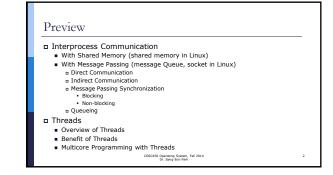
Process Model -real model, virtual model Process Creation Process Creation Process Termination -voluntary, involuntary Process Table (Process Control Block) Process Table (Process Control Block) Process Scheduling Process Scheduling Context Switch Process Creation in Linux Process Creation in Linux Process Creation in Linux Process Creation in Linux

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park



Interprocess Communication

Android Process Hierarchy

- □ Processes executing concurrently in the operating system may be either independent processes or cooperating processes.
- □ Cooperating processes require an <u>interprocess communication</u> (IPC) <u>mechanism</u> that will allow them to exchange data.
- Reasons for providing an environment that allows <u>process</u> <u>cooperation</u>:
 - Information Sharing- several processes or threads can share information (client-sever)
 - Computation Speed up jobs are divided and several process or threads run
 on different part of job on different CPU core. Eventually create final result.
 (rendering)
 - Modularity construct the system in a modular fashion, dividing the system functions into separate processes or threads
 CONCORD DEPARTS FAMILY AND ADDRESS AND A

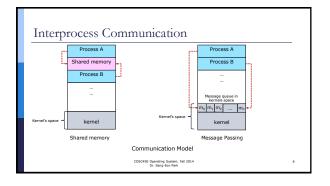
Interprocess Communication

- □ There are two fundamental models of interprocess communication:
 - Shared Memory- a region of memory is shared by processes with read /write operations. It is useful for exchanging smaller amount of data since no conflicts need be avoided. <u>OS only involved in creation but not any synchronization or mutal exclusion!</u>
 - Message Passing communication takes place by means of messages exchanged between the cooperating processes (Message Queue, Socket). It is also easier to implement in a distributed system than shared memory. OS involved in creation and synchronization and mutual exclusion!

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

- Shared memory can be faster than message passing, since message-passing systems are typically implemented using system calls (shared memory are located in user's space).
- □ In shared-memory systems, system calls are <u>required</u> <u>only to establish shared memory regions</u>.
- Once shared memory is established, all accesses are treated as routine memory accesses, without kernel's assistance.



Interprocess Communication

(Shared Memory)

- A process can create a shared-memory segment in RAM for interprocess communication.
- Other processes that wish to communicate using this shared-memory segment must attach it to their address space by key values.
- Normally, the OS prevent one process from accessing another process's memory but OS remove this restriction for shared-memory since it is not belong to kernel's space. <u>OS only involved in creation of shared memory</u>. Then, several process can exchange information by reading and writing data in the shared
- The form of the data and the location are determined by these processes and are not under the operating system's control.
- □ The processes are also <u>responsible for mutual exclusion</u> for writing to avoiding race condition.

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Shared Memory)

- □ Before using the shared memory what we needs to be done with the system calls,
 - Create the shared memory segment or use an already created shared memory segment (shmget())
 - Attach the process to the already created shared memory segment (shmat())
 - Detach the process from the already attached shared memory segment (shmdt()).
 - Control operations on the shared memory segment (shmctl())

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Shared Memory)

producer consumer problem with shared memory

- □ To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- $\ensuremath{\mathbf{n}}$ This buffer will reside in a region of memory that is shared by the producer and consumer processes. A 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 yet been produced. (Both producer and consumer must be written with considering mutual exclusion to avoid race condition)

COSC450 Operating System, Fall 2014 Dr. Sano-Eon Park

Interprocess Communication Shared memory Consumer kernel Produce-Consumer with Shared Memory COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Message-Passing)

- $\, {\rm \square} \,$ OS provide the means for interprocess communication via a $\underline{{\rm message-passing}}$ facility (message queue, socket).
- Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.
- It is particularly <u>useful in a distributed environment</u>, where the communicating processes may reside on different computers connected by a network.
- □ Two operation are provided as library or system calls.
- Send(message)
- To communicate between processes, a communication link must exist between them. This link can be implemented in various ways based on mean (message queue, socket, ...)

