Data Provided: None

The University Of Sheffield.

DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

Autumn Semester 2012-13 (2.0 hours)

EEE6032 Operating Systems 6

Answer **THREE** questions. **No marks will be awarded for solutions to a fourth question.** Solutions will be considered in the order that they are presented in the answer book. Trial answers will be ignored if they are clearly crossed out. **The numbers given after each section of a question indicate the relative weighting of that section.**

1. **a.** Give three examples of instructions that are 'privileged' in a multi-tasking operating system? What common property do all such instructions share? **(5)**

b. Explain the mechanism by which a user-mode process in a multi-tasking operating system can call a privileged instruction. Explain how the user process can pass parameters to the systems call. **(6)**

c. A process running under a multi-tasking operating system executes an input/output (I/O) operation. Explain how execution is passed to the operating system's scheduler. On being passed execution, what would the scheduler do? **(5)**

d. In a modern operating system – for example, one with a journalling file system - does every file I/O operation have to result in a context switch? Justify your answer. **(4)**

2. **a.** What is the difference between a *logical address* and a *physical address*? **(2)**

b. Explain the operation of single-level paged memory. **(8)**

In a single-level paged memory system, to reduce losses due to internal fragmentation, it is beneficial to use very small pages. What is the problem with using very small pages? Explain how multi-level page tables can reduce this problem. What drawback do multi-level page tables themselves create? Is this really a problem in practice? **(4)**

c. In the context of a Unix-style fork, what is a *copy-on-write* (COW) process? What is the advantage of such an arrangement? Explain what sort of exception would be used in a copy-on-write implementation? **(6)**

3. **a.** Suppose you are given the binary executable of a program written many years ago for a long-since obsolete mainframe computer. The source code was lost many years ago. Describe how it might be possible to run such a program. What steps would be necessary to allow this to happen? Comment on any possible runtime performance penalty with your approach. **(4)**

b. How is a child process created under Windows? **(4)**

c. Why are some kernel processes scheduled using first-come, first-served scheduling, a scheduling method that is often regarded as inefficient and outdated? **(4)**

d. Process A spawns a child process B. Process B spawns another child process C. Then process B terminates with an unrecoverable error. What happens next? **(4)**

e. What is the advantage of performing inter-process communication (IPC) using shared memory? What advantage does shared memory have, for example, over pipes? **(4)**

4. **a.** What is cross compilation? Give two examples of when cross compilation can be useful. **(3)**

In the context of cross compilation, what do you understand by the terms:

1. Build system
2. Host system
3. Target system **(3)**

b. What is the advantage of using shared object libraries? Is this advantage always realised in practice? What runtime overhead does the use of shared object libraries incur? **(4)**

Under UNIX, what is the difference between a shared object library and a dynamic link library? Give an example of where using a dynamic link library would be advantageous. **(4)**

c. What are the differences between real-time systems, embedded systems and safety-critical systems? **(2)**

In a real-time system, identify the two types of *latency* which may limit system performance. How can these latencies be minimised? **(4)**

PIR