WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

1: Operating System - Process Scheduling

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table â^'.

Next Page Definition The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue. A new process is always put in this queue. The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute. Schedulers Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers. Medium Term Scheduler Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

2: What is a System Component?

A process is a program in execution. For example, when we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are both programs. When we actually run the binary code, it becomes a process. A process is an 'active' entity, as opposed to.

The need to have clearly defined roles, apportioned accountability and expectations for how the company is to be managed become increasingly vital. The further away that senior leadership separates from day-to-day operations the more the onus for communication of core objectives is heightened. This is where the Operational Management System OMS is supposed to come in to play, suturing and marrying together the collection of processes, expectations and requirements for the successful and holistic management of a business. Often standards become anything but standard, and the processes that are in place for how the business is managed deviate so much that sharing information, working together and coordination becomes difficult. A business is an evolving entity, constantly flexing between expansion and triage. As the body corporate develops, it is necessary that your OMS bends in unison to accommodate these changes. To make this as efficient a possible, it will demand thorough and periodic review. If an organisation has not implemented a management system before, then self-assessment will not be effective as there is no frame of reference for whether a project is going well or badly. Self-assessments will come back with positive feedback: There will definitely be a shake up of some kind, maybe heads will roll, and then the external auditors will be called in to compile a damage report and suggest a framework for repairs. What they thought before was plain sailing is actually a perfect storm. Years and millions wasted, and then the stress and strain of a companywide overhaul to boot. Not the best place to be inâ€¦. Fortunately, all of these attackers can be headed off at the pass with some easily-installable processes and behaviours. Seifert points to three easy fixes: Involve the fly-on-the-wall early: Instead of contracting in an external pair of eyes and ears as the weapon of last resort against a failing OMS, include this resource at the beginning of the process. This will act to mitigate those systemic problems that become noticed in year three. A management system should follow an in-built annual cycle with continuous inspection and evaluation of the gaps that need to be filled and shortfalls that need to be addressed. Those companies that are recognised as the standard-bearers of OMS systems in the oil and gas industry, the likes of ExxonMobil and Chevron, have decades of OMS development and implementation under their belts, but Seifert also contents that they stand out from the crowd in another all-important way:

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

3: Process management (computing) - Wikipedia

Process management is an integral part of any modern-day operating system (OS). The OS must allocate resources to processes, enable processes to share and exchange information, protect the resources of each process from other processes and enable synchronization among processes.

A process is a program object code stored on some media in execution. Processes are, however, more than just the executing program code often called the text section in Unix. They also include a set of resources such as open files and pending signals, internal kernel data, processor state, an address space, one or more threads of execution, and a data section containing global variables. Processes, in effect, are the living result of running program code. Threads of execution, often shortened to threads, are the objects of activity within the process. Each thread includes a unique program counter, process stack, and set of processor registers. The kernel schedules individual threads, not processes. In traditional Unix systems, each process consists of one thread. In modern systems, however, multithreaded programsâ€”those that consist of more than one threadâ€”are common. As you will see later, Linux has a unique implementation of threads: It does not differentiate between threads and processes. To Linux, a thread is just a special kind of process. On modern operating systems, processes provide two virtualizations: The virtual processor gives the process the illusion that it alone monopolizes the system, despite possibly sharing the processor among dozens of other processes. Chapter 4, "Process Scheduling," discusses this virtualization. Virtual memory lets the process allocate and manage memory as if it alone owned all the memory in the system. Virtual memory is covered in Chapter 11, "Memory Management. A program itself is not a process; a process is an active program and related resources. Indeed, two or more processes can exist that are executing the same program. In fact, two or more processes can exist that share various resources, such as open files or an address space. A process begins its life when, not surprisingly, it is created. In Linux, this occurs by means of the fork system call, which creates a new process by duplicating an existing one. The process that calls fork is the parent, whereas the new process is the child. The parent resumes execution and the child starts execution at the same place, where the call returns. The fork system call returns from the kernel twice: Often, immediately after a fork it is desirable to execute a new, different, program. In modern Linux kernels, fork is actually implemented via the clone system call, which is discussed in a following section. Finally, a program exits via the exit system call. This function terminates the process and frees all its resources. A parent process can inquire about the status of a terminated child via the wait4 2 system call, which enables a process to wait for the termination of a specific process. When a process exits, it is placed into a special zombie state that is used to represent terminated processes until the parent calls wait or waitpid. Another name for a process is a task. The Linux kernel internally refers to processes as tasks. Process Descriptor and the Task Structure The kernel stores the list of processes in a circular doubly linked list called the task list 3. The process descriptor contains all the information about a specific process. This size, however, is quite small considering that the structure contains all the information that the kernel has and needs about a process. Prior to the 2. This allowed architectures with few registers, such as x86, to calculate the location of the process descriptor via the stack pointer without using an extra register to store the location. The new structure also makes it rather easy to calculate offsets of its values for use in assembly code. Because of backward compatibility with earlier Unix and Linux versions, however, the default maximum value is only 32, that of a short int , although the value can optionally be increased to the full range afforded the type. The kernel stores this value as pid inside each process descriptor. This maximum value is important because it is essentially the maximum number of processes that may exist concurrently on the system. Although 32, might be sufficient for a desktop system, large servers may require many more processes. The lower the value, the sooner the values will wrap around, destroying the useful notion that higher values indicate later run processes than lower values. Consequently, it is very useful to be able to quickly look up the process descriptor of the currently executing task, which is done via the current macro. This macro must be separately implemented by each architecture. The assembly is shown here: When 4KB stacks are enabled, is used in lieu of Thus, current on PPC merely returns the value stored in the register r2.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

