Course Operating Systems

Final Examination – Fall 2005

February, 2006

Duration: 2h

ludovic.apvrille@enst.fr

Authorized documents are limited to course's slides and to the code you produced during lab sessions. Basic calculators are authorized but quite useless. Answers should be as concise as possible. Also, you are free to answer either in **English** or in **French,** but please do not mix them!

I. Understanding of the course (7 points)

(a) What are the differences between the polling technique and the interrupt technique, when communicating with devices? For which kind of transfers / devices is the polling technique more efficient? Less efficient? (2 points)

(b) Explain what happened in the computer system when you get a "segmentation fault – core dumped" when executing an application. (2 points)

(c) The course has illustrated the use of condition variables over a producer / consumer example. First, explain what are "mutex" objects, and condition variables. Then, imagine a quite different programming example in which condition variables may be useful. Explain this example, and provide only most relevant parts of its code. (3 points)

II. Memory management in Linux Kernel (6 points)

In annex is provided a *HowTo* Linux document. Most customizations of Linux kernels are provided in that kind of documents. The provided document addresses the problem of transferring data from and to memory. Read it carefully before answering to next questions.

(a) Provide a short summary of this *HowTo* (a few lines).

(b) Explain, in general, what is a DMA transfer.

(c) Explain what is the main issue stated in this *HowTo*? Using figures might help you to explain it.

(d) Suppose you have a PC on which you want to solve these issues. What are all different steps you have to take on your system? Also, try to think about other steps not provided in this document.

III. Tasks sharing several resources (6 points)

Consider the following 5 tasks: P1, P2, P3, P4. Assume that Prio(P1)>Prio(P2)>Prio(P3)>Prio(P4)>Prio(P5) where Prio(P) is the priority of task P.

Task	Priority	Execution Sequence	Release time
P1	**High: 7**	**EE+S-E**	**7**
P2	**5**	**E+QQ-E**	**5**
P3	**3**	**E**	**4**
P4	2	**E+S+Q-S-E**	**2**
P5	**Low: 1**	**E+QQQQ-E**	**0**

These tasks share two resources S and Q as indicated in the table above.

We consider the following algorithm (named "stack based protocol"):

- The system is maintaining a variable V = current ceiling (as defined in the Priority Ceiling Protocol)
- When a task P is released, it is blocked from starting execution if Prio(P) is less or equal to V.
- When a task P requests a resource S, it is allocated.

(a) Explain why a deadlock cannot occur

(b) What are the values of the Priority Ceiling of Q and S?

(c) Draw the evolution of the value of V over the time interval [0, 21]

(d) Draw the execution of the above tasks over the time interval [0, 21]

(e) What would be the execution if we had used Highest Lock instead of the stack based protocol?

(f) For each task give the blocking factor under Highest Lock assumptions.

(g) How would you compute the blocking factors if we use the stack based protocol?

ANNEX: Excerpt of the *HowTo Linux* on "Patch to avoid bounce buffers"

Avoiding Bounce Buffers

This section provides information on applying and using the bounce buffer patch on the Linux 2.4 kernel. The bounce buffer patch, written by Jens Axboe, enables device drivers that support direct memory access (DMA) I/O to high-address physical memory to avoid bounce buffers.

This document provides a brief overview on memory and addressing in the Linux kernel, followed by information on why and how to make use of the bounce buffer patch.

Memory and Addressing in the Linux 2.4 Kernel

The Linux 2.4 kernel includes configuration options for specifying the amount of physical memory in the target computer. By default, the configuration is limited to the amount of memory that can be directly mapped into the kernel's virtual address space starting at PAGE_OFFSET. On i386 systems the default mapping scheme limits kernel-mode addressability to the first gigabyte (GB) of physical memory, also known as low memory. Conversely, high memory is normally the memory above 1 GB. High memory is not directly accessible or permanently mapped by the kernel. Support for high memory is an option that is enabled during configuration of the Linux kernel.

The Problem with Bounce Buffers

When DMA I/O is performed to or from high memory, an area is allocated in low memory known as a bounce buffer. When data travels between a device and high memory, it is first copied through the bounce buffer.

Systems with a large amount of high memory and intense I/O activity can create a large number of bounce buffers that can cause memory shortage problems. In addition, the excessive number of bounce buffer data copies can lead to performance degradation.

Peripheral component interface (PCI) devices normally address up to 4 GB of physical memory. When a bounce buffer is used for high memory that is below 4 GB, time and memory are wasted because the peripheral has the ability to address that memory

directly. Using the bounce buffer patch can decrease, and possibly eliminate, the use of bounce buffers.

Locating the Patch

The latest version of the bounce buffer patch is *block-highmem-all-18b.bz2*, and it is available from Andrea Arcangeli's -aa series kernels at http://kernel.org/pub/linux/kernel/people/andrea/kernels/v2.4/.

Configuring the Linux Kernel to Avoid Bounce Buffers

This section includes information on configuring the Linux kernel to avoid bounce buffers. The Linux Kernel-HOWTO at http://www.linuxdoc.org/HOWTO/Kernel-HOWTO.html explains the process of re-compiling the Linux kernel.

The following kernel configuration options are required to enable the bounce buffer patch:

Development Code - To enable the configurator to display the `High I/O Support` option, select the `Code maturity level options` category and specify "y" to `Prompt for development and/or incomplete code/drivers`.

High Memory Support - To enable support for physical memory that is greater than 1 GB, select the `Processor type and features` category, and select a value from the `High Memory Support` option.

High Memory I/O Support - To enable DMA I/O to physical addresses greater than 1 GB, select the `Processor type and features` category, and enter "y" to the `HIGHMEM I/O support` option. This configuration option is a new option introduced by the bounce buffer patch.

Enabled Device Drivers

The bounce buffer patch provides the kernel infrastructure, as well as the SCSI and IDE mid-level driver modifications to support DMA I/O to high memory. Updates for several device drivers to make use of the added support are also included with the patch.

If the bounce buffer patch is applied and you configure the kernel to support high memory I/O, many IDE configurations and the device drivers listed below perform DMA I/O without the use of bounce buffers:

aic7xxx_drv.o, aic7xxx_old.o, cciss.o, cpqarray.o, megaraid.o, qlogicfc.o, sym53c8xx.o