**CSE 221 – Graduate Operating Systems**

**Homework 1**

Name: Andreas Prodromou

SID: A53049230

**Question 1:**

**Hydra:**

* Protection Domain: Local Name Space (LNS).
  + "An LNS represents the current set of objects and rights to which a process has access, and those objects and rights change when a process moves from one LNS to another."

* Mechanism for crossing protection domains: CALL and RETURN primitive functions.
  + "The essential function of the CALL mechanism is to instantiate a procedure: to create an LNS for its execution domain, and to transfer control to the code body."

* Rights were represented through capabilities
  + "This protection includes not only the traditional read, write, execute capabilities, but arbitrary protection conditions whose meaning is determined by higher-level software"

* When crossing domains, rights can be amplified if the callee has greater freedom.
  + "If so, the kernel creates a new LNS which defines the new environment; the caller's LNS is superceded by this new LNS for the duration of the called procedure's execution"

* The OS determines whether to allow a domain crossing by comparing the rights included in the procedure's LNS against the object's requirements (template) and allows domain crossing on a successful match.
  + "The kernel examines the actual parameter capabilities supplied by the caller and determines whether all protection requirements are met"

**Multics:**

* Protection Domain: Principal Identifiers
  + "Associated with each process is an unforgeable character string identifier, assigned to the process when it was created"

* Mechanism for crossing protection domains: Authentication of users. The user authenticates and the user name is associated with one principal identifier that will later allow the user to call those procedures that he is allowed to.
  + "The general strategy chosen by Multics is to maintain individual accountability on a personal basis"

* Rights were represented via Access Control Lists (ACLs)
  + "Associated with each segment is an access control list, an open-ended list of names of users who are permitted to reference the segment"

* Rights can be amplified through a proxy login scheme
  + "Proxy login scheme permits the friend to identify himself, under his own password, and then request that the job be run under the principal identifier of the original researcher"

* The OS compares the Principal Identifier object's ACL on every access to every object.
  + "Whenever the process attempts to access a segment or other object cataloged by the storage system, the principal identifier of the process is compared with those appearing in the access control list of the object; if no match is found access is not granted"

**Pilot:**

Pilot was designed as a single-user personal computer operating system. As such, not much attention was paid in the protection aspect, since the main concern was errors and not maliciousness. Pilot's protection mechanisms were defensive, relying mostly on the language's strict type checking.

* Protection Domain: Object's type
  + "File types are used to determine which files should be processed by which client-level scavengers"

* Mechanism for crossing protection domains: The notion of "crossing domains" did not exist in Pilot.
  + "All computations on the machine (including Pilot itself) run in the same address space, which is unadorned with any noteworthy features, save a set of three flags attached to each page: referenced, written, and write-protected"

* Rights were represented via capabilities (Designed to protect against errors - not to withstand attacks)
  + "The File.Create operation creates a new file and returns a capability for it. Pilot file capabilities are intended for defensive protection against errors"

* Right amplification: The manager/Kernel pattern
  + "Layers interposed between the kernel and the manager can make use of the kernel and can in turn be used by the manager"

* Domain crossing was not really part of the Pilot system. One exception is that write-protected files could not be overwritten and the process who attempted that misuses the virtual memory and has to be debugged.
  + "The decision to cast Address-Fault and WriteProtectFault (i.e., storing into a write-protected space) as fatal errors is based on the judgment that any program which has incurred such a fault is misusing the virtual memory facilities and should be debugged"

**Question 2:**

* Why traditional Operating Systems explicitly support process debugging:
  + In order to have a system that is implementable, maintainable and modifiable (Tenex).
  + It is used for debugging the scheduler, portions of the core manager, and other basic routines. (Tenex)
  + Debugging can prevent programs from performing erroneous use of resources that can lead to fatal events

* Pilot supports "world-swap" where the debugger takes over during a process's execution. Two operations that need to be performed for this and need support from the OS are:
  + Save the contents of memory and the total machine state
  + Start (essentially boot) CoPilot from a boot-file

* Debuggers can't use the support that operating systems already provide for process communication and coordination because the debugger needs to be able to continue its operation even after a process that is being debugged crashes.