PPC can take this approach because, unlike x86, it has plenty of registers. Because accessing the process descriptor is a common and important job, the PPC kernel developers deem using a register worthy for the task. Process State The state field of the process descriptor describes the current condition of the process see Figure 3. Each process on the system is in exactly one of five different states. This value is represented by one of five flags: This is the only possible state for a process executing in user-space; it can also apply to a process in kernel-space that is actively running. The process also awakes prematurely and becomes runnable if it receives a signal. This is used in situations where the process must wait without interruption or when the event is expected to occur quite quickly. If the parent calls wait4 , the process descriptor is deallocated. If applicable, it also provides a memory barrier to force ordering on other processors this is only needed on SMP systems. Process Context One of the most important parts of a process is the executing program code. Normal program execution occurs in user-space. When a program executes a system call see Chapter 5, "System Calls" or triggers an exception, it enters kernel-space. At this point, the kernel is said to be "executing on behalf of the process" and is in process context. When in process context, the current macro is valid 7. Upon exiting the kernel, the process resumes execution in user-space, unless a higher-priority process has become runnable in the interim, in which case the scheduler is invoked to select the higher priority process. System calls and exception handlers are well-defined interfaces into the kernel. A process can begin executing in kernel-space only through one of these interfacesâ€”all access to the kernel is through these interfaces. All processes are descendents of the init process, whose PID is one. The kernel starts init in the last step of the boot process. The init process, in turn, reads the system initscripts and executes more programs, eventually completing the boot process. Every process on the system has exactly one parent. Likewise, every process has zero or more children. Processes that are all direct children of the same parent are called siblings. The relationship between processes is stored in the process descriptor. Consequently, given the current process, it is possible to obtain the process descriptor of its parent with the following code: A good example of the relationship between all processes is the fact that this code will always succeed: Oftentimes, however, it is desirable simply to iterate over all processes in the system. This is easy because the task list is a circular doubly linked list. To obtain the next task in the list, given any valid task, use: On each iteration, task points to the next task in the list: It can be expensive to iterate over every task in a system with many processes; code should have good reason and no alternative before doing so.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

4: Linux Kernel Process Management | Process Descriptor and the Task Structure | InformIT

Process scheduling is a major element in process management, since the efficiency with which processes are assigned to the processor will affect the overall performance of the system. It is essentially a matter of managing queues, with the aim of minimising delay while making the most effective use of the processor's time.

