Dept. Math. & Comp. Sc.	**Midterm Operating Systems**
Vrije Universiteit	**31.03.2005**

1a An operating system can be seen as a virtual machine or as a resource manager. Explain the difference. *5pt*

As a virtual machine, an operating system provides an abstraction over the hardware by means of, for example, system calls. In this way, it provides a convenient way to program a machine without the need to know about hardware details. As a resource manager, it allows multiple processes (or users) to share the various resources such as CPU, storage, and network. Its role of manager consists of protecting those resources against simultaneous access, protecting processes against each other, supports fair sharing of those resources, and accounts processes for resource usage.

1b UNIX operating systems represent a hard disk by means of a so-called *block special file*. Explain what such a file is. *5pt*

A block special file for a hard disk is a representation of that disk by means of a file allowing a program to access the k-th block as simply the k-th block in that file. Read and write operations are thus carried out immediately on the raw device, but this fact is hidden from the program using the special file.

1c There is no DELETE file system call in MINIX. How then is a file deleted? *5pt*

MINIX, as other Unices, keeps track of the number of links *to a file. The* UNLINK *system call removes a link to a file. When the last link is removed, the operating system deletes the file.*

2a What is meant by the context of a process? *5pt*

The context of a process consists of the values contained in various registers of, for example, the CPU and MMU that are absolutely necessary to restore in order to let a process continue exactly where it left off. Crucial registers include the program counter, segment registers, and stack pointer.

2b When a hardware interrupt occurs, there is a moment when the software takes control over from the hardware. Explain when. *5pt*

This question is best explained by describing how an interrupt is initially handled. When an interrupt occurs, the interrupt controller passes the interrupt identifier to the CPU. The value in the program counter in the meantime has been pushed onto the current stack. The interrupt identifier is an offset into the interrupt table, from where the hardware loads a start address of the associated interrupt handler. At that point, the software takes control over from the hardware.

2c Explain what the test-set-lock instruction (TSL) does, and show how it can be used to protect a critical region. *5pt*

The TSL instruction reads the value of a variable and immediately sets it to 1 in a single atomic action. The following code can be used to protect a critical region:

```
enter_region:
  tsl  register,lock     // copy lock to register and set lock to 1
  cmp  register,#0       // was lock zero?
  jne  enter_region      // if it was non zero, lock was set, so loop
  ret                    // return to caller; critical region entered

leave_region:
  move  lock,#0          //  store a 0 in lock
  ret                    // return to caller
```

2d In MINIX, what does the procedure MINI_SEND do (see code on other page)? *5pt*

MINI_SEND *is called, for example, as the result of a system call. The operating system copies a message on behalf of the sender to the address space of the receiver, provided the latter was blocked waiting for that message. If the receiver is not waiting, the sender will be queued for the receiver and marked as* BLOCKED *(so that the CPU will be deallocated from it). When the message has been copied to the receiver's address space, the latter is marked as* READY *so that the CPU can eventually be allocated to it.*

3a What are the necessary and sufficient conditions for a deadlock to take place? *5pt*

(1) A resource is assigned to at most one process at a time; (2) Processes can request a resource while holding another; (3) A resource cannot be taken away from a process; (4) Two or more processes must be waiting for release of a resource held by another.

3b What is the fundamental difference between I/O tasks and processes in MINIX? *5pt*

Each process runs in its own address space with its own stack, whereas an I/O task is part of the kernel and thus runs in the context of the kernel. In essence, a task can be compared to a thread: it shares the address space with other tasks running in kernel mode, but still has its own stack and context.

THIS EXAM CONSISTS OF TWO PAGES

```
0001  PRIVATE  int  mini_send(caller_ptr,      dest,  m_ptr)
0002  register  struct  proc  *caller_ptr;
0003  int  dest;
0004  message  *m_ptr;
0005  {
0006    register  struct  proc  *dest_ptr,    *next_ptr;
0007    vir_bytes   vb;
0008    vir_clicks   vlo,  vhi;
0009
0010    if  (isuserp(caller_ptr)      &&  !issysentn(dest))      return(E_BAD_DEST);
0011    dest_ptr   = proc_addr(dest);
0012    if  (dest_ptr->p_flags      &  P_SLOT_FREE)     return(E_BAD_DEST);
0013
0014    vb  =  (vir_bytes)    m_ptr;
0015    vlo  =  vb  >>  CLICK_SHIFT;
0016    vhi  =  (vb  +  MESS_SIZE   -  1)  >>  CLICK_SHIFT;
0017    if  (vhi  <  vlo  ||
0018         vhi  -  caller_ptr->p_map[D].mem_vir         >=  caller_ptr->p_map[D].mem_len)
0019      return(EFAULT);
0020
0021    if  (dest_ptr->p_flags      &  SENDING)   {
0022      next_ptr   = proc_addr(dest_ptr->p_sendto);
0023      while  (TRUE)  {
0024           if  (next_ptr   ==  caller_ptr)    return(ELOCKED);
0025           if  (next_ptr->p_flags      &  SENDING)
0026                next_ptr   = proc_addr(next_ptr->p_sendto);
0027           else
0028                break;
0029      }
0030    }
0031
0032    if  (  (dest_ptr->p_flags      &  (RECEIVING    |  SENDING))    ==  RECEIVING    &&
0033          (dest_ptr->p_getfrom      ==  ANY  ||
0034           dest_ptr->p_getfrom      ==  proc_number(caller_ptr)))       {
0035      CopyMess(proc_number(caller_ptr),          caller_ptr,    m_ptr,   dest_ptr,
0036            dest_ptr->p_messbuf);
0037      dest_ptr->p_flags      &=  ~RECEIVING;
0038      if  (dest_ptr->p_flags      ==  0)  ready(dest_ptr);
0039    }  else  {
0040      caller_ptr->p_messbuf       =  m_ptr;
0041      if  (caller_ptr->p_flags      ==  0)  unready(caller_ptr);
0042      caller_ptr->p_flags      |=  SENDING;
0043      caller_ptr->p_sendto=       dest;
0044
0045      if  (  (next_ptr   =  dest_ptr->p_callerq)       ==  NIL_PROC)
0046           dest_ptr->p_callerq      =  caller_ptr;
0047      else  {
0048           while   (next_ptr->p_sendlink       !=  NIL_PROC)
0049                next_ptr   =  next_ptr->p_sendlink;
0050           next_ptr->p_sendlink       =  caller_ptr;
0051      }
0052      caller_ptr->p_sendlink       =  NIL_PROC;
0053    }
0054    return(OK);
0055  }
```

Grading: *The final grade is calculated by accumulating the scores per question (maximum: 45 points), and adding 5 bonus points. The maximum total MT is therefore 50 points. The final exam consists of two parts. Part 1 covers the same material as the midterm. Let $P1$ be the number of points for part 1, and $P2$ the number of points for part 2 (each being at most 50 points). The final grade E is computed as $E = \max\{MT, P1\} + P2$. The midterm exam counts only for first full exam.*