System Practicum : CS-307

Assignment - 05

Group No. : 05
Ankush Jindal [B13109], Shubham Chandel [B13231], Tushar Jain [B13236]

October 17, 2015

Note:

1. This is a theoretical assignment required to be submitted in the specified template. Use only the Latex template provided..
2. Reference material is uploaded on moodle. (The William Stallings Operating System Book)
3. Per group only one submission but viva will be individual.
4. Maximum score is 80 points.
5. Solve all parts of a question together.

---

1. Differentiate between:
   (a) Multiprogramming and Multiprocessing using a suitable example.
   (b) Process and Threaded environment.
   (c) Main memory, Secondary memory, Virtual Memory. (5+5+5=15 points)

Solution - 01 :

(a)
Multiprogramming is the ability of an operating system to execute more than one program on a single processor machine. More than one task/program/job/process can reside into the main memory at one point of time. A computer running excel and firefox browser simultaneously is an example of multiprogramming.
Multiprocessing is the ability of an operating system to execute more than one process simultaneously on a multiprocessor machine. In this, a computer uses more than one CPU at a time. Ex. When create a file computer take Time and Dates default.

(b)
Both processes and threads are independent sequences of execution. The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.
A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads.
A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources.

(c)
Processor access the primary memory in a random fashion. Unlike primary memory, secondary memory is not directly accessed through CPU.
Primary memory is volatile in nature while secondary memory is more or less permanent. Data processing speed in primary is much faster than secondary memory. Primary memory is more costly than secondary memory
Virtual memory is a memory management technique that is implemented using both hardware and software. It maps

memory addresses used by a program, called virtual addresses, into physical addresses in computer memory. It helps the process by reducing the fragmentation and using the technique of paging and swap to make the best use of secondary memory. Through this process we can have more memory available in primary memory than it actually has.

2. What are the prime responsibilities of any operating system ? (5 points)

   Solution - 02 :

   An OS has three main responsibilities:-
   (a) Perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.
   (b) Ensure that different programs and users running at the same time do not interfere with each other.
   (c) Provide a software platform on top of which other programs (i.e., application software) can run.
   The first two responsibilities address the need for managing the computer hardware and the application programs that use the hardware. The third responsibility focuses on providing an interface between application software and hardware so that application software can be efficiently developed. Since the operating system is already responsible for managing the hardware, it should provide a programming interface for application developers.

3. Discuss the process state transition diagram for UNIX operating system as shown in Fig. Define each state and explain each transition's meaning and significance. (20 points)

   Solution - 03 :
   UNIX Process States are as follows:
   User Running–Executing in User mode
   Kernal Running–Executing in Kernel mode
   Ready to Run, in Memory–Ready to run as soon as the kernel schedules it
   Asleep in Memory–Unable to execute until an event occurs, process in Main memory
   Ready to Run, Swapped– Process is ready to run, but the swapper must swap the process into main memory before kernel can schedule it to execute
   Sleeping, Swapped–The process is awaiting an event and has been swapped to secondary storage
   Preempted–Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process
   Created–Process is newly created and not yet ready to run
   Zombie–Process no longer exists but it leaves a record for its process to collect.

   The running state is divided into 2 states, to distinguish when the process is running in the user mode and in the kernel mode. Note that when running in the kernel mode, the process is in fact running OS code. In class, the view was taken that when an interrupt occurs, the OS runs outside the processes. The UNIX state diagram reflects the view that the OS runs in the context of processes. The advantage of this approach is to reduce process switching. When the kernel code has completed and no process switching is required (e.g. the system call was satisfied immediately), then control is simply returned to the user code. Thus the OS is seen as a collection of subprograms run in kernel mode when some interrupt or system trap occurs. The zombie state corresponds to the terminated state
   Asleep in memory corresponds to the waiting state. Note that when insufficient memory is available to run a process when it is created, it is placed in swap space (on the hard drive). Note that when dealing with hardware interrupts in kernel mode, the interrupt is serviced and control returned back to the kernel code running at the time of the interrupt
   The Preempted state and ready to run in memory state are essentially the same state (as indicated by the dotted line). Processes in either state are placed in the same scheduling queue (the ready queue). When the kernel is ready to return control to the user program, it may decide to preempt the current process in favor of another that is ready and has a higher priority. In this case, the current process is placed in the preempted state.