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Message-Passing: Message Queue)

- □ A message gueue is a linked list of message stored within the kernel's space and identified by a message queue ID.
- □ A new message queue is created or opened by msgget().
- □ A new messages are added to the end of a queue by msgsnd().
- □ Messages are fetched from a queue by msgrcv().
- □ The msgctl() system call provides a variety of message control operations as specified by cmd.
 - IPC_STAT, IPC_SET, IPC_RMID

Interprocess Communication

(Message-Passing)

- □ Logical Methods for implementing a link and send()/ receive operations.
 - Direct or indirect communication
 - Synchronous or asynchronous communication
 - Automatic or explicit buffering
- □ Under direct communication, each processes must know end point address for sending or receiving message between.
- Under indirect communication, the messages are sent to and received from mailboxes, or ports.

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Message-Passing: Direct Communication)

- under direct communication, each processes must know end point address for sending or receiving message between.
 - send (P, message) send a message to process with end point address P.
 - receive(Q, message) receive a message from process with end point address Q.
- A communication link properties in direct communication
 - A link is established between every pair of processes to communicate. (each process in pair knows each identity (by ex. IP + port number).
 - A link is associated with exactly two processes.
 - There is one link between each pair of process (peer to peer)

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Message-Passing: Direct Communication)

- □ Direct communication method exhibits
- Symmetry in addressing since both processes know end point address each other (Full Dunley)
 - address each other (Full Duplex).

 a send (P, message) send a message to process with end point address P.

 a receive(Q, message) receive a message from process with end point address Q
- Asymmetry in addressing since only one processes know end point address (Half Duplex).
 - send (P, message) send a message to process with end point address P.
 receive(id, message) receive a message from any process.

COSC450 Operating System, Fall 2014 Dr. Sano-Eon Park

System, Fall 2014 Eon Park

Interprocess Communication

(Message-Passing: Indirect Communication)

- With indirect communication, the messages are sent to and received from mailboxes, or ports.
- Each mailbox has a unique identification. For example, POSIX message queues use an integer value to identify a mailbox
- A process can communicate with another process via a number of different mailboxes, but two processes can communicate only if they have a shared mailbox.

COSC450 Operating System, Fall 2014 Dr. Sano-Eon Park 1

Interprocess Communication

(Message-Passing: Indirect Communication)

Operations

- create (A) create a mailbox (in POSIX with ftok(), msqqet())
- send(A, message)—Send a message to mailbox A. (in POSIX with msgsnd())
- receive(A, message)—Receive a message from mailbox A.(in POSIX with msgrcv())
- remove (A) remove mailbox (in POSIX with msgctl())
- □ communication link properties in indirect communication
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes
 - Each pair of processes may share several communication links

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park 17

Interprocess Communication

(Message-Passing: Indirect Communication)

- $\ensuremath{\square}$ Since a mailbox can be shared by several processes, we need consider mailbox sharing.
 - $\,\blacksquare\,$ Let's assume process $P_1,\,P_2$ and P_3 share mailbox A
 - P₁ send a message to the mailbox.
 - P₂ and P₃ try to receive message from the mailbox.
- Who gets the message?
- Possible solutions
 - Allow a link to be associated with at most two processes
 - Allow only one process at a time to execute a receive operation
 - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

18

Interprocess Communication

(Message-Passing: Synchronization)

- Message passing may be either blocking or non-blocking
- Blocking is considered synchronous (TCP or SCTP with socket)
- Blocking send -- the sender is blocked until the message is received
- Blocking receive -- the receiver is blocked until a message is available ■ Non-blocking is considered asynchronous (UDP with socket)
 - Non-blocking send -- the sender sends the message and continue
 - Non-blocking receive -- the receiver receives:
 - A valid message, or
 Null message
- Different combinations possible
 - If both send and receive are blocking, we have a rendezvous

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Interprocess Communication

(Message-Passing: Buffering)

