Introduction

What is an Operating System?

A program that acts as an intermediary between a user and the hardware, it is between the application program and the hardware

What are the three main purposes of an operating system?

to provide an environment for a computer user to execute programs.
to allocate and separate resources of the computer as needed.
to serve as a control program: supervise the execution of user programs, management of the operation control of i/o devices

What does and Operating System Do?

Resource Allocator: reallocates the resources, manages all of the resources, decides between the requests for efficient and fair resource use.
Control Program: Controls the execution of programs to prevent errors and improper use of the computer

GOALS of the operating system

execute programs and make solving problems easier, make the computer system easy to use, use the computer hardware in an efficient manner.

What happens when you start your computer?

when you start your computer the bootstrap program is loaded at power-up or reboot. This program is usually stored in the ROM or the EROM generally known as *Firmware* . This program loads the operating system kernel and starts the execution. The one program running at all times is the *kernel*

What are interrupts and how are they used?

Introduction (cont)

an interrupt is an electronic signal , interrupts serve as a mechanism for process cooperation and are often used to control I/O, a program issues an interrupt to request the operating system support. The hardware requests an interrupt and then transfers the control to the interrupt handler, where the interrupt then ends.

The operating System Structure

the operating system utilizes multiprogramming. multiprogramming organizes jobs so that the CPU always has something to do, this allows no wasted time. in multiprogramming one job is selected and run via the job scheduling. when it is waiting the os switches to another job

How does the operating system run a program, What does it need to do?

1.) reserve machine time
2.) manually load urge program into memory
3.) load starting address and begin execution
4.) monitor and control execution of program from console

What is a process?

A process is a program in execution, it s active, while a program is passive. The program becomes the process when it is running.
The process needs resources to complete its task so it waits.
A process includes: a counter, a stack, and a data section.

What is process management?

The operating system is responsible for managing the processes. The os
1.) creates and deletes the user and system processes.
2.) suspends and resumes processes.
3.) provides mechanisms for process synchronization
4.) provides mechanisms for process communication
5.) provides mechanisms for deadlock handling

Problems that Processes run in to

The Producer and Consumer Problem

in cooperating processes the produce-consumer problem is common where the producer process produces information that is consumed by the consumer process

Producer and Consumer Explained

he Producer relies on the Consumer to make space in the data-area so that it may insert more information whilst at the same time, the Consumer relies on the Producer to insert information into the data area so that it may remove that information

examples of Producer - Consumer Problem

Client - Server paradigm, the client is the consumer and the server as the producer

Solution to the Producer- Consumer problem

the solution and producer processes must run concurrently, to allow this there needs to be an available buffer of items that can be filled by the producer and emptied by the consumer.
the producer can produce one item while the consumer is consuming another item.
the producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not been produced

Two types of buffers can be used

Unbounded buffer- no limit on the size of the buffer
Bounded buffer- there is a fixed buffer size, in this case the consumer must wait if the buffer is empty and the producer must wait if the buffer is full

Bounded Buffer Solution

By **Makaila Akahoshi** (makahoshi1)
cheatography.com/makahoshi1/

Published 21st October, 2015.
Last updated 13th May, 2016.

Problems that Processes run in to (cont)

The bounded buffer can be used to enable processes to share memory, in the example code the variable 'in' points to the next free position in the buffer. 'out' points to the first full position in the buffer.
The buffer is empty when in == out.
when (in+1)% buffer size == out then the buffer is full

CPU Scheduling

What is CPU scheduling?

The basis of multiprogrammed operating systems

What is the basic concept of CPU scheduling?

To be able to have a process running at all time to maximize CPU utilization. The operating system takes the CPU away from a process that is in wait, and gives the CPU to another process.

What is a CPU-I/O Burst Cycle?

The process execution cycle where the process alternates between CPU execution and I/O wait. Begins with CPU burst, then I/O burst, and then CPU burst, and so on. The CPU burst eventually ends with a system request to terminate execution.