4. Compare the thread state diagram of Windows and Unix as shown in Fig and Fig respectively. (20 points)

   Solution - 04 :
   Windows Thread-Based States: Windows execution is an object-oriented, Multi threaded, six state process, compared to

the 8 or 9 associated with UNIX. Since the thread is the basic level of execution on Windows, processing states are also displayed as thread-based. There are two executable states; Runnable and Not Runnable.
A thread in Windows goes into a Ready state, moves to Standby when Picked to Run, then is Switched to Running. From the Running state, a thread can be placed back into the Ready state if it is Preempted by a higher-priority thread, or consumes its allocated time slice. Otherwise, it can become Blocked/Suspended and moves to Waiting for some condition, or it has finished and is Terminated.
A Waiting state is either Unblocked/Resumed when the resource becomes available and the state changes to Ready, or Unblocked to the Transition state if the thread is ready to run but still waiting on a resource.

Unix Thread-Based States: There are nine process states recognized by UNIX, are broken down into various states, from Created, Running, Ready to Run, Sleeping (Blocked), to Zombie (exit/termination).
When a new process in UNIX is Created, it becomes Ready to Run; either in memory, or swapped to disk. If it is initially swapped out (due to insufficient memory), it becomes swapped back into memory and Ready to Run, in Memory. When a process is in memory, it has multiple paths it can take. Towards a Running path, it is either accepted by the kernel for running in the kernel and/or user space, or it is preempted (queued in memory) by a higher-priority process before it is executed by the processor.
A user process runs programs and utilities in the user space, and in kernel mode in order to execute system-level code that uniquely belongs to the kernel (disk and file functions, hardware I/O functions, memory allocation, and process swapping).

UNIX Running states:
- User mode
- Kernel (system) mode: Memory allocation and process swapping.
There are two equivalent Ready states that form a single queue for the UNIX dispatcher:
- Ready to Run, in Memory: Process that has been loaded into memory and is awaiting execution by the processor.
- Preempted: Read to run and in memory, but subordinated by another process with higher priority that has become available.
Various system calls and interrupts may occur to processes between the user mode and kernel mode. A process running in the user mode may also be preempted by another process of higher priority, but cannot if it is running in kernel mode - only as a process moves from the kernel to the user mode.
Otherwise, a process transitions to Asleep, in Memory, where it can wake up to become Ready to Run, in Memory, or change to Sleep, Swapped to disk. If it has been put to sleep with memory swapped out to disk, it can wake up to a Ready to Run, Swapped state while it awaits to be swapped from disk back into memory. The final state is called, Zombie. This is the exit, or termination state of a process, which lingers around for a while to provide a debugging trail.

5. Study the comparison between Windows and Linux operating system as shown in Fig. Explain and write your comments for each and every point discussed. (20 points)

Solution - 05 :
1. In windows physical memory (RAM) dynamically mapped in to kernel address space as needed.But in Linux kernel splits RAM 3/1 (could also be 2/2, or 1/3) into user space (high memory) and kernel space (low memory) respectively. The user space range: 0x00000000 - 0xbfffffff The kernel space range: 0xc0000000 - 0xffffffff The kernel can directly access this 1 GB of addresses (well, not the full 1 GB, there are 128 MB reserved for high memory access). Every kernel process can also access the user space range if it wishes to. And to achieve this, the kernel maps an address from the user space (the high memory) to its kernel space (the low memory), the 128 MB mentioned above are especially reserved for this.
2. Page size versus TLB usage
Since every access to memory must be mapped from virtual to physical address, reading the page table every time can be quite costly. Therefore, a very fast kind of cache, the Translation Lookaside Buffer (TLB), is often used. The TLB is of limited size, and when it cannot satisfy a given request (a TLB miss) the page tables must be searched manually (either in hardware or software, depending on the architecture) for the correct mapping. Larger page sizes mean that a TLB cache of the same size can keep track of larger amounts of memory, which avoids the costly TLB misses.
3. In computer operating systems, paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory.In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages. The main advantage of paging overmemory segmentation is that it allows the physical address space of a process to be non-contiguous.
Paging is an important part of virtual memory implementation in most contemporary general-purpose operating systems,