* Java programs are being ran in a Virtual Machine and the debugger is part of that Virtual Machine. There is no need for support from the native operating system. In the case of Perl, since the programs are not compiled into an executable, rather they are handled by the Perl interpreter, I would assume that the debugger is part of the language/interpreter and not a separate process. So there is no need for OS support for these two debuggers.

* Debugging the Kernel of an OS is more challenging, because when an error occurs in the OS, the debugger still needs to be able to operate correctly. In the case of debugging a user-level process, the OS can still support the debugger. Pilot runs the debugger from its own separate logical volume, and through the world-swap mechanism it allows for OS debugging.

**Question 3:**

Specialized Hardware in Tenex:

The Tenex system required modifications in the processor hardware for its operation. Specifically, it required the BBN pager, trapping facilities to underlie the pager's operation, a new system call instruction (JSYS) and one bit to the state word of the processor to record the context of the "caller". They also added an execute instruction and an instruction group that moves data between general registers between contexts.

The BBN pager provides individual mapping (relocation) of each page (512 words) of both user and monitor address spaces using separate maps for each. One benefit offered easy page sharing to multiple processes without high memory requirements. The pager permitted individual pages to be shared for read as well as write references.

When a page is not in the core, is protected from the requested type of access, or is nonexistent the BBN pager would generate a page fault (trap) that required processor support to be handled (trapping facilities)

All of Tenex monitor calls were reached via the new JSYS instruction. JSYS was not conflicting with th "UUO" system calls used by DEC software. It accomplishes a transfer from the user program to the specified monitor routine in one instruction time with the use of the JSYS transfer vector which occupied one page in the monitor space.

Drawback from using specialized Hardware:

One obvious drawback of using specialized hardware is compatibility: A modification (or improvement) in the hardware requires re-designing the OS. This makes it difficult to upgrade a system.

Do we still use hardware of this form today?

Today, Operating Systems do not rely on special hardware. Modern operating systems are flexible and can operate on a very wide range of underlying hardware. For example the Android operating system runs on a wide variety of processor/memory/screen combinations. Operating systems like Windows, Linux and Mac OS can run on virtually any combination of hardware. However the underlying processor may dictate re-compiling the OS (and applications) for the supported ISA.

**Question 4:**

My opinion is that this statement holds for an average user. Memory density increases rapidly and the cost per byte reduces, allowing an average user with relatively low storage requirements to never worry about the available storage space.

However, this statement does not hold for major internet services like Google/Amazon/Facebook etc. The amount of digital information created each year increases exponentially, while the increase in memory density is roughly linear. Furthermore, companies like these offer reliability guarantees to their users, which requires data redundancy to ensure that nothing will ever be lost. And on top of everything else, cloud storage services like Google Drive/Dropbox/Skydrive etc, allow the user to "un-delete" data, meaning that even when a user decides that he/she no longer needs something, the company is forced to keep it in their servers for a while more. It is obvious that the growth of digital information along with the reliability guarantees are a major concern for companies that hold massive amounts of data. One could argue that these companies could purchase more (cheap) storage, but the side-effects of this could be devastating. Area/cooling/power consumption/availability guarantees can incur a significant cost to the company, much higher than the cost of purchasing storage.

Departments and enterprises might fall into any of the above categories, depending on what exactly their storage needs are.

**Question 5:**

I believe there are several reasons multi-user time sharing systems are still in use:

1. Modern hardware can support huge workloads. Removing multi-user timesharing capability could be considered a waste of resources because usually an average user cannot utilize a modern system efficiently.
2. Allowing multi-user timesharing -- even when it's not really needed -- allows Operating Systems to be designed more generically, for every PC usage, ranging from a single user at home writing a text document, all the way up to a research facility with thousands of researchers simultaneously executing their programs.
3. With multimedia and multimedia streaming being one of the main uses of modern computers, even a "home PC" can benefit from a multi-user timesharing OS. For example, several users can simultaneously write, read, or stream data over a network connection to several devices like smartphones/TVs.