What is a CPU Scheduler? (Also called short-term scheduler)

Carries out a selection process that picks a process in the ready que to be executed if the CPU becomes idle. It then allocates the CPU to that process.

When might a CPU scheduling decision happen?

CPU Scheduling (cont)

1) Switches from running to waiting state
2) Switches from running to ready state
3) Switches from waiting to ready state
4) Process terminates

The scheduling under 1 and 4 is nonpreemptive (or cooperative), otherwise it is preemptive. Preemptive: Priority to high priority processes, Nonpreemeptive: Running task is executed till completion and can not be interrupted.

Potential issues with preemptive scheduling?

1) Processes that share data: While one is in a state of updating its data, another process is given priority to run but can not read the data from the first process.

2) Operating system kernal: Another process might be given priority while the kernal is being utilized by another process. The kernal might be going through important data changes, leaving it in a vulnerable state. A possible solution is waiting for the kernal to return to a consistent state before starting another process.

What is the dispatcher?

It is a module that gives control of the CPU to the process selected by the CPU scheduler. This involves {{nl})a) switching context
b) switching to user mode
c) jumping to the proper location in the user program to restart that program.

It is involked in every process switch; the time the dispatcher takes to stop one process and start another is the dispatch latency.

Describe the Scheduling Criteria

CPU Scheduling (cont)

Various criterias used when comparing various CPU- scheduling algorithms.
a) CPU utilization: Keep the CPU as busy as possible. Ranges from 0 to 10%, usually ranges from 40% (lightly loaded system) to 90% (heavily loaded system).
b) Throughput: Measures the number of processes that are completed per time unit.
c) Turnaround time: Amount of time to execute a a particular process. It is the sum of time spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
d) Waiting time: The time spent waiting in the ready queue.
e) Response time: The amount of time it takes to produce a response after a submission of a request. Generally limited by the speed of the output device.

Best to maximize CPU utilization and throughput, and minimize turnaround time, waiting time, and response time, but can still vary depending on the task.

Describe the First-Come, First-Served scheduling algorithm

The process that requests the CPU first is allocated the CPU first. The **Gantt chart** illustrates a schedule of start and finish times of each process. The average waiting time is heavily dependent on the order of arrival of the processes. If a processes with longer burst time arrive first, the entire process order will now have a longer average wait time. This effect is called the **convoy effect**

Describe the short-job-first scheduling algorithm

By **Makaila Akahoshi**
(makahoshi1)
cheatography.com/makahoshi1/

Published 21st October, 2015.
Last updated 13th May, 2016.

CPU Scheduling (cont)

Associates processes by the length of their next CPU burst and gives CPU priority to the process with the smallest next CPU burst. If the next CPU burst of multiple processes are the same, First-come-first-serve scheduling is used. It is difficult to know the length of the next CPU request even though it is optimal over FCFS.

What is exponential averaging?

Uses the previous CPU bursts to predict future bursts. The formula is Tn+1 = α tn+ (1 - α) Tn. tn is the length of the nth CPU burst and Tn+1 is the predicted value of the next burst. α is a value from 0 to 1. If α = 0 the recent history has no effect and current conditions are seen as consistent. If α = 1 then only the most recent CPU burst matters. Most commonly α =1/2, where recent and past history are equally weighted. In a **shortest-remaining-time-first** exponential averaging, you line up the previous processes based on their burst times ascending instead.
Example of shortest-remaining-time-first exponential averaging: If T1 = 10 and α = 0.5 and the previous runs are 8,7,4,16.
T2=.5(4+10) =7
T3=.5(7+7)=7
T3=.5(8+7)=7.5
T4=.5(16+7.5)=11.25

What is priority scheduling?

CPU Scheduling (cont)

A priority number is assigned to each process based on its CPU burst. Higher burst gets a lower priority and vice versa.

Interally defined priority uses measurable qualities such as average I/O burst, time limits, memory requirements, etc.
externally defined priorities are criteria set not by the OS, mostly human qualities like the type of work, importance of the process in relation to business, amount of funds being paid, etc.