A computer program consists of a series of machine code instructions which the processor executes one at a time. This means that, even in a multi-tasking environment, a computer system can, at any given moment, only execute as many program instructions as there are processors. In a single-processor system, therefore, only one program can be running at any one time. The fact that a modern desktop computer can be downloading files from the Internet, playing music files, and running various applications all at apparently the same time, is due to the fact that the processor can execute many millions of program instructions per second, allowing the operating system to allocate some processor time to each program in a transparent manner. In recent years, the emphasis in processor manufacture has been on producing multi-core processors that enable the computer to execute multiple processes or process threads at the same time in order to increase speed and performance. Essentially, a process is what a program becomes when it is loaded into memory from a secondary storage medium like a hard disk drive or an optical drive. Each process has its own address space, which typically contains both program instructions and data. Despite the fact that an individual processor or processor core can only execute one program instruction at a time, a large number of processes can be executed over a relatively short period of time by briefly assigning each process to the processor in turn. While a process is executing it has complete control of the processor, but at some point the operating system needs to regain control, such as when it must assign the processor to the next process. It then creates a data structure in memory called a process control block PCB that will be used to hold information about the process, such as its current status and where in memory it is located. The operating system also maintains a separate process table in memory that lists all the user processes currently loaded. When a new process is created, it is given a unique process identification number PID and a new record is created for it in the process table which includes the address of the process control block in memory. Information about the resources allocated to a process is also held within the process control block. Process states The simple process state diagram below shows three possible states for a process. They are shown as ready the process is ready to execute when a processor becomes available , running the process is currently being executed by a processor and blocked the process is waiting for a specific event to occur before it can proceed. The lines connecting the states represent possible transitions from one state to another. At any instant, a process will exist in one of these three states. On a single-processor computer, only one process can be in the running state at any one time. The remaining processes will either be ready or blocked, and for each of these states there will be a queue of processes waiting for some event. A simple three-state process state diagram Note that certain rules apply here. Processes entering the system must initially go into the ready state. A process can only enter the running state from the ready state. A process can normally only leave the system from the running state, although a process in the ready or blocked state may be aborted by the system in the event of an error, for example , or by the user. Although the three-state model shown above is sufficient to describe the behaviour of processes generally, the model must be extended to allow for other possibilities, such as the suspension and resumption of a process. When a process is suspended, it essentially becomes dormant until resumed by the system or by a user. Because a process can be suspended while it is either ready or blocked, it may also exist in one of two further states - ready suspended and blocked suspended a running process may also be suspended, in which case it becomes ready suspended. A five-state process state diagram The queue of ready processes is maintained in priority order, so the next process to execute will be the one at the head of the ready queue. The queue of blocked process is typically unordered, since there is no sure way to tell which of these processes will become unblocked first although if several processes are blocked awaiting the same event, they may be prioritised within that context. To prevent one process from monopolising the processor, a system timer is started each time a new process starts executing. The process will be allowed to run for a set period of time,

