, "System call dump" },
      { SF6,  proc_queue_dmp, "Process queue dump" },
```

~~~~~~~~~~ servers/is/dmp_kernel.c ~~~~~~~~~~

```
/*===========================================================================*
 *                              msg_matrix_dmp                               *
 *===========================================================================*/
PUBLIC void msg_matrix_dmp()
{
  register struct proc *rp;
  static struct proc *oldrp = BEG_PROC_ADDR;
  int i, j, r, n = 0;

  /* try to get a copy of the process table */
  if ((r = sys_getproctab(proc)) != OK) {
    report("IS", "warning: couldn't dump message matrix", r);
    return;
  }

  rp = BEG_PROC_ADDR;
  printf("\n");
  printf("-----Message Matrix Dump----");
  printf("\n");
  printf("  i->j|");
  for (i = 0; i < 10; i++) {
    printf("%6s|",rp->p_name);
    rp++;
  }
  printf("\n");
  for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
      if (isemptyp(rp)) continue;
      if (++n > 20) break;
      if (rp == NIL_PROC) continue;
    printf("%6s|",rp->p_name);
    for (j = 0; j < 10; j++) {
        printf("%6d|",rp->num_messages[j]);
    }
    printf("\n");
  }
  if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR;
  oldrp = rp;
}
```
~~~~~~~~~~

```
/*===========================================================================*
 *                             proc_queue_dmp                                *
 *===========================================================================*/
PUBLIC void proc_queue_dmp()
{
  register struct proc *rp;
  static struct proc *oldrp = BEG_PROC_ADDR;
  int i, j, r, n = 0;

  /* try to get a copy of the process table */
  if ((r = sys_getproctab(proc)) != OK) {
    report("IS","warning: couldn't getprocqueue dmp",r);
    return;
  }
  printf("\n");
  printf("-----Process Queue Dump-----");
  printf("\n");
  for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
    if (isemptyp(rp)) continue;
    if (++n > 20) break;
    if (rp != NIL_PROC) {
      printf("Name: %10s, Creation_Time: %10ld, CPU_Time: %10ld\n",
      rp->p_name, rp->creation_time, rp->p_user_time);
    }
  }
  if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR;
  oldrp = rp;
}

/*===========================================================================*
 *                               sys_call_dmp                                *
 *===========================================================================*/
PUBLIC void sys_call_dmp()
{
  register struct proc *rp;
  static struct proc *oldrp = BEG_PROC_ADDR;
  int i, j, r, n = 0;

  /* try to get a copy of the process table */
  if ((r = sys_getproctab(proc)) != OK) {
    report("IS","warning: couldn't dump system calls",r);
    return;
  }

  printf("\n");
  printf("-----System Call Dump-----");
  printf("\n");
  for (rp = oldrp; rp < END_PROC_ADDR; rp++) {
    if (isemptyp(rp)) continue;
    if (++n > 10) break;
    if (rp == NIL_PROC) continue;
    printf("%7s|",rp->p_name);
    for (j = 0; j < 91; j++) {
            if (rp->system_calls[j] == 0) continue;
        printf("%2d: %4d,", j, rp->system_calls[j]);
    }
    printf("\n");
  }
  if (rp == END_PROC_ADDR) rp = BEG_PROC_ADDR;
  oldrp = rp;
}
```