Preemptive priority will ask the CPU if the newly arrived process is higher priority than the currently running process. A **nonpreemptive priority** will simply put the new process at the head of the queue.

Potential problems with priority scheduling?

indefinite blocking (also called starvation): A process that is ready to run is left waiting indefinitely because the computer is constantly getting higher-priority processes. **Aging** is a solution where the priority of waiting processes are increased as time goes on.

Describe the Round-Robin scheduling algorithm

CPU Scheduling (cont)

Similar to first-come-first-serve, but each process is given a unit of time called the time quantum (usually between 10 to 100 milisecond), where the CPU is given to the next process after the time quantum(s) for the current process is over - regardless of if the process is finished or not. If the process is interrupted, it is prempted and put back in the ready queue. Depending on the size of the time quantum, the RR policy can appear like a first-come-first-serve policy or **processor sharing**, where it creates an appearance that each processor has its own processor because it is switching from one process to the next so quickly.

Turnaround time is dependent on the size of the time quantum, where the average turnaround time does not always improve as the time quantum size increased, but improves when most processes finish their CPU burst in a single time quantum. A rule of thumb is 80% of CPU bursts should be shorter than the time quantum in order to keep the context switches low.

Describe the multilevel queue scheduling

It is a method of scheduling algorithm that separates priority based on the type of processes in this order:
1) system processes
2)interactive proesses
3)interactive editing processes
4)batch processes
5)student processes

Each queue also has it's own scheduling alogorithm, so System processes could use FCFS while student processes use RR. Each queue has absolute priority over lower priority queues, but it is possible to time-slice among queues so each queue gets a certain portion of CPU time.

Describe a multilevel feedback queue scheduler

CPU Scheduling (cont)

Works similarly as the multilevel queue scheduler, but can separate queues further based on their CPU bursts. its parameters are:
The number of queues
The scheduling algorithm for each queue
The method used to determine when to upgrade a process
the method used to determine when to demote a process
The method used to determine which queue a process will enter when the process needs service

It is by definition the most general CPU-scheduling algorithm, but it is also the most complex.

Describe thread scheduling

User level threads: Managed by a thread library that the kernel is unaware of and is mapped to an associated kernel level thread, and runs on available light weight process. This is called **process-contention scope (PCS)** since it makes threads belonging to the same process compete for CPU. Priority is set by the programmer and not adjusted by the thread library

Kernel-level threads: Scheduled by the operating system, and uses the **system-contention scope** to schedule kernel threads onto a CPU, where competition for the CPU takes place among all threads in the system. PCS is done according to priority

Describe Pthread scheduling

The POSIX Pthread API allows for specifying either PCS or SCS during thread creation where PTHREAD_SCOPE_PROCESS schedules threads using PCS and PTHREAD_SCOPE_SYSTEM handles SCS. On systems with the many-to-many model, the PTHREAD_SCOPE_PROCESS policy schedules user-level threads onto LWPs, whereas the PTHREAD_SCOPE_SYSTEM creates and binds LWP for each user-level thread. Linux and Mac OS X only allow PTHREAD_SCOPE_SYSTEM.

CPU Scheduling (cont)

Describe how multiple-processor scheduling works

Multiple processes are balances between multiple processors through load sharing. One approach is through **asymmetric multiprocessing** where one processor acts as the master, is in charge of all the scheduling, controls all activites, and runs all kernel code, while the rest of the processors only run user code. **symmetric multiprocessing, SMP** has each processor self schedule throgh a common ready queue, or seperate ready queues for each processor. Almost all modern OSes support SMP.

Processors contain cache memory and if a process were switch from processor to another, the cache data would be invalidated and have to be reloaded. SMP tries to keep familiar processes on the same processor through **processor affinity**. Soft affinity attempts to keep processes on the same processor, but makes no guarantees. Hard affinity specifies that a process is not moved between processors.