after which the timer generates an interrupt that causes the operating system to regain control of the processor. The operating system sends the previously running process to the end of the ready queue, changing its status from running to ready, and assigns the first process in the ready queue to the processor, changing its status from ready to running. Process control blocks The process control block PCB maintains information that the operating system needs in order to manage a process. PCBs typically include information such as the process ID, the current state of the process e. The PCB also stores the contents of various processor registers the execution context , which are saved when a process leaves the running state and which are restored to the processor when the process returns to the running state. When a process makes the transition from one state to another, the operating system updates the information in its PCB. When the process is terminated, the operating system removes it from the process table and frees the memory and any other resources allocated to the process so that they become available to other processes. The diagram below illustrates the relationship between the process table and the various process control blocks. The process table and process control blocks The changeover from one process to the next is called a context switch. During a context switch, the processor obviously cannot perform any useful computation, and because of the frequency with which context switches occur, operating systems must minimise the context-switching time in order to reduce system overhead. Many processors contain a register that holds the address of the current PCB, and also provide special purpose instructions for saving the execution context to the PCB when the process leaves the running state, and loading it from the PCB into the processor registers when the process returns to the running state. Process scheduling Process scheduling is a major element in process management, since the efficiency with which processes are assigned to the processor will affect the overall performance of the system. The operating system carries out four types of process scheduling: Before accepting a new program, the long-term scheduler must first decide whether the processor is able to cope effectively with another process. The long-term scheduler may limit the total number of active processes on the system in order to ensure that each process receives adequate processor time. New processes may subsequently be created, as existing processes are terminated or suspended. Medium-term scheduling is part of the swapping function. The term "swapping" refers to transferring a process out of main memory and into virtual memory secondary storage or vice-versa. This may occur when the operating system needs to make space for a new process, or in order to restore a process to main memory that has previously been swapped out. Any process that is inactive or blocked may be swapped into virtual memory and placed in a suspend queue until it is needed again, or until space becomes available. The swapped-out process is replaced in memory either by a new process or by one of the previously suspended processes. The task of the short-term scheduler sometimes referred to as the dispatcher is to determine which process to execute next. This will occur each time the currently running process is halted. The objectives of short-term scheduling are to ensure efficient utilisation of the processor and to provide an acceptable response time to users. Note that these objectives are not always completely compatible with one another. On most systems, a good user response time is more important than efficient processor utilisation, and may necessitate switching between processes frequently, which will increase system overhead and reduce overall processor throughput. Queuing diagram for scheduling Threads A thread is a sub-process that executes independently of the parent process. A process may spawn several threads, which although they execute independently of each other, are managed by the parent process and share the same memory space. Most modern operating systems support threads, which if implemented become the basic unit for scheduling and execution. If the operating system does not support threads, they must be managed by the application itself. Threads will be discussed in more detail elsewhere.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

5: Process management of operating system - IncludeHelp

Operating system manages processes by performing tasks such as resource allocation and process scheduling. When a process runs on computer device memory and CPU of computer are utilized. The operating system also has to synchronize the different processes of computer system.

The basic thing to learn here is that the PCB has all the info needed so that if a process is stopped it can be restarted. This diagram is taken from the source material for this post, available here on slide The diagram is fairly simple. You can see how when the actual switching of processes is happening that neither process is executing, as the CPU cannot perform any processing when it is in the middle of switching. But why not have multiple threads within one process? You can have multiple processes sharing one computer, so why not have multiple threads sharing a process? This would be useful for a web browser for example, as you can have one thread receiving data from the network while another displays text. With a web browser, you could have one thread that is getting input from the user, while another thread is autosaving. Imagine if your word processor was single threaded: Threads You might hear something called a lightweight process. This is just another way of saying thread. So threads are multiple flows of control sharing one address space. Remember that all threads that are in the same process will share the same code section, global variables, network connections and open files. You may have heard of multithreading. This is when multiple threads share a process. Benefits of using threads Programs become more responsive, and resources are shared better. If part of a program is blocked it can keep running. As the IO and CPU use is shared between threads of the same process, performance is usually better too. There is a lot of cost associated with making a process. Memory and resources need to be allocated. Creating a thread can be up to x faster. There are some concerns though: I mentioned before that two threads share the same data. Therefore one thread might read from a location while another is writing to it, so you need to take care to stop problems like this. In the user space Implementing threading is fast when done in the user space. In the kernel This is useful when support for multiprocessors dual core is available. This was the OS schedules individual threads. However, it is slow as it requires a system call. Slightly more detailed notes available at http:

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

6: Process Excellence Network | Operations Management Systems: Stripping Out The Complexity

The basic unit of software that the operating system deals with in scheduling the work done by the processor is either a process or a thread, depending on the operating system.

