Department of Computer Science
University College London

1B11 Operating Systems Coursework

December 2002

Answer all parts of this question.

Consider a simple machine which does not have memory management. It has 8 general purpose registers, a stack pointer register and a program counter register. Interrupts arise when DMA transfers complete and when the clock ticks (every 1/50th second). In-memory processes are allowed to run for up to 80 milliseconds (ms) before being pre-empted if there is another process waiting to execute. When a synchronous I/O operation to disk starts the process is blocked until it completes.

The following pseudo-code shows the outline of some aspects of the operating system to manage the in-memory processes and the time-of-day clock. The purpose of this exercise is for you to understand and explain how the dispatching of processes and input-output interact and how multiprocessing is achieved. Your pseudo-code and comments should show how kernel and user mode state information (registers and stacks) is managed (eg. stack pointers swapped) .

a) Extend the pseudo-code to dispatch CPU-bound processes (eg. A, B and C), including the null process
b) Further extend it to deal with disk transfers. For the sake of simplicity assume that after a time the processes each do 10 msec of processing and then read or write a random block on the disk.
c) Suggest policies that the disk driver might be programmed to use to order or prioritise the access to disk blocks and indicate which you feel is the best and why.
d) Write pseudo code for the system call to create a new process which is an exact copy of the current process. On return from the system call a return value of 0 indicates to the programmer that it is the original process and –1 indicates it is the new process.

In each case discuss how you expect the execution of the various processes to proceed, showing when I/O occurs and completes.

Hints: Do NOT write this in assembly language nor in your favourite programming language. The pseudo-code should indicate important steps, not the full detail. You will need to assume that functions to add items to lists (queues) and to remove the first element from such lists are available.

All of the following code is executed in kernel mode. Comments are introduced by #

```
SYSCALL:    check argument to ensure valid syscall
            # calling process's PC and processor status will have been
            # saved on (user) stack and PC reset to address of this code

            switch to code to deal with syscall

            # invalid syscall; R0 holds return value from syscall
            Set R0 to error code
            return to calling process (mode changed to user)

#System Calls
GetTimeOfDay: set R0 to ToD
            return to calling process (mode changed to user)

ReadDisk:   Get parameters from stack (disk block etc)
            QueueDiskTransfer()         # Write this
            SuspendCurrentProcess()     # Write this
            ReturnToCurrentProcess()    # Write this
```

```
NewProcess: # Make a copy of the current process's memory etc (assume
            # enough memory available for this and no fragmentation
            # issues)
            # Set original process's return value to 0 and make it ready
            # to run
            # Set new process's return value to -1 and make it current
            # process and resume

Others:     # lots more – add any you feel necessary for these exercises

#------------------------------

# INTERRUPT HANDLER

ToD:        data   # datum holding the current date and time in ticks
TickCount:  data   # datum to count 4 ticks (200ms) assume zero intially

Interrupt:  check type and go to relevant ISR

ISRClock:   Save registers that will be used in ISR
            Increment ToD
            Increment TickCount
            If TickCount >= 4 then
               Reset TickCount to 0 and DispatchNextProcess()
            Restore saved registers
            Return from ISR

#--------------------------------

#DISPATCHER

DispatchNextProcess:
            # Running process is suspended unless only null process
            # is ready to run
            # you need to complete this


SuspendCurrentProcess:
            # Running process is added to blocked list
            # new process (or null P) is scheduled
            # you need to complete this

ReturnToCurrentProcess:
            # resume the (possibly new) current process.
            # you need to write this

#----------------------------------

#DISK DRIVER

QueueDiskTransfer:
            # Disk I/O from several processes may be requested at
            # same time so need to have a queue to manage them
            # This function adds requests to Q and orders it
            # you need to complete this
```

```
ISRDisk:    # ISR needs to flag the P that initiated this transfer as
            # ready to run
            # and check for and start another disk I/O if one exists
            # you need to write this
```

PLEASE NOTE THAT THIS IS GRADED COURSEWORK. A STUDENT WHO FAILS TO COMPLETE IT WILL BE AWARDED ZERO MARKS FOR IT.

THIS COURSEWORK SHOULD BE HANDED IN TO THE DEPARTMENTAL OFFICE NO LATER THAN:

12 noon on Friday 17th January 2003.

THERE ARE FIXED PENALTIES FOR HANDING YOUR WORK IN LATE -- THEY ARE:

<= 2 WORKING DAYS	10% OF MARK OBTAINED
> 2 WORKING DAYS	100% OF MARK OBTAINED