Dept. Computer Science Vrije Universiteit	Operating Systems 28.03.2007

This exam consists of two pages

1a MINIX3 has adopted the **client-server model** to structure itself. Explain what this model entails, and notably what the role of the kernel is. *5pt*

1b Drivers in MINIX3 run as ordinary user-space processes. In what sense does this impose restrictions for communicating with hardware controllers, and how is that solved? *5pt*

1c Sketch the flow of control when a user-space process calls the library routine read(fd,buffer,bytes) in a monolithic operating system. Hint: use a diagram. *5pt*

1d In many cases, the hardware offers support for multiple rings of protection for programs. How can we make use of this support when organizing an operating system? *5pt*

2a The semantics of the *atomic* **swap** machine instruction is defined as follows. Show how this instruction can be used to protect a critical section. *5pt*

```
swap(inout boolean a, inout boolean b){ temp = a; a = b; b = temp;}
```

2b Consider the program on page 2, which is to be executed as a separate MINIX3 user-space **lock manager** process. The core of the program is formed by the functions do_down() and do_up() which are standard operations on counting semaphores. Given lock_manager(), give a pseudo-code implementation of the function do_down(sema). *5pt*

2c Also give a sketch of the implementation of do_up(). *5pt*

2d Returning SUSPEND by do_down() has the result of suspending a process. Explain which process that is, and how this blocking is actually effectuated. *5pt*

3a Explain how MINIX3 (and many other operating systems) simulate multiple timers using a single clock. Draw a figure to explain your answer. *5pt*

3b Explain the difference between **character devices** and **block devices**, and why making this distinction can be helpful for improving I/O. *Hint: think of writing a stream of bytes to disk.* *5pt*

4a Explain the principle working of the fork() system call. *5pt*

4b **Copy-on-write** is a technique by which a block of memory is filled with data from a specific source only when first written to. How can this technique help in optimizing the implementation of fork()? *Be precise!* *5pt*

5a Explain what the mount() system call does by means of an example. *Explain your example!* *5pt*

5b Mount() changes fields in inodes and in-memory copies of superblocks. Explain these changes. *5pt*

5c Consider the following operations that are carried out on a formatted, but otherwise empty USB stick. Explain what the result will be when listing the directory contents (by means of the last operation ls). *5pt*

```
mount /dev/sdb1 /usbstick    Mount the USB stick
cd /usbstick                 Enter the directory
mkdir test                   Create a subdirectory ''test.''
touch test/x                 Create a file ''x'' in ''test.''
mount /dev/sdb1 test         Mount the USB stick again
ls test                      List the directory contents
```

5d Explain precisely what happened with the superblock table and inode table after the two mount operations from the previous example have been carried out. *5pt*

6a What is a protection domain? *5pt*

6b Give a practical example of how to switch from one protection domain to another, and explain how such a switch could be implemented by an operating system. *5pt*

```
PUBLIC int lock_manager(){
  int result, s, proc_nr;
  struct mproc *rmp;
  while (TRUE) {
    receive(ANY, &msg_in);
    who = msg_in.m_source;          /* who sent the message          */
    sema = msg_in.m5_l1;            /* which semaphore is this?      */
    call_request = msg_in.m5_i1;    /* which operation is requested? */
    mp = &mproc[who];
    switch(call_request){
      DOWN: result = do_down(sema); break;
      UP:   result = do_up(sema); break;
    }

    /* Send the results back to the user to indicate completion. */
    if (result != SUSPEND) setreply(who, result); /* Prepare  reply message */
    /* Send out all pending reply messages, including the answer to
     * the call just made above.
     */
    for (proc_nr = 0, rmp = mproc; proc_nr < NR_PROCS; proc_nr++, rmp++) {
      if ((rmp->mp_flags & REPLY) == REPLY ){
        send(proc_nr, &rmp->mp_reply);
        rmp->mp_flags &= ~REPLY;
      }
    }
  }
  return(OK);
}
```

***Grading:** The final grade is calculated by adding the scores per question (maximum: 90 points), and adding 10 bonus points. The maximum total is therefore 100 points.*