A process table as displayed by KDE System Guard In general, a computer system process consists of or is said to own the following resources: An image of the executable machine code associated with a program. Operating system descriptors of resources that are allocated to the process, such as file descriptors Unix terminology or handles Windows , and data sources and sinks. Processor state context , such as the content of registers and physical memory addressing. The state is typically stored in computer registers when the process is executing, and in memory otherwise. The operating system keeps its processes separate and allocates the resources they need, so that they are less likely to interfere with each other and cause system failures e. The operating system may also provide mechanisms for inter-process communication to enable processes to interact in safe and predictable ways. Multitasking and process management[edit] Main article: Process management computing A multitasking operating system may just switch between processes to give the appearance of many processes executing simultaneously that is, in parallel , though in fact only one process can be executing at any one time on a single CPU unless the CPU has multiple cores, then multithreading or other similar technologies can be used. A process is said to own resources, of which an image of its program in memory is one such resource. However, in multiprocessing systems many processes may run off of, or share, the same reentrant program at the same location in memory, but each process is said to own its own image of the program. Processes are often called "tasks" in embedded operating systems. The sense of "process" or task is "something that takes up time", as opposed to "memory", which is "something that takes up space". If a process requests something for which it must wait, it will be blocked. All parts of an executing program and its data do not have to be in physical memory for the associated process to be active. Process state The various process states, displayed in a state diagram , with arrows indicating possible transitions between states. An operating system kernel that allows multitasking needs processes to have certain states. Names for these states are not standardised, but they have similar functionality. After that the process scheduler assigns it the "waiting" state. While the process is "waiting", it waits for the scheduler to do a so-called context switch and load the process into the processor. The process state then becomes "running", and the processor executes the process instructions. If a process needs to wait for a resource wait for user input or file to open, for example , it is assigned the "blocked" state. The process state is changed back to "waiting" when the process no longer needs to wait in a blocked state. Once the process finishes execution, or is terminated by the operating system, it is no longer needed. The process is removed instantly or is moved to the "terminated" state. When removed, it just waits to be removed from main memory. Inter-process communication When processes communicate with each other it is called "Inter-process communication" IPC. Processes frequently need to communicate, for instance in a shell pipeline, the output of the first process need to pass to the second one, and so on to the other process. It is preferred in a well-structured way not using interrupts. It is even possible for the two processes to be running on different machines. The operating system OS may differ from one process to the other, therefore some mediator s called protocols are needed. History of operating systems By the early s, computer control software had evolved from monitor control software , for example IBSYS , to executive control software. Over time, computers got faster while computer time was still neither cheap nor fully utilized; such an environment made multiprogramming possible and necessary. Multiprogramming means that several programs run concurrently. At first, more than one program ran on a single processor, as a result of underlying uniprocessor computer architecture, and they shared scarce and limited hardware resources; consequently, the concurrency was of a serial nature. On later systems with multiple processors , multiple programs may run concurrently in parallel. Programs consist of sequences of instructions for processors. A single processor can run only one instruction at a time: A program might need some resource , such as an input device, which has a large delay, or a program might start some slow operation, such as sending output to a printer. This would lead to processor being "idle" unused. To keep the

processor busy at all times, the execution of such a program is halted and the operating system switches the processor to run another program. To the user, it will appear that the programs run at the same time hence the term "parallel". Shortly thereafter, the notion of a "program" was expanded to the notion of an "executing program and its context". The concept of a process was born, which also became necessary with the invention of re-entrant code. Threads came somewhat later. However, with the advent of concepts such as time-sharing , computer networks , and multiple-CPU shared memory computers, the old "multiprogramming" gave way to true multitasking , multiprocessing and, later, multithreading.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

7: Linux Tutorial - Learn Process Management

The more fused or complex the operating system is, the more it is expected to do on behalf of its users. Even though its main concern is the execution of user programs, it also requires taking care of various system tasks which are better left outside the kernel itself.