allowing them to use secondary storage[a] for data that does not fit into physical random-access memory (RAM). Much of code and data for kernel and drivers is Pageable (means is capable of paging to disk). Initialization code is deleted after boot and page table (is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical memory) are fully Pageable. But for Linux Kernel and modules are non-pages but can be unloaded.
4. In windows User mode allocation of virtual address separated from mapping address as a view of a physical object while in Linux User mode addresses directly mapped to physical object.
5. AWE solves this problem by allowing applications to directly address huge amounts of memory while continuing to use 32-bit pointers. AWE allows applications to have data caches larger than 4GB (where sufficient physical memory is present). AWE uses physical non-paged memory and window views of various portions of this physical memory within a 32-bit virtual address space. Using Address Windowing Extension (AWE), physical memory can be allocated to large applications and directly man- aged by efficiently mapping/ unmapping into the address space.
6. In both Windows and Linux Copy-on-write (sometimes referred to as COW), is the name given to the policy that whenever a task attempts to make a change to the shared information, it should first create a separate (private) copy of that information to prevent its changes from becoming visible to all the other tasks. If this policy is enforced by the operating system kernel, then the fact of being given a reference to shared information rather than a private copy can be transparent to all tasks, whether they need to modify the information or not.
7. There are two different modes: user mode and kernel mode for CPU. Normal user/kernel split is 2GB/2GB; but in Windows it can be booted to give 3GB/1GB. While in Linux normal user/kernel split is 3GB/1GB; Linux can run kernel and user in separate address spaces, giving user up to 4GB.
8. In windows Cache manager manages memory mapping of files into kernel address space, using virtual memory manager to do actual paging and caching of pages in the standby and modified lists of pages while in Linux page cache implements caching of pages and used as lookaside cache for paging system.
9. In windows Threads can do direct I/O to bypass cache manager views while in Linus processes can do direct I/O to bypass page cache.
10. In Windows Page Frame Number (PFN) database is central data structure. Pages in PFN are either in a process page table or linked into one of several lists; standby, modified, free, bad. In Linux pages removed from process address spaces kept in page cache.
11. In windows Section Objects describe map-able memory objects like files, and include pageable, create-on-demand prototype page table which can be used to uniquely locate pages, including when faulted pages are already in transition. In Linux swap cache used to manage multiple instances of faulting the same page.
12. In a computer operating system that uses paging for virtual memory management, page replacement algorithms decide which memory pages to page out (swap out, write to disk) when a page of memory needs to be allocated. Paging happens when a page fault occurs and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold. When the page that was selected for replacement and paged out is referenced again it has to be paged in (read in from disk), and this involves waiting for I/O completion. This determines the quality of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm. A page replacement algorithm looks at the limited information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimize the total number of page misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself.
13. In windows page replacement is based on working sets, for both process and the kernel-mode(the system process) while in Linux page replacement uses a global clock algorithm.
14. In windows security features for encrypting page file and clearing pages when freed. There is security features for page file when freed.
15. In windows allocate space in paging file as needed, so writes can be localized for a group of freed pages; shared pages use indirection through prototype page tables associated with section object, so pagefile space can be freed immediately. In Linux allocate space in swap disk as needed, so writes can be localized for a group of freed pages; shared pages keep swap slot until all processes the slot have faulted the page back in