- □ Queue of messages attached to the link.
- □ Implemented in one of three ways
 - Zero capacity no messages are queued on a link.
 Sender must wait for receiver (rendezvous)
 - Bounded capacity finite length of n messages
 Sender must wait if link full
 - Unbounded capacity infinite length Sender never waits

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Overview of Thread

- □ Most software run on modern computers or mobile devices are multithreaded.
- □ An application typically is implemented as a separate process with several threads of control.
- Each threads are run on different part of a process.
- □ Each thread consist of a thread ID, a program counter (PC), a register set, and a stack.
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

Overview of Thread code data files data files stack PC Single-threaded process Multi-threaded process COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

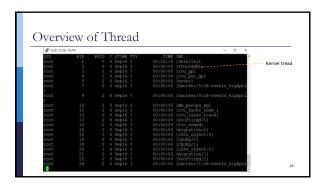
Overview of Thread

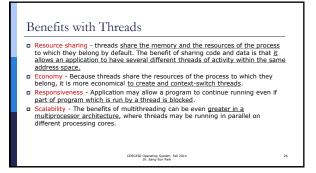
- Multi-threaded software example
 - Web browser: one thread display images or text, another thread retrieves data from the network.
 - A word processer: a thread for displaying graphics, a thread for responding to keyboard keystrokes, a thread performing spelling and grammar checking in the background.
 - A web server: for each client's request, server create a thread to take care one client request.

COSC450 Operating System, Fall 2014 Dr. Sang-Eon Park

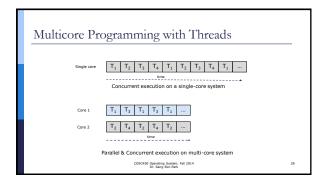
Overview of Thread

- Most operating system kernels are also typically multithreaded.
- As an example, during system boot time on Linux systems, several kernel threads are created.
 - Each thread performs a specific task, such as managing devices, memory management, or interrupt handling.
- $\ensuremath{\blacksquare}$ The command $\ensuremath{\mathsf{ps}}$ -ef can be used to display the kernel threads on a running Linux system.
- Examining the output of this command will show the kernel thread kthreadd (with pid = 2), which serves as the parent of all other kernel threads.





Multicore Programming with Threads Multithreaded programming with multicore CPU provides a improved concurrency (some threads can run in parallel). Consider an application with four threads run on a system with a single core CPU. Since the single core in CPU can take care one thread at a time, need context switch between threads to support concurrency. Consider an application with four threads run on a system with two-core CPU. Some threads can run in parallel in this system. A concurrent system supports more than one task by allowing all the tasks to make progress. In contrast, a parallel system can perform more than one task simultaneously.



Multicore Programming with Threads OS and application developer's challenges with Multi-core or multiple CPU. Designers of operating systems must write scheduling algorithms that use multiple processing cores to allow the parallel execution. For application programmers, the challenge is to modify existing programs as well as design new programs that are multithreaded.

Multicore Programming with Threads Five areas present challenges in programming for multicore systems: 1. Identifying tasks – examining applications to find areas that can be divided into separate concurrent tasks. 2. Balance – applications are divided into multiple tasks with balanced working load 3. Data Splitting -Just as applications are divided into separate tasks, the data accessed and manipulated by the tasks must be divided to run on separate cores. 4. Data Dependency – If there are data dependency between tasks, programmer need consider synchronization for avoid race condition. 5. Testing and debugging -Testing and debugging multi-threaded programs is more difficult than single threaded programs

Multicore Programming with Threads (Types of Parallelism) Data parallelism focuses on distributing subsets of the same data across multiple computing cores and performing the same operation on each core. Ex) summation of array size 2N A thread on core, (or CPU₀) sum the elements [0] ... [N-1] A thread on core, (or CPU₁) sum the elements [N] ... [2N-1] Task parallelism involves distributing tasks (threads) across multiple computing cores. Each thread is performing a unique operation. Different threads may be operating on the same data, or they may be operating on different data