The heart of managing the processor comes down to two related issues: The application you see word processor, spreadsheet or game is, indeed, a process, but that application may cause several other processes to begin, for tasks like communications with other devices or other computers. There are also numerous processes that run without giving you direct evidence that they ever exist. For example, Windows XP and UNIX can have dozens of background processes running to handle the network, memory management, disk management, virus checks and so on. A process, then, is software that performs some action and can be controlled -- by a user, by other applications or by the operating system. It is processes, rather than applications, that the operating system controls and schedules for execution by the CPU. In a single-tasking system, the schedule is straightforward. The operating system allows the application to begin running, suspending the execution only long enough to deal with interrupts and user input. Interrupts are special signals sent by hardware or software to the CPU. Sometimes the operating system will schedule the priority of processes so that interrupts are masked -- that is, the operating system will ignore the interrupts from some sources so that a particular job can be finished as quickly as possible. These non-maskable interrupts NMIs must be dealt with immediately, regardless of the other tasks at hand. While interrupts add some complication to the execution of processes in a single-tasking system, the job of the operating system becomes much more complicated in a multi-tasking system. Now, the operating system must arrange the execution of applications so that you believe that there are several things happening at once. This is complicated because the CPU can only do one thing at a time. In order to give the appearance of lots of things happening at the same time, the operating system has to switch between different processes thousands of times a second. A process occupies a certain amount of RAM. It also makes use of registers, stacks and queues within the CPU and operating-system memory space. When two processes are multi-tasking, the operating system allots a certain number of CPU execution cycles to one program. After that number of cycles, the operating system makes copies of all the registers, stacks and queues used by the processes, and notes the point at which the process paused in its execution. It then loads all the registers, stacks and queues used by the second process and allows it a certain number of CPU cycles. When those are complete, it makes copies of all the registers, stacks and queues used by the second program, and loads the first program. This content is not compatible on this device.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

8: The Importance of an Operating System | www.enganchecubano.com

Processes in the operating system can be in any of the following states: NEW - The process is being created. READY - The process is waiting to be assigned to a processor.

Not to be confused with a hardware component , a system component is similar to a computer program, but is not something an end-user directly interact with when using a computer. There are multiple system components at work in a computer operating system , each serving a specific function. Together, they allow the operating system and computer to function correctly and efficiently. Process Management The process management component is tasked with managing the many processes that are running on the operating system. Software programs each have one or more processes associated with them when they are running. For example, when you use an Internet browser , there is a process running for that browser program. The operating system also has many processes associated with it, each performing a different function. All of these processes are managed by process management, which keeps processes in order, running efficiently, using memory allocated to them, and shutting them down when necessary. Memory Management The memory management component, also sometimes called main memory management or primary memory management, handles primary memory, or RAM. When programs are running, including the operating system, those programs store data in RAM for quick access at any time. Memory management monitors and manages the memory and knows which blocks of memory are in use, which programs are using memory, and which memory blocks are available to be used. File Management The file management component manages just about anything to do with computer files. When a file is created, file management is involved in the creation of the file, including where it is stored on a storage device. When a file is modified, file management helps with the modification of the file. If a file is deleted, file management is there to help with deleting the file and freeing up the space for another file to be stored there at a later time. File management also handles tasks related to the creation, modification, and deletion of folders , or directories , on a storage device. Secondary Storage Management The secondary storage management component works with storage devices, like a hard drive , USB flash drive , DVD drive, or floppy disk drive. While the file management component takes care of the actual files on the storage device, the secondary storage management component manages the storage device itself. It manages the available space, or free space, on the storage device and allocating space for new files to be stored there. Requests for data on a storage device are handled by secondary storage management as well. For example, when a user double-clicks on a file to open it, secondary storage management receives that request and helps in the retrieval of that file from the storage device. Access Management The access management component is tasked with managing user access to data on a computer. User accounts provide each user with specific access to software, files, and functionality within an operating system. The ability to install a software program is controlled by access management. Access to view, edit, and delete a file is managed by access management. Changing settings within the operating system is managed by access management. How a user interacts with the computer operating system and uses software is handled by access management, in relation to the permissions they have been granted through user accounts. System Resource Management The system resource management component is responsible for managing the allocation of system resources, like memory and CPU time. When programs are running, they require the use of memory and CPU time to function properly. System resource management determines how much memory and CPU time that program is allowed to use at any given time. Managing system resource usage is a big responsibility, as it can directly impact the performance of the computer. If too many resources are allocated to one process, other programs and processes may become slow or unresponsive. If the operating system does not have enough resources allocated to it, the entire computer can run slow or stop working altogether. System resource management ensures system resources are allocated properly.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

9: Operating Systems â€“ Process Management | Computer Science: Source

A process contains its own independent virtual address space with both code and data, protected from other processes. Each process, in turn, contains one or more independently executing threads.