Load balancing tries to balance the work done between processors so none sits idle while another is overloaded. **Push migration** uses a separate process that runs periodically and moves processes from heavily loaded processors onto less loaded ones.
Pull migration makes idole processors take processes from other processors. Push and pull are not mutually exclusive, and can also counteract possessor affinity if not carefully managed.

To remedy this, modern hardware designs implemented multithreaded processor cores in which two or more hardware threads are assigned to each core, so if one is stalling the core can switch to another thread.

Processes

Objective of multiprogramming

is to have some process running at all times, to maximize CPU utilization.

How does multiprogramming work?

several processes are stored in memory at one time, when one process is done and is waiting, the os takes the CPU away from that process and gives it to another process

Benefits of multiprogramming

higher throughput (amount of work accomplished in a given time interval) and increased CPU utilization

What is a process?

A process is a program in execution

What do processes *need*?

A process needs: CPU time, memory, files, and i/o devices

What are the Process States?

New: the process is being created
Ready: The process is waiting to be assigned to a processor
Waiting: The process is waiting for some event to occur
Running: instructions are being executed
Terminated: the process has finished execution

Why is the operating system good for resource allocation?

The operating system is good for resource allocation because it acts as hardware /software interface

What does a Process include?

1.) a program counter
2.) stack: contains temporary data
3.) data section: contains global variables

What is the Process Control Block?

Processes (cont)

processes are represented in the operating system by a PCB
The process control block includes:
1.) Process state
2.) program counter
3.) CPU registers
4.) CPU scheduling information
5.) Memory Management
6.) Accounting information
7.) I/O status

Why is the PCB created?

A process control block is created so that the operating system knows information on the process.

What happens when a program enters the system?

When a program enters the system it is placed in the queue by the queuing routine and the scheduler redirects the program from the queue and loads it into memory

Why are queues and schedulers important?

they determine which program is loaded into memory after one program finishes processes and when the space is available

What is a CPU switch and how is it used?

when the os does a switch it stops one process from executing (idling it) and allows another process to use the processor

What is process scheduling?

the process scheduler selects among the available processes for next execution on CPU

QUEUE

generally the first program on the queue is loaded first but there are situations where there are multiple queues,
1.) job Queue: when processes enter the system they are put into the job queue
2.) Ready Queue: the processes that are*ready* and *waiting* to execute are kept on a list (the ready queue)
3.) Device Queue: are the processes that are waiting for a particular i/o device (each device has its own device queue

Processes (cont)

How does the operating decide which queue the program goes to?

it is based on what resources the program needs, and it will be placed in the corresponding queue

What are the types of Schedulers?

1.) long term scheduler: selects which processes should be brought into the ready queue
2.) short term scheduler: selects which process should be executed next and then allocates CPU

What is a context switch?

a context switch is needed so that the CPU can switch to another process, in the context switch urge system saves the state of the process

Processes run concurrently

No two processes can be running *simultaneously* (at the same time) but they can be running *concurrently* where the CPU is multitasking

How are processes created?

The parent creates the child which can create more processes.
The child process is a duplicate of the parent process

fork()

fork creates a new process
when you run the fork command it either returns a 0 or a 1.
the 0 means that it is a child process
the 1 means that it is a parent process

execve()

the execve system call is used to assign a new program to a child.
it is used after the fork command to replace the process' memory space with a new program

Process Creation

Processes (cont)

every process has a process id, to know what process you are on and for process management every process has an id
very important when a process is created with the fork() only the shared memory segments are shared between the parent process and the child process, copies of the stack and the heap are made for the new child

Process Creation Continue

when a process creates a new process the parent can continue to run concurrently or the parent can wait until all of the children terminate

How are processes terminated?

A process terminates when it is done executing the last statement, when the child is terminated it may return data back to the parent through an exit status uses the *exit()* system call

Can a process terminate if it is not done?