Bounds violation; for example: This includes determining the interleaving pattern for execution and allocation of resources to processes. One part of designing an OS is to describe the behaviour that we would like each process to exhibit. The simplest model is based on the fact that a process is either being executed by a processor or it is not. The process or some portion of it then exists in main memory , and it waits in the queue for an opportunity to be executed. From this model we can identify some design elements of the OS: The need to represent, and keep track of each process. The state of a process. This design does not make efficient use of the processor. The three states in this model are: The process that is currently being executed. A process that is queuing and prepared to execute when given the opportunity. At any instant, a process is in one and only one of the three states. For each of the three states, the process occupies space in main memory. While the reason for most transitions from one state to another might be obvious, some may not be so clear. Other reasons can be the imposition of priority levels as determined by the scheduling policy used for the Low Level Scheduler , and the arrival of a higher priority process into the READY state. A request to the OS is usually in the form of a system call, i. For example, requesting a file from disk or a saving a section of code or data from memory to a file on disk. Five-state process management model[edit] While the three state model is sufficient to describe the behavior of processes with the given events, we have to extend the model to allow for other possible events, and for more sophisticated design. In particular, the use of a portion of the hard disk to emulate main memory so called virtual memory requires additional states to describe the state of processes which are suspended from main memory, and placed in virtual memory on disk. Of course, such processes can, at a future time, be resumed by being transferred back into main memory. The Medium Level Scheduler controls these events. A process can be suspended for a number of reasons; the most significant of which arises from the process being swapped out of memory by the memory management system in order to free memory for other processes. Other common reasons for a process being suspended are when one suspends execution while debugging a program, or when the system is monitoring processes. For the five-state process management model, consider the following transitions described in the next sections. Note that this requires the state information concerning suspended processes be accessible to the OS. In that case, the OS designer may dictate that it is more important to get at the higher priority process than to minimise swapping. The PCB contains the basic information about the job including: This contains all of the information needed to indicate the current state of the job. This contains information used mainly for billing purposes and for performance measurement. It indicates what kind of resources the process has used and for how long. Processor modes[edit] Contemporary processors incorporate a mode bit to define the execution capability of a program in the processor. This bit can be set to kernel mode or user mode. Kernel mode is also commonly referred to as supervisor mode , monitor mode or ring 0. In kernel mode, the processor can execute every instruction in its hardware repertoire, whereas in user mode, it can only execute a subset of the instructions. Instructions that can be executed only in kernel mode are called kernel, privileged or protected instructions to distinguish them from the user mode instructions. The system may logically extend the mode bit to define areas of memory to be used when the processor is in kernel mode versus user mode. If the mode bit is set to kernel mode, the process executing in the processor can access either the kernel or user partition of the memory. However, if user mode is set, the process can reference only the user memory space. We frequently refer to two classes of memory user space and system space or kernel, supervisor or protected space. The mode bit is set by the user mode trap instruction, also called a supervisor call instruction. This instruction sets the mode bit, and branches to a fixed location in the system space. Since only system code is loaded in the system space, only system code can be invoked via a trap. When the OS has completed the supervisor call, it resets the mode bit to user mode prior to the return. This fundamental distinction is usually the irrefutable distinction between the operating system and other system software. The part of the system executing in kernel supervisor state is