Yes, the parent may terminate the child (*abort*) if:
the child has exceeded its usage of some of its resources it has been allocated
the task assigned to the child is no longer needed

wait() or waitpid()

these are the system call command that are used for process termination

Cascading Termination

some operating systems do not allow children to be alive if the parent has died, in this case if the parent is terminated, then the children must also terminate. this is known as cascading termination

Processes may be either *Cooperating* or *Independent*

By **Makaila Akahoshi** (makahoshi1)
cheatography.com/makahoshi1/

Published 21st October, 2015.
Last updated 13th May, 2016.

Processes (cont)

Cooperating: the process may be cooperating if it can affect or be affectedly the other processes executing in the system.
Some characteristics of cooperating processes include: state is shared, the result of execution is nondeterministic, result of execution cannot be predicted.
Independent: a process can be independent if it cannot be affected or affect the other processes.
Some characteristics of independent processes include: state not shared, execution is deterministic and depends on input, execution is reproducible and will always be the same, or if the execution can be stopped.

Advantages of Process Cooperation

information sharing, computation speed-up, modularity, convenience

What is Interprocess Communication

Cooperating processes need *interprocess communication* a mechanism that will allow them to exchange data and information
There are two models of IPC: shared memory and Message passing

What is Shared Memory?

a region of memory that is shared by cooperating processes is established, processes can exchange information by reading and writing to the shared region
Benefits of Shared Memory: allows maximum speed and convenience of communication and is faster than message passing.

What is Message Passing

Processes (cont)

message passing is a mechanism for processes to communicate and to synchronize their actions. processes communicate with each other without sharing variables
Benefits of message passing: message passing is easier to implement for inter computer communication and is useful for smaller amounts of data
Message passing can be either Blocking or Non-Blocking

Message Passing facilitates:

the message passing facility provides two operations:
send(message)- message size fixed or variable
receive(message)

How do processes P and Q communicate

for two processes to communicate they must:
1.) send messages to an receive messages from each other
2.) they must establish a communication link between them, this link can be implemented in a variety of ways.

Implementations of communication link include

1.) physical (ex. shared memory, hardware bus)
2.) logical (direct/indirect, synchronous/asynchronous, automatic/explicit buffering

Direct vs. Indirect Communication Links

Direct Communication Link: processes must name each other explicitly, they must state where they are sending the message and where they are receiving the message.
this can be either symmetric where they both name each other or asymmetric where only the sender names the receipient

Indirect Communication Link: messages are sent to and received from mailboxes or ports.

Properties of Direct Communication Link

Processes (cont)

1.) Links are established automatically
2.) A link is associated with one pair of communicating processes
3.) between each pair there exists exactly one link
4.) the link may be unidirectional or bidirectional (usually bidirectional)

Properties of Indirect Communication Links

1.) Link established only if processes share a mailbox
2.) a link may be associated with many processes
3.) each pair of processes may share several communication links
4.) link may be unidirectional or bidriectional

Message-Passing Synchronization

Message Passing may be either *blocking* or *non blocking*
Blocking is considered synchronous, sends and receives until a message is available/received
Nonblocking is considered asynchronous, the sender sends process and resumes operation, the receiver retrieves either a message or null

Buffering

In both direct and indirect communication messages exchanges are placed in a temporary queue. These queues are implemented in three ways
1.) zero capacity: has a max length of 0, the link cannot have any messages waiting in it. sender blocks until recepient receives
Bounded capacity: the queue has finite length n, at most n messages can be placed there. the sender must wait if link is full
Unbounded Capacity: the queues length is potentially infinite, any number of messages can wait in it. the sender never blocks

Other strategies for communication

Some other ways for communication include: Sockets, Remote Procedure Calls, and Pipes

Sockets

Processes (cont)

sockets is defined as an endpoint for communication, need a pair of sockets-- one for each process.
A socket is defined by an IP address concatenated with a port number

Remote Procedure Controls

A way to abstract the procedure-call mechanism for use between systems with network connections.
the RPC scheme is useful in implementing a distributed file system

Pipes

A pipe acts as a conduit allowing two processes to communicate. Pipes were one of the first IPC mechanisms and provided one of the simpler ways for processes to communicate with one another, there are however limitations

Ordinary Pipes

allow communication in standard producer-consumer style
ordinary pipes are unidirectional
an ordinary pipe cannot be accessed from outside the process that creates it. typically a parent process creates a pipe and uses it to communicate with a child process
only exists while the processes are communicating

Named Pipes

more powerful than ordinary pipes
communication can be bidirectional
no parent- child relationship is required
once a name pipe is established several processes can be used for named pipes

Threads

What is a thread?

A basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers. They also form the basics of multithreading.

Benefits of multi-threading?

Responsiveness: Threads may provide rapid response while other threads are busy.
Resource Sharing: Threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
Economy: Creating and managing threads is much faster than performing the same tasks for processes.
Scalability: A single threaded process can only run on one CPU, whereas the execution of a multi-threaded application may be split amongst available processors.

Threads (cont)

Multicore Programming Challenges

Dividing Tasks: Examining applications to find activities that can be performed concurrently.
Balance: Finding tasks to run concurrently that provide equal value. I.e. don't waste a thread on trivial tasks.
Data Splitting: To prevent the threads from interfering with one another.
Data Dependency: If one task is dependent upon the results of another, then the tasks need to be synchronized to assure access in the proper order.
Testing and Debugging: Inherently more difficult in parallel processing situations, as the race conditions become much more complex and difficult to identify.

Multithreading Models

Many-To-One: Many user-level threads are all mapped onto a single kernel thread.
One-To-One: Creates a separate kernel thread to handle each user thread. Most implementations of this model place a limit on how many threads can be created.
Many-To-Many: Allows many user level threads to be mapped to many kernel threads. Processes can be split across multiple processors. Allows the OS to create a sufficient number of kernel threads.

Threads (cont)

Thread Libraries

Provide programmers with an API for creating and managing threads. Implemented either in User Space or Kernel Space.
User Space: API functions are implemented solely within user space. & no kernel support.
Kernel Space: Involves system calls and requires a kernel with thread library support.

Three main thread libraries:

POSIX Pthreads: Provided as either a user or kernel library, as an extension to the POSIX standard.
Win32 Threads: Provided as a kernel-level library on Windows systems.
Java Threads: Implementation of threads is based upon whatever OS and hardware the JVM is running on, i.e. either Pthreads or Win32 threads depending on the system.

Pthreads

*POSIX standard defines the specification for pThreads, not the implementation.
* Global variables are shared amongst all threads.
*One thread can wait for the others to rejoin before continuing.
*Available on Solaris, Linux, Mac OSX, Tru64, and via public domain shareware for Windows.

Java Threads

*Managed by the JVM
*Imlemented using the threads model provided by underlying OS.
*Threads are created by extending thread class and by implementing the Runnable interface.

Thread Pools

Threads (cont)

A solution that creates a number of threads when a process first starts, and places them into a thread pool to avoid inefficient thread use.
* Threads are allocated from the pool as needed, and returned to the pool when no longer needed.
* When no threads are available in the pool, the process may have to wait until one becomes available.
* The max. number of threads available in a thread pool may be determined by adjustable parameters, possibly dynamically in response to changing system loads.

Threading Issues

Threads (cont)

The fork( ) and exec( ) System Calls
Q: If one thread forks, is the entire process copied, or is the new process single-threaded?
*A: System dependent.
*A: If the new process execs right away, there is no need to copy all the other threads. If it doesn't, then the entire process should be copied.
*A: Many versions of UNIX provide multiple versions of the fork call for this purpose.
Signal Handling used to process signals by *generating* a particular event, *delivering* it to a process, and *handling* it.)
Q: When a multi-threaded process receives a signal, to what thread should that signal be delivered?
A: There are four major options:
*Deliver the signal to the thread to which the signal applies.
* Deliver the signal to every thread in the process.
*Deliver the signal to certain threads in the process.
* Assign a specific thread to receive all signals in a process.
Thread Cancellation can be done in one of two ways
*Asynchronous Cancellation: cancels the thread immediately.
*Deferred Cancellation: sets a flag indicating the thread should cancel itself when it is convenient. It is then up to the cancelled thread to check this flag periodically and exit nicely when it sees the flag set.
Scheduler Activations
Provide Upcalls, a communication mechanism from the kernel to the thread library. This communication allows an application to maintain the correct number kernel threads.