called the kernel , or nucleus, of the operating system. The kernel operates as trusted software, meaning that when it was designed and implemented, it was intended to implement protection mechanisms that could not be covertly changed through the actions of untrusted software executing in user space. Extensions to the OS execute in user mode , so the OS does not rely on the correctness of those parts of the system software for correct operation of the OS. Hence, a fundamental design decision for any function to be incorporated into the OS is whether it needs to be implemented in the kernel. If it is implemented in the kernel, it will execute in kernel supervisor space, and have access to other parts of the kernel. It will also be trusted software by the other parts of the kernel. If the function is implemented to execute in user mode , it will have no access to kernel data structures. However, the advantage is that it will normally require very limited effort to invoke the function. While kernel-implemented functions may be easy to implement, the trap mechanism and authentication at the time of the call are usually relatively expensive. The kernel code runs fast, but there is a large performance overhead in the actual call. This is a subtle, but important point. Message passing Operating systems are designed with one or the other of these two facilities, but not both. First, assume that a user process wishes to invoke a particular target system function. For the system call approach, the user process uses the trap instruction. The idea is that the system call should appear to be an ordinary procedure call to the application program; the OS provides a library of user functions with names corresponding to each actual system call. Each of these stub functions contains a trap to the OS function. When the application program calls the stub, it executes the trap instruction, which switches the CPU to kernel mode , and then branches indirectly through an OS table , to the entry point of the function which is to be invoked. When the function completes, it switches the processor to user mode and then returns control to the user process; thus simulating a normal procedure return. In the message passing approach, the user process constructs a message, that describes the desired service. Then it uses a trusted send function to pass the message to a trusted OS process. The send function serves the same purpose as the trap; that is, it carefully checks the message, switches the processor to kernel mode, and then delivers the message to a process that implements the target functions. Meanwhile, the user process waits for the result of the service request with a message receive operation. When the OS process completes the operation, it sends a message back to the user process. The distinction between two approaches has important consequences regarding the relative independence of the OS behavior, from the application process behavior, and the resulting performance. As a rule of thumb, operating system based on a system call interface can be made more efficient than those requiring messages to be exchanged between distinct processes. This is the case, even though the system call must be implemented with a trap instruction; that is, even though the trap is relatively expensive to perform, it is more efficient than the message passing approach, where there are generally higher costs associated with process multiplexing , message formation and message copying. The system call approach has the interesting property that there is not necessarily any OS process. Instead, a process executing in user mode changes to kernel mode when it is executing kernel code, and switches back to user mode when it returns from the OS call. If, on the other hand, the OS is designed as a set of separate processes, it is usually easier to design it so that it gets control of the machine in special situations, than if the kernel is simply a collection of functions executed by users processes in kernel mode. Even procedure-based operating system usually find it necessary to include at least a few system processes called daemons in UNIX to handle situation whereby the machine is otherwise idle such as scheduling and handling the network.

WHAT IS PROCESS MANAGEMENT IN OPERATING SYSTEM pdf

Division of marital property Peein Off The Porch Disciplinary sanctions against students Consonant blends worksheets for grade 2 Basic civil engineering notes 1st year Chinese foreign policy think tanks and Chinas policy towards Japan Federal Aviation Administration Revitalization Act of 1995 Thoughts from Dad Open access to scientific and technical information Winter sketches from the saddle Contemporary assessment of child dietary intake in the context of the obesity epidemic Anthea M. Magarey, Marine electrical practice watson Involving Parents in Schools Places to Visit Level 3 (Early Readers from TIME For Kids (Early Readers) Its Going to Be Perfect (Picture Books) The rise of organicism Puedo Ser Bombero (I Can Be a Firefighter): I Can Be Books (I Can Be Books) V. 4. The dead terme, 1608. Worke for armourours, 1609. The rauens almanacke, 1609. A rod for run-awayes, Contemporary issues in parenting Towards the discovery of Canada Send Walter White Research questions on income security for sole parents NRSV Go-Anywhere Bible w/Apoc NuTone (tan/green) Let My Children Hear Music Kings of the Night Restructuring: American and Beyond An Ainu Legend of the Large Trout Creating web applications Sun One Studio 4, Community Edition Tutorial Canon 500d photography tutorial Race, income, and college in 25 years Democracy and legislation. Among the millet and other poems Sister of the birds, and other gypsy tales The Great Canadian Stripper Shortage Rebellion in the borderlands Jenni rivera book unbreakable Thomas Stonestreet of Birchden, Withyham, East Sussex, and of Charles County, Maryland, with his posterit Patient Or Profit Ibnu khaldun muqaddimah bahasa indonesia