Synchronization

Background

> Concurrent access to shared data may result in data inconsistency. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

Race Condition

> A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access take place.

Critical Section

> Each process has a critical section segment of code. When one process is in critical section, no other may be in its critical section.

Parts of Critical Section

> Each process must ask permission to enter critical section in **entry section**, may follow critical section with **exit section**, then **remainder section**.

Solutions to Critical Section Problem

> The three possible solutions are **Mutual Exclusion**, **progress**, and **bounded waiting**.

Mutual Exclusion

> If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

Progress

> If no process is executing in its critical section and there exists some processes that wish to execute their critical section, then the selection of the process that will enter the critical section cannot be postponed indefinitely.

By **Makaila Akahoshi**
(makahoshi1)
cheatography.com/makahoshi1/

Published 21st October, 2015.
Last updated 13th May, 2016.

Synchronization (cont)

Bounded Waiting

A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Peterson's Solution

Two process solution. Assume that LOAD and STORE instructions are atomic; that is, cannot be interrupted. The two processes share two variables: int turn and Boolean flag[2]. Turn indicates whose turn it is to enter the critical section. The flag array is used to indicate if a process is ready to enter the critical section.

Synchronization Hardware

Many systems provide hardware support for critical section code. Modern machines provide special atomic hardware instructions.

Semaphore

A semaphore is a synchronization tool that does not require busy waiting. You can have a counting semaphore or a binary semaphore. Semaphores provide mutual exclusion.

Semaphore Implementation

When implementing semaphores you must guarantee that no two processes can execute **wait ()** and **signal ()** on the same semaphore at the same time.

Deadlock

Deadlock is when two or more processes are waiting indefinitely for an even that can only be caused by one of the waiting processes.

Starvation

Indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.

Synchronization (cont)

Bounded Buffer Problem

The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

Readers Writers Problem

The problem is that you want multiple readers to be able to read at the same time but only one single writer can access the shared data at a time.

Dining Philosophers Problem

Monitors

A high level abstraction. *Abstract data type*, internal variables only accessible by code within the procedure. Only one process may be active within the monitor at a given time.

Race Condition

A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access take place.

Semaphore

Semaphore

A generalization of a spin-lock process, a more complex kind of mutual exclusion
The semaphore provides mutual exclusion in a protected region for groups of processes, not just one process at a time

Why are semaphores used?

Semaphore (cont)

Semaphores are used in cases where you have n amount of processes in a critical section problem

How do you initialize a semaphore

You initialize a semaphore with the semaphore_init --> the shared int sem holds the semaphore identifier

Wait and Signal

A simple way to understand wait (P) and signal (V) operations is: wait: If the value of semaphore variable is not negative, decrements it by 1. If the semaphore variable is now negative, the process executing wait is blocked (i.e., added to the semaphore's queue) until the value is greater or equal to 1. Otherwise, the process continues execution, having used a unit of the resource. signal: Increments the value of semaphore variable by 1. After the increment, if the pre-increment value was negative (meaning there are processes waiting for a resource), it transfers a blocked process from the semaphore's waiting queue to the ready queue.

How do we get rid of the busy waiting problem?

Rather than busy waiting the process can *block* itself, the block operation places a process into a waiting queue and the state is changed to waiting

Signal

A process that is blocked can be waked up which is done with the signal operation that removes one process from the list of waiting processes, the wakeup resumes the operation of the blocked process.
removes the busy waiting from the entry of the critical section of the application

in busy waiting there may be a negative semaphore value

Deadlocks and Starvation

By **Makaila Akahoshi** (makahoshi1)
cheatography.com/makahoshi1/

Published 21st October, 2015.
Last updated 13th May